Prediction du prix du vin pour 2015

Antoine
Tixier

2015

Table des matières

Packages utilisés

```
setwd(wd)
# sélection
libs = c("forecast", "tseries", "locfit", "sm", "doParallel", "parallel", "xlsx",
    "knitr")
# installation
lapply(libs, install.packages, character.only = TRUE, repos = "http://cran.cnr.Berkeley.edu")
# chargement
lapply(libs, library, character.only = TRUE)
```

Fonction personnalisée

Cette fonction diagnostique les résidus (blancheur et distribution) d'une série temporelle après modélisation, permettant de déterminer l'adéquation du modèle.

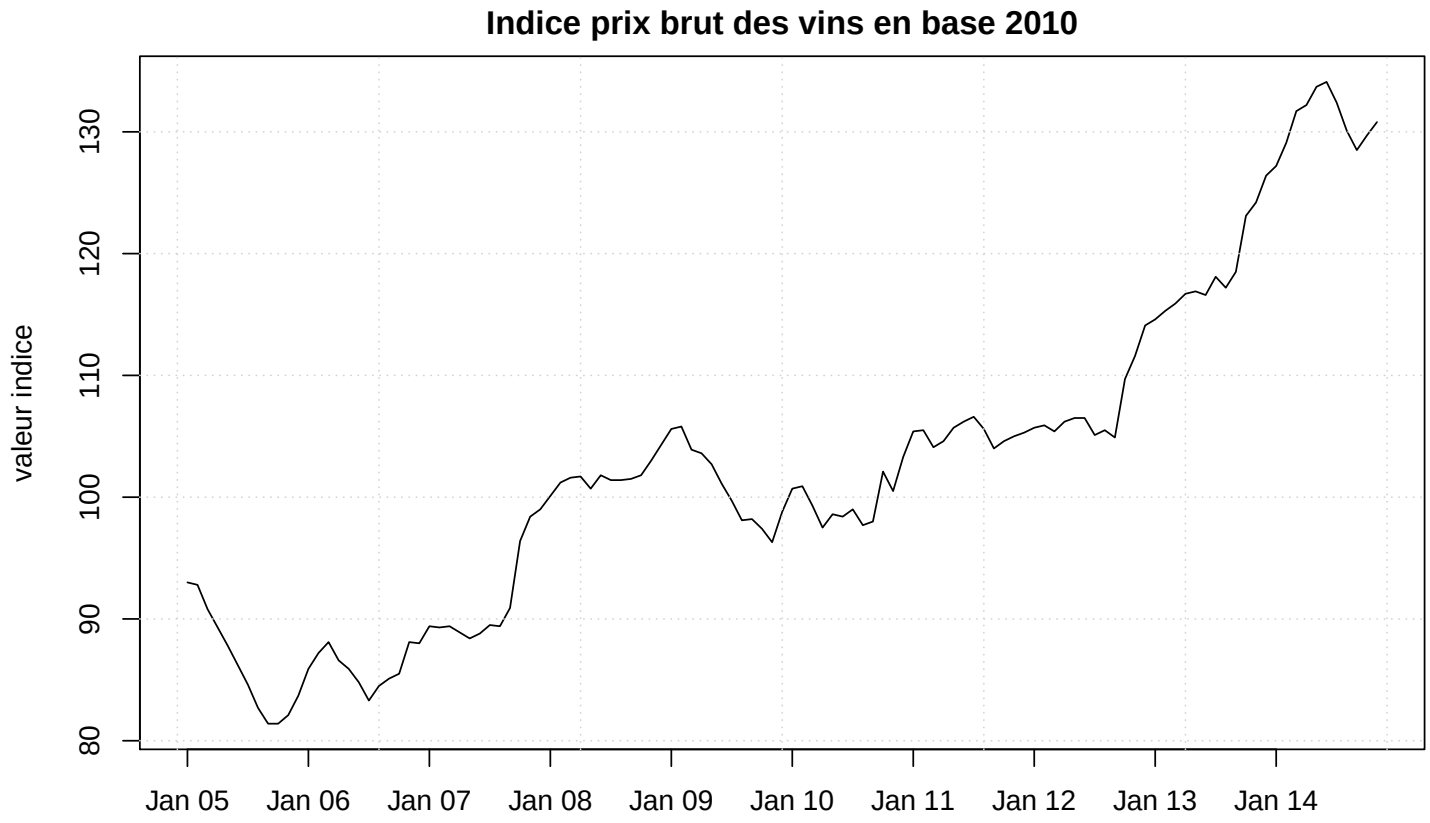
```
check.residuals.ts = function(x) {

    # affichage simultané de 4 graphiques
    par(mfrow = c(2, 2))

    # marges optimisées
    par(mar = c(2, 2.5, 2.5, 1))

    # 1) résidus
    plot(x, main = "résidus", xlab = "", ylab = "")
    abline(h = 0)

    # 2) histogramme (distribution empirique) des résidus avec Kernel smoother
    hist(x, probability = TRUE, main = "distribution des résidus", xlab = "",
        ylab = "")
    sm.density(x, add = T, col = "red")

    # 3) diagramme quantile-quantile avec distribution Normale
    qqnorm(x, main = "Normal Q-Q plot des résidus", xlab = "", ylab = "")
    qqline(x, col = "red")

    # 4) lag-plot des résidus
    plot(x[-length(x)], x[-1], main = "lag-plot des résidus", xlab = "", ylab = "")

    par(mfrow = c(2, 1))
    par(mar = c(5, 2.5, 3, 1.5))
    # fonctions d'autocorrélation
    acf(x, main = "fonction d'autocorrélation des résidus", ylab = "")
    pacf(x, main = "fonction d'autocorrélation partielle des résidus", ylab = "")

    # retour à l'affichage défaut
    par(mfrow = c(1, 1))
    par(mar = c(5.1, 4.1, 4.1, 2.1))

    # portmanteau test: l'hypothèse nulle est que les résidus sont indépendents
    # (pas de correlation)
    cat("Box-Ljung test p-value:", Box.test(x, type = "Ljung")$p.value)

    # Kruskal-Wallis test: l'hypothèse nulle est que les résidus suivent une loi
    # Normale
    cat(". Kruskal-Wallis test p-value:", ks.test(x, pnorm, alternative = "two.sided")$p.value)

}

dump("check.residuals.ts", file = "check.residuals.ts.R")
source("check.residuals.ts.R")
```

Données

Chargement et affichage

L'indice des prix des produits agricoles à la production (IPPAP) mesure l'évolution des prix des produits vendus par les agriculteurs. Cet indice est élaboré à partir de l'observation des prix (bruts) du marché. Ici on considère les valeures mensuelles de l'IPPAP pour les vins de Janvier 2005 à Novembre 2014. Les données peuvent être téléchargées sur cette page.

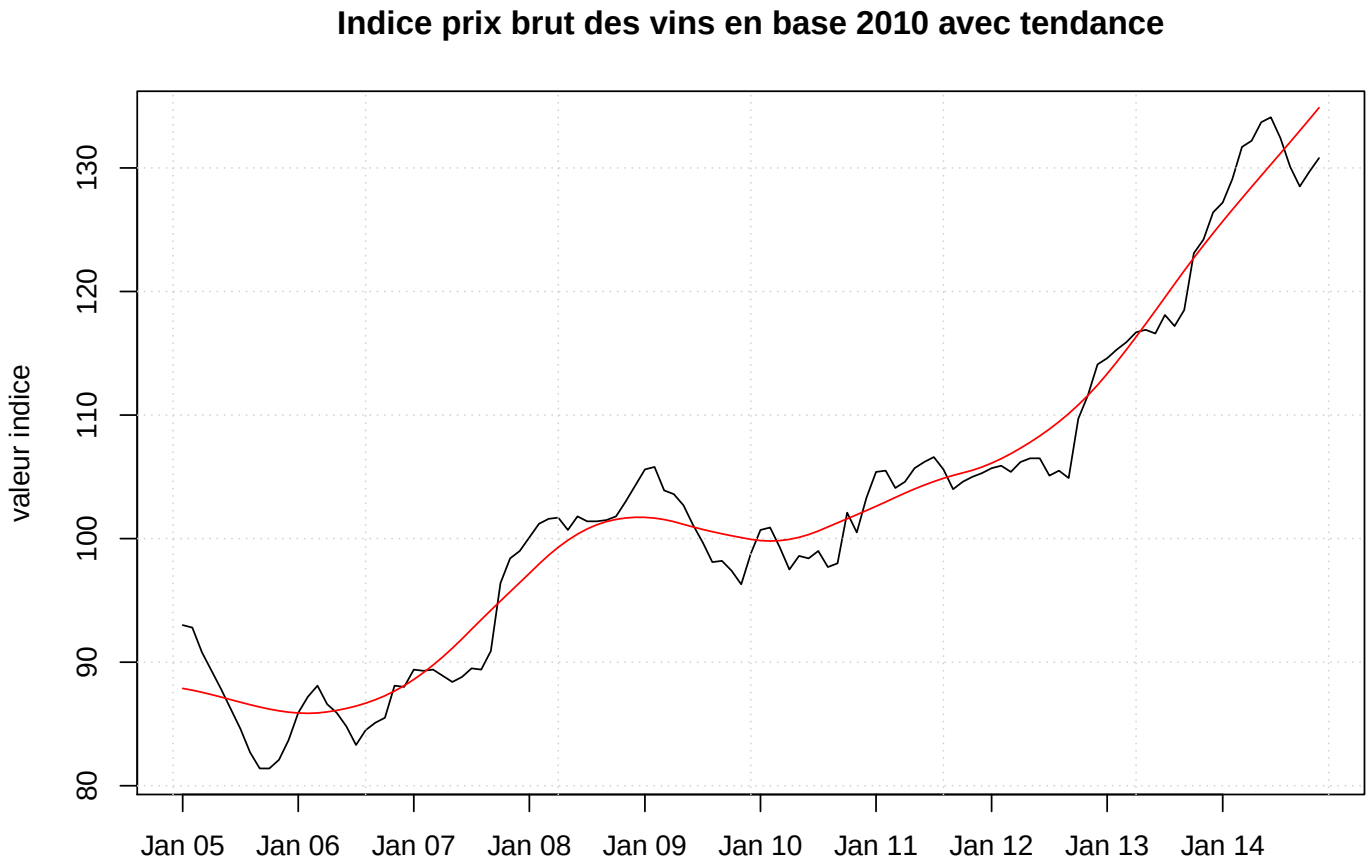
```
# déclaration du working directory
setwd(wd)

# lecture des données
data=read.xlsx2("wine.price.xlsx",sheetName="Valeurs",startRow=4,endRow=122,header=FALSE,
colIndex=c(1:3),colClasses=c(rep("numeric",3)))

# réarrangement des observations dans l'ordre chronologique
wine=data
for (i in 1:dim(data)[1]){
  wine[i,]=data[dim(data)[1]-(i-1),]
}
colnames(wine)=c("année","mois","index")

# visualisation du résultat
head(wine)

  année mois index
1  2005    1  93.0
2  2005    2  92.8
3  2005    3  90.8
4  2005    4  89.3
5  2005    5  87.8
6  2005    6  86.2

tail(wine)

    année mois index
114  2014    6 134.1
115  2014    7 132.4
116  2014    8 130.1
117  2014    9 128.5
118  2014   10 129.7
119  2014   11 130.8

price=wine[,"index"]

# optimisation des paramètres d'affichage
par(mar=c(2.5,4,2,0.5))

# affichage de la série
plot(price,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Indice prix brut des vins en base 2010")
axis(1,at=which(wine[,"mois"]==1),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14"))
grid()
```



Détermination de la tendance

On ajuste une tendance par la méthode des polynômes locaux (LOESS).

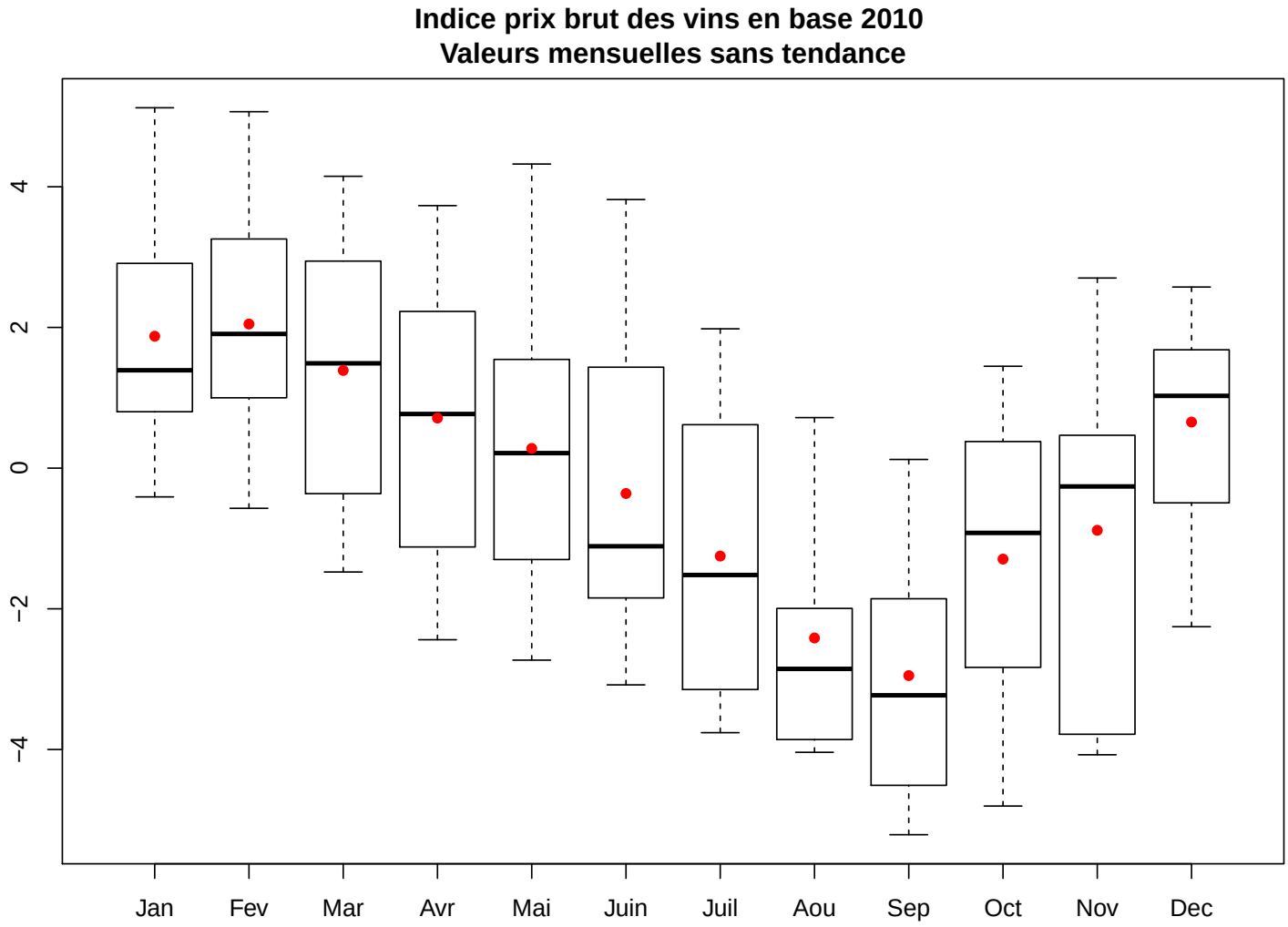
```
local=locfit(price~time(price),alpha=0.3,deg=1,kern="bisq",scale=TRUE)
# visualisation
plot(price,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Indice prix brut des vins en base 2010 avec tendance")
axis(1,at=which(wine[,"mois"]==1),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14"))
grid()
lines(x=time(price),y=fitted(local),col="red")
```



Détermination de la saisonnalité

En extrayant la tendance de la série, on pourra observer une éventuelle saisonnalité.

```
# retranchement de la tendance
price.detrended=price-fitted(local)
# regroupement des valeurs par mois
monthly=as.data.frame(matrix(ncol=12,nrow=10))
colnames(monthly)=c("Jan","Fev","Mar","Avr","Mai","Juin","Juil","Aou","Sep","Oct","Nov","Dec")
for (i in 1:11){
monthly[,i]=price.detrended[which(wine[,2]==i)]
}
monthly[,12]=c(price.detrended[which(wine[,2]==12)],median(price.detrended[which(wine[,2]==12)]))
# visualisation sous formes de boxplots mensuelles
par(mar=c(2.5,2.5,3,0.5))
boxplot(monthly,xlab="mois",ylab="",main="Indice prix brut des vins en base 2010
Valeurs mensuelles sans tendance")
# rajout des moyennes mensuelles
points(x=1:12,y=apply(monthly,2,mean),col="red",pch=16)
```



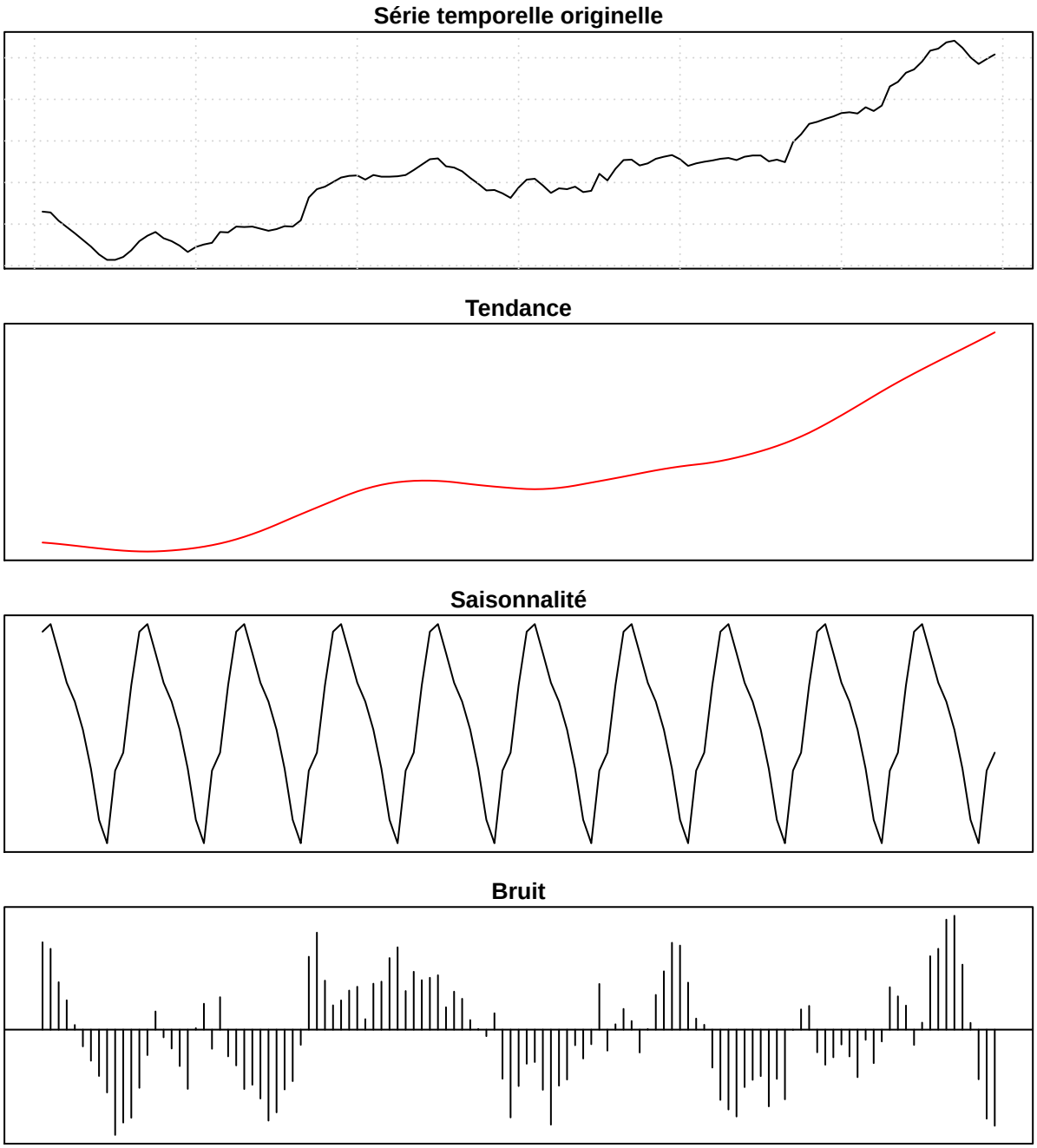
On observe une saisonnalité : les prix sont à leur maximum en hiver et diminuent ensuite jusqu'à atteindre un minimum en été. Au cours de l'automne, les prix remontent.

Décomposition tendance-saisonnalité-bruit

Le modèle additif donne la relation : $y_t = T_t + S_t + E_t$, où T désignes la tendance, S le terme saisonnier, et E le terme d'erreurs stochastiques. La série désaisonnalisée est donnée par $z_t = y_t - (s_t - \bar{s})$. Ici, s_t est la moyenne pour le mois t (t va de 1 à 12), et $\bar{s}$ est la moyenne des moyennes mensuelles.

```
# moyennes mensuelles
monthly.means = apply(monthly, 2, mean)
# moyenne globale
overall.mean = mean(monthly.means)
# retrancher la tendance et la saisonalité donne les erreurs
price.detrended.deseasonalized = price
for (i in 1:length(price)) {
    k = wine[i, 2]  # numéro du mois
    price.detrended.deseasonalized[i] = price.detrended[i] - (monthly.means[k] -
        overall.mean)
}
# on obtient finalement les termes saisonniers
price.season = price - fitted(local) - price.detrended.deseasonalized
# visualisation
par(mfrow = c(4, 1))
par(mar = c(1, 1, 1.5, 1.5))
plot(price, type = "l", xlab = "", xaxt = "n", ylab = "", yaxt = "n", main = "Série temporelle originelle")
grid()
plot(fitted(local), type = "l", xlab = "", xaxt = "n", ylab = "", yaxt = "n",
    col = "red", main = "Tendance")
plot(price.season, type = "l", xlab = "", xaxt = "n", ylab = "", yaxt = "n",
    main = "Saisonnalité")
plot(price.detrended.deseasonalized, type = "h", xlab = "", xaxt = "n", ylab = "",
    yaxt = "n", main = "Bruit")
```

```r
abline(h = 0)
```



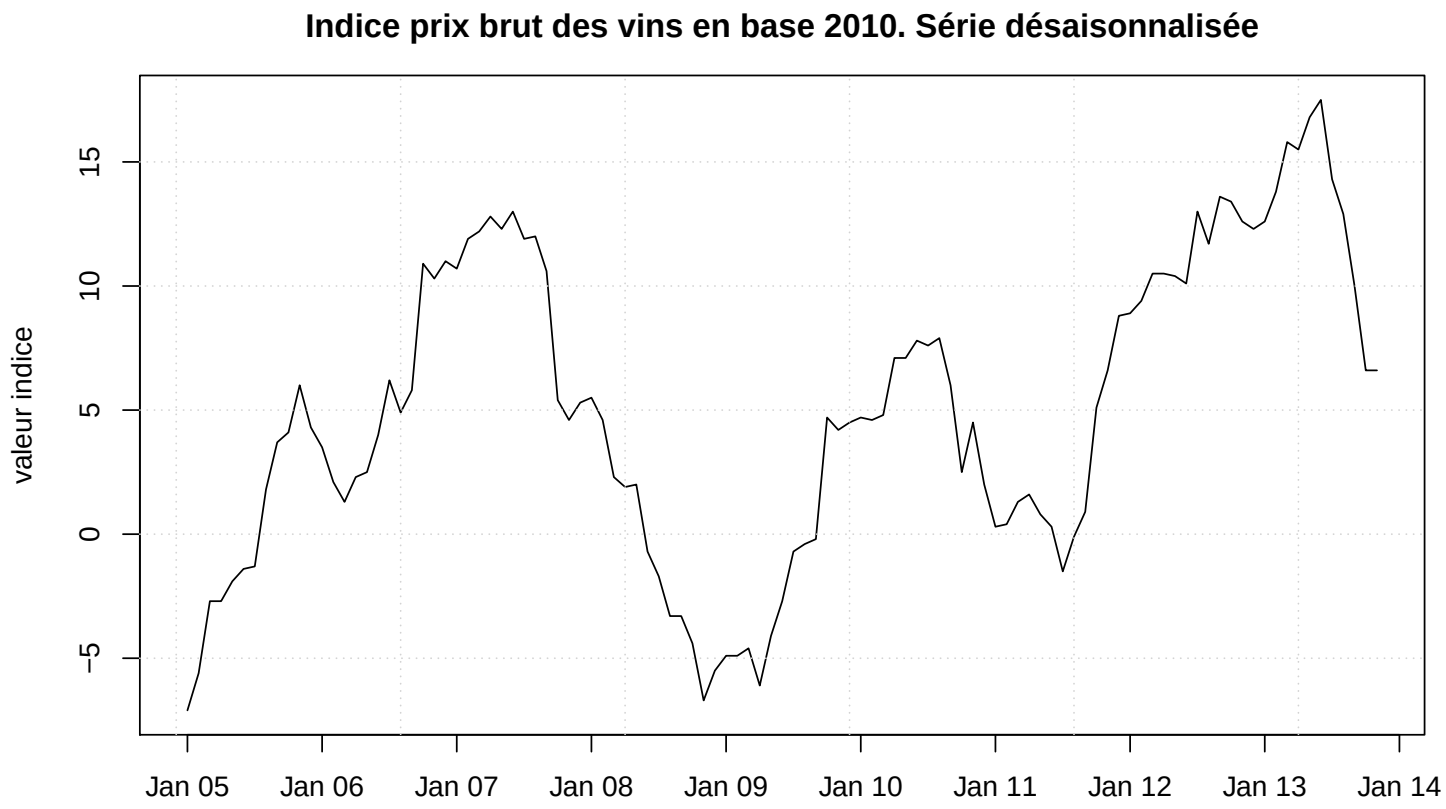

```r
par(mfrow = c(1, 1))
```

Modélisation

Suppression de la saisonnalité

On commence par désaisonnaliser la série en différentiant d'ordre 1 avec un lag de 12 mois.

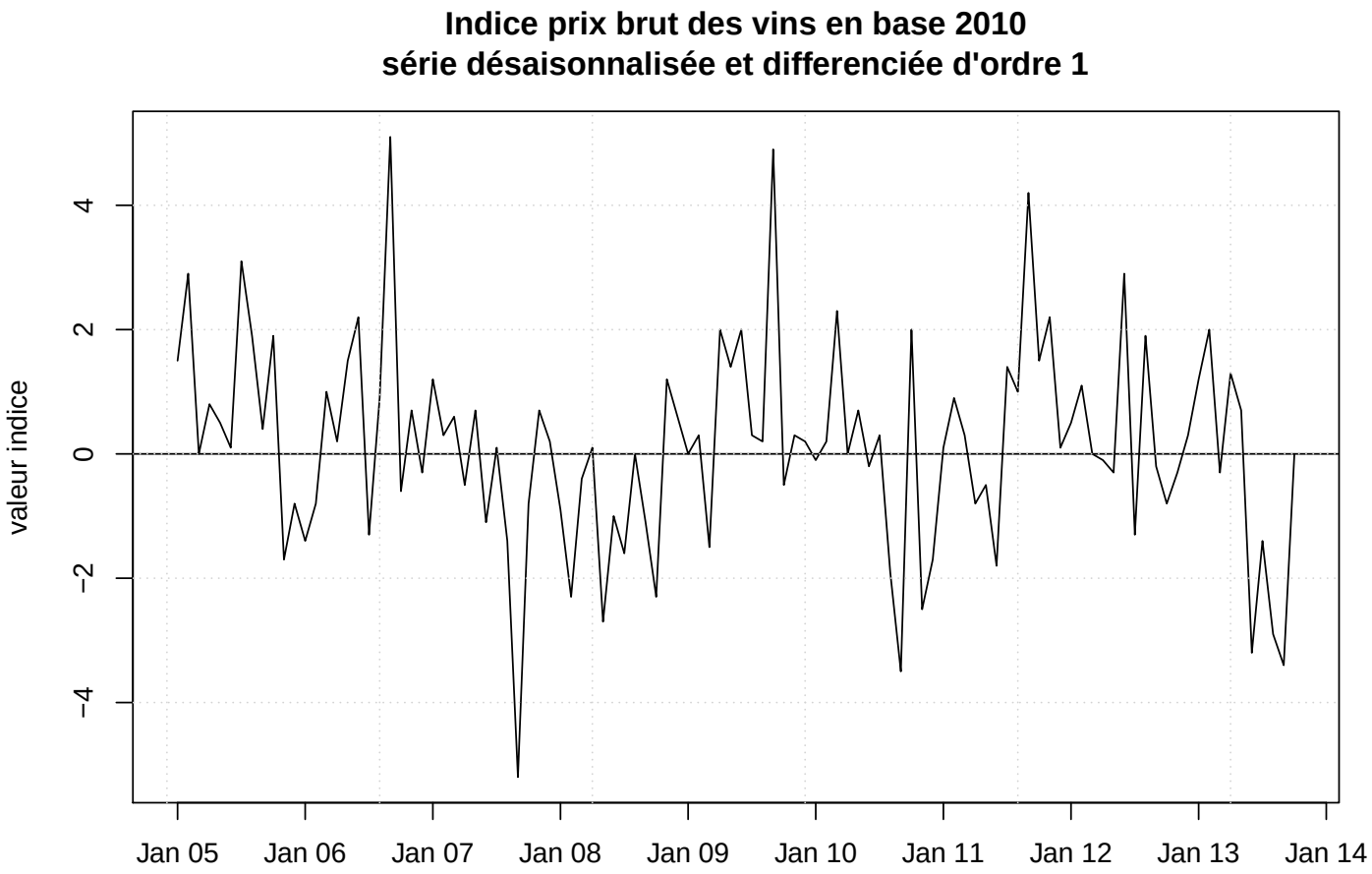
```
par(mar=c(2.5,4,3,0.5))
diff=diff(price,lag=12,differences=1)
plot(diff,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Indice prix brut des vins en base 2010. Série désaisonnalisée")
axis(1,at=which(wine[,"mois"]==1),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14"))
grid()
```



Atteinte de la stationnarité

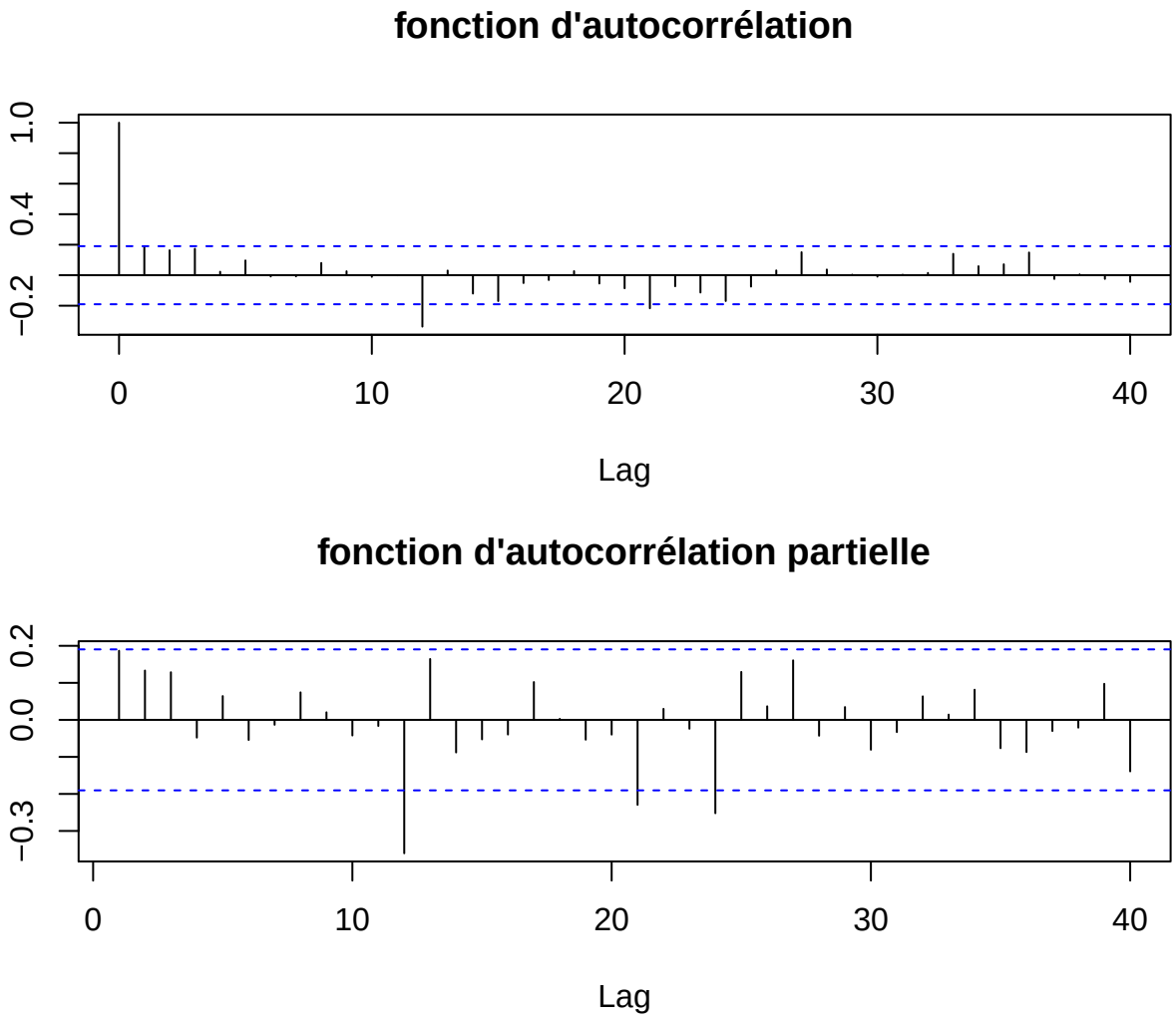
La série désaisonnalisée n'est clairement pas stationnaire. On la différencie donc d'ordre 1 (avec cette fois un lag de 1).

```
diff2=diff(diff,lag=1,differences=1)
plot(diff2,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Indice prix brut des vins en base 2010
série désaisonnalisée et differenciée d'ordre 1")
axis(1,at=which(wine[,"mois"]==1),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14"))
abline(h=0)
grid()
```



Cette fois-ci, une analyse plus précise est nécessaire pour déterminer si la série obtenue est stationnaire ou non. On examine donc les fonctions d'autocorrélation et d'autocorrélation partielle.

```
par(mar = c(5, 2.5, 3, 1.5))
par(mfrow = c(2, 1))
acf(diff2, main = "fonction d'autocorrélation", lag.max = 40, ylab = "")
pacf(diff2, main = "fonction d'autocorrélation partielle", lag.max = 40, ylab = "")
```

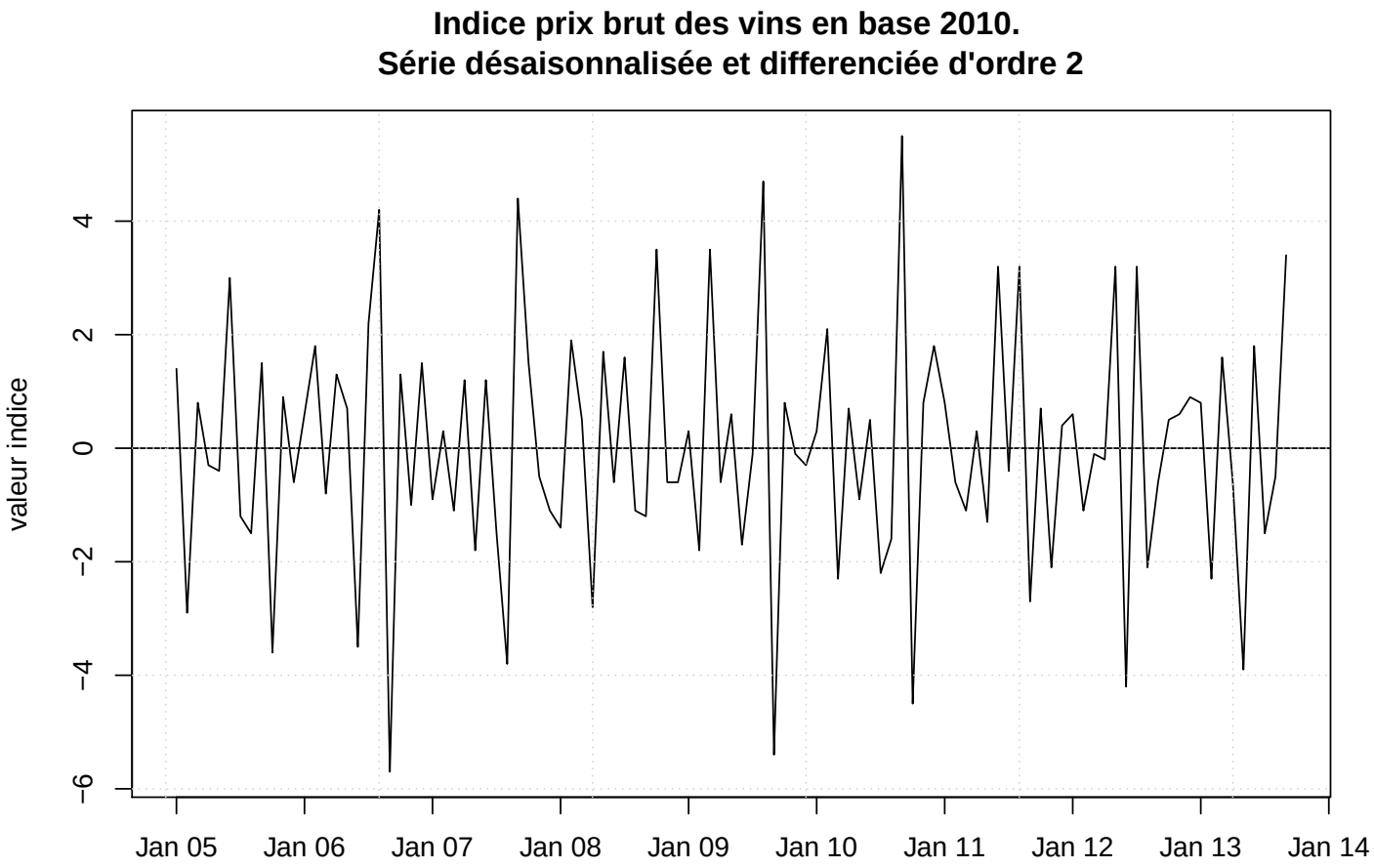


Les trois premiers lags de l'ACF sont à la limite d'être significatifs. On utilise l'Augmented Dickey-Fuller (ADF) test pour trancher. Ce test a pour hypothèse nulle que la série a une racine unité, et n'est donc pas stationnaire. Obtenir une p-value significative (<0.05) et rejeter l'hypothèse nulle revient donc à dire que la série est stationnaire.

```
cat("Augmented Dickey-Fuller test p-value:", adf.test(diff2)$p.value)

Augmented Dickey-Fuller test p-value: 0.1151
```

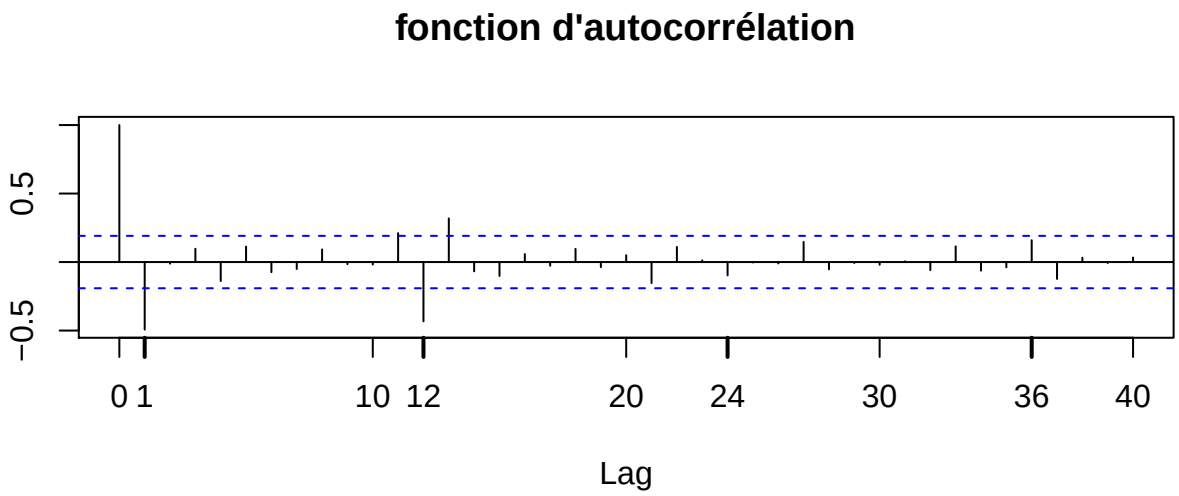
La p-value etant élevée, on ne peut pas rejeter l'hypothèse nulle (non-stationnarité). On conclut donc qu'une différence de premier ordre ne suffit pas à rendre la série stationnaire. Il faut donc différencier d'ordre 2.

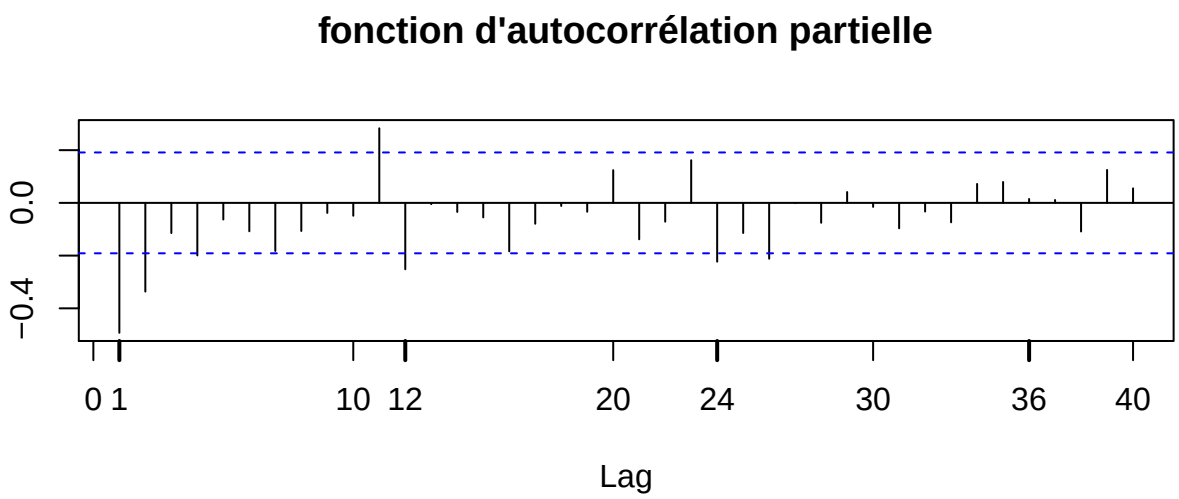
```
diff3=diff(diff,lag=1,differences=2)
par(mfrow=c(1,1))
plot(diff3,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Indice prix brut des vins en base 2010.
Série désaisonnalisée et differenciée d'ordre 2")
axis(1,at=which(wine[,"mois"]==1),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14"))
abline(h=0)
grid()
```



La série ressemble cette fois-ci à du bruit blanc. De plus, le ACF et PACF montrent que la série est stationnaire : en dehors des lags saisonniers indiqués en gras et du premier lag du ACF, aucun lag n'est significatif.

```
par(mfrow = c(2, 1))
par(mar = c(5, 2.5, 3, 1.5))
acf(diff3, lag.max = 40, main = "fonction d'autocorrélation")
axis(1, at = c(1, 12, 24, 36), lab = c("1", "12", "24", "36"), lwd = 0, lwd.ticks = 2)
pacf(diff3, lag.max = 40, main = "fonction d'autocorrélation partielle")
axis(1, at = c(1, 12, 24, 36), lab = c("1", "12", "24", "36"), lwd = 0, lwd.ticks = 2)
```






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

Le ADF test confirme la stationnarité :

```
cat("Augmented Dickey-Fuller test p-value:", adf.test(diff3)$p.value)
```

```
Augmented Dickey-Fuller test p-value: 0.01
```

Les graphiques suggèrent un modèle SARMA. Comme la série a été différenciée d'ordre 2, et désaisonnalisée d'ordre 1 (avec une période de 12 mois), le modèle sera en réalité un SARIMA(p,2,q)(s.p,1,s.q) de période 12. Il est difficile de déterminer précisément les meilleurs ordres (p,q) et (s.p,s.q) des parties saisonnière et non-saisonnière par une approche graphique. C'est pourquoi une approche quantitative basée sur le AIC est utilisée.

Choix du meilleur modèle

Le Akaike Information Criterion (AIC) est défini par : $AIC = 2k - 2\log(L)$. L est la function de vraisemblance et k le nombre de paramètres du modèle. Pour chaque combinaison d'ordres (p,q,s.p,s.q), la valeur du AIC pour le modèle SARIMA correspondant est calculée. Le meilleur modèle (la meilleure combinaison d'ordres) est celui qui est associé au plus petit AIC. En d'autres termes, le meilleur modèle maximise la probabilité conjointe d'observer les valeurs de la série temporelle (en maximizant le logarithme de la fonction de vraisemblance), tout en étant parcimonieux (petit nombre de paramètres). Par exemple, un SARIMA(0,2,1)(1,1,0) sera moins pénalisé qu'un modèle plus complexe, comme un SARIMA(3,2,2)(2,1,3). Dans notre recherche de combinaisons, on se restreint à des ordres compris entre 0 et 3.

```
# tableau pour recenser toutes les combinaisons d'ordres et les AICs
output = cbind(expand.grid(p = 0:3, q = 0:3, s.p = 0:3, s.q = 0:3), as.numeric(vector(length = 256)))
colnames(output)[5] = "AIC"
head(output)
```

```
  p q s.p s.q AIC
1 0 0   0   0   0
2 1 0   0   0   0
3 2 0   0   0   0
```

```
4 3 0   0   0   0
5 0 1   0   0   0
6 1 1   0   0   0
```

```
tail(output)
```

```
    p q s.p s.q AIC
251 2 2   3   3   0
252 3 2   3   3   0
253 0 3   3   3   0
254 1 3   3   3   0
255 2 3   3   3   0
256 3 3   3   3   0
```

```
# les modèles correspondants à toutes les combinaisons (256) de p, q, s.p,
# et s.q sont créés. tryCatch() interceptes les éventuelles erreurs
# d'optimisation de la fonction Arima() avant que la boucle ne s'arrête.
# Comme la boucle est lente elle est parallélisée.

cl = makeCluster(4)
registerDoParallel(cl)

output[, 5] = foreach(i = 1:length(output[, 1]), .packages = "forecast", .combine = "c",
    .export = c("price", "output")) %dopar% {
    tryCatch({
        Arima(price, order = c(output[i, 1], 2, output[i, 2]), seasonal = list(order = c(output[i,
            3], 1, output[i, 4]), period = 12), method = "ML")$aic
    }, error = function(e) {
        NA
    })
}

stopCluster(cl)

# suppression des valeurs manquantes
output = na.omit(output)
# triage du tableau par valeurs d'AIC croissantes
output = output[order(output[, 5]), ]
# affichage des 8 meilleurs candidats
best = output[1:8, ]
best
```

```
    p q s.p s.q   AIC
229 0 1   2   3 378.7
69  0 1   0   1 379.4
245 0 1   3   3 379.6
233 0 2   2   3 380.1
149 0 1   1   2 380.2
133 0 1   0   2 380.6
70  1 1   0   1 380.7
73  0 2   0   1 380.8
```

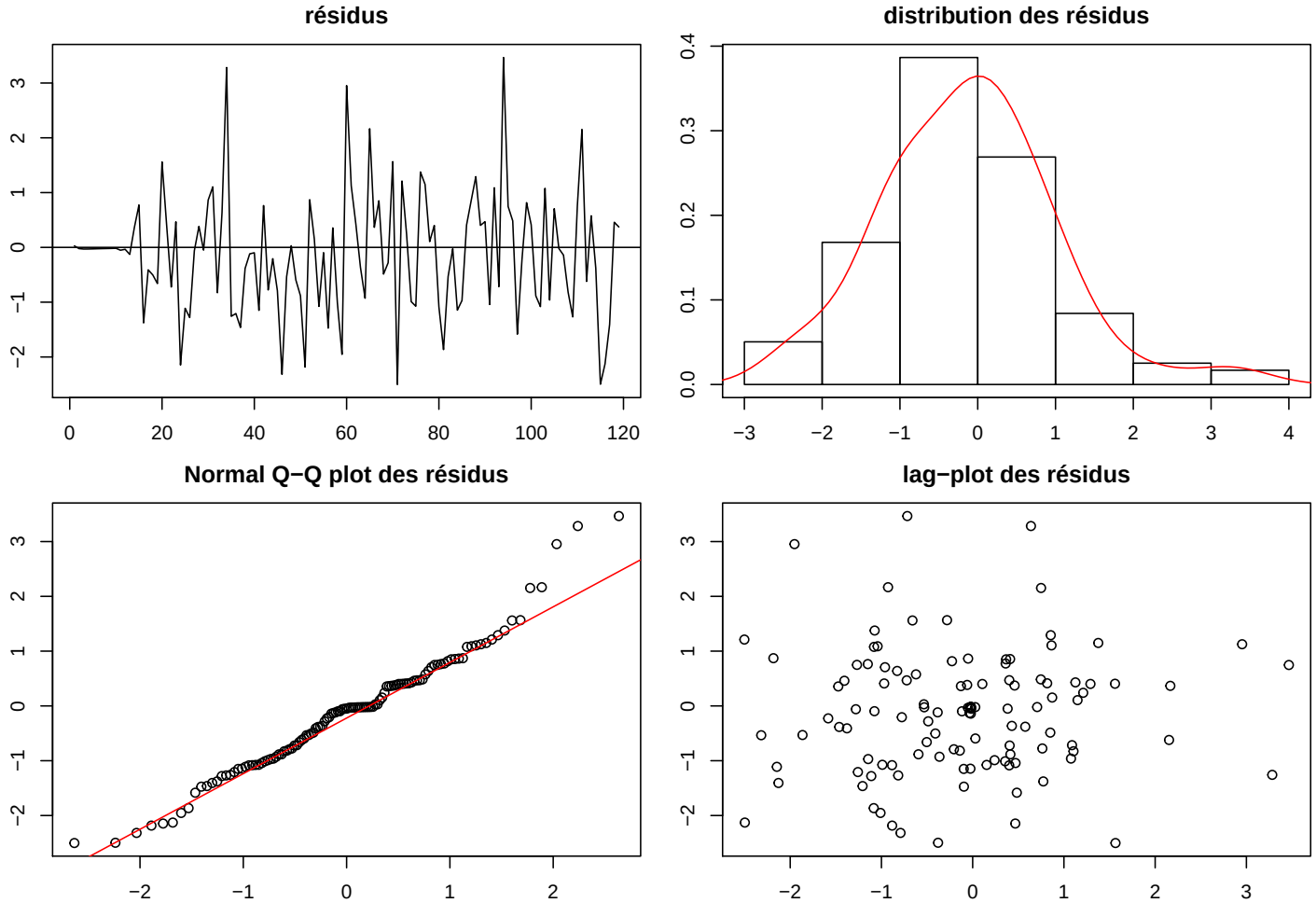
```
# affichage des plus mauvais (pour comparer)
tail(output)
```

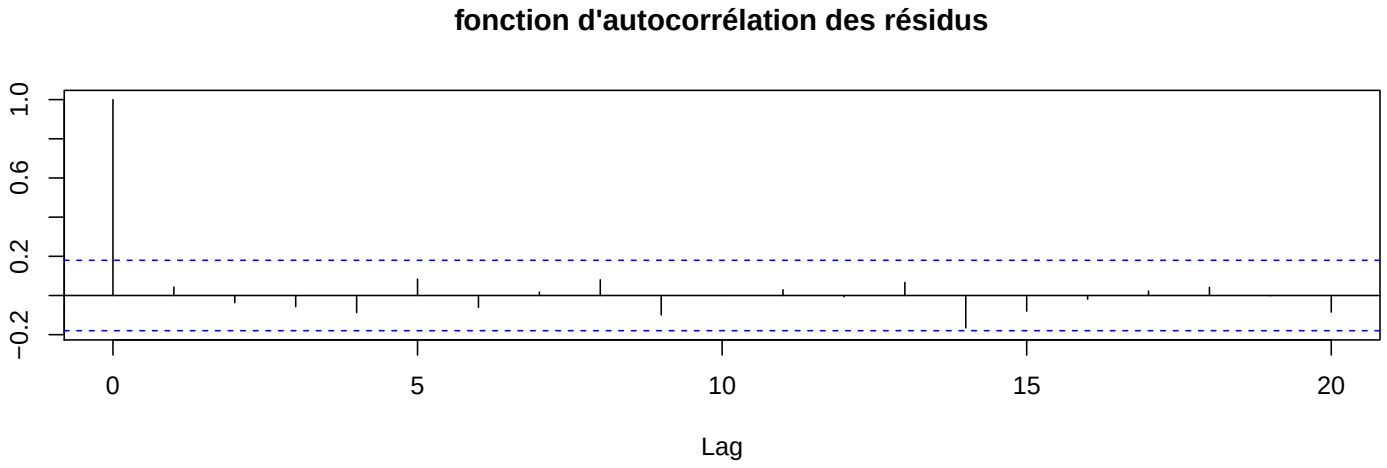
```
    p q s.p s.q   AIC
209 0 0   1   3 427.3
33  0 0   2   0 427.5
49  0 0   3   0 429.2
2   1 0   0   0 435.4
17  0 0   1   0 441.4
1   0 0   0   0 463.3
```

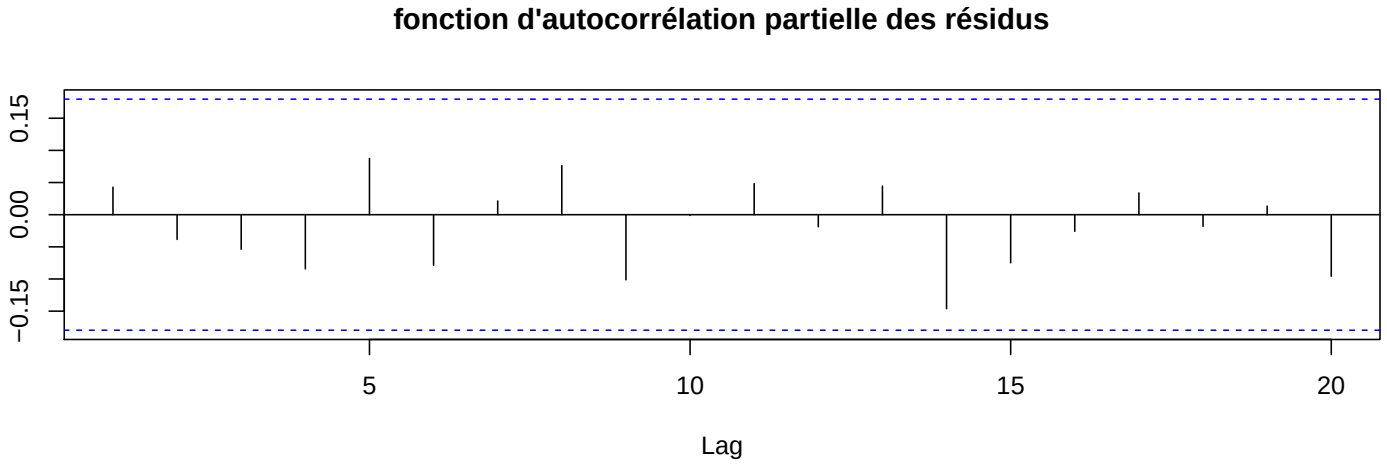
Diagnostique des résidus du meilleur modèle

Le meilleur modèle est pour p=0, q=1, s.p=2, et s.q=1 : SARIMA(0,2,1)(2,1,3) de période 12. On examine les résidus de ce modèle pour déterminer son adéquation.

```
setwd(wd)
source("check.residuals.ts.R")
# meilleur modèle
model = Arima(price, order = c(0, 2, 1), seasonal = list(order = c(2, 1, 3),
    period = 12), method = "ML")
summary(model)

Series: price
ARIMA(0,2,1)(2,1,3)[12]

Coefficients:
          ma1     sar1     sar2    sma1     sma2     sma3
      -0.816  -1.320  -0.930  0.684  -0.188  -0.728
s.e.   0.068   0.147   0.105  0.319   0.337   0.292

sigma^2 estimated as 1.38:  log likelihood=-182.4
AIC=378.7   AICc=379.9   BIC=397.3

Training set error measures:
                    ME  RMSE    MAE     MPE   MAPE    MASE
Training set -0.1231 1.105 0.8377 -0.1181 0.8104 0.08266

# résidus
res = residuals(model)
# examen des résidus
check.residuals.ts(res)
```







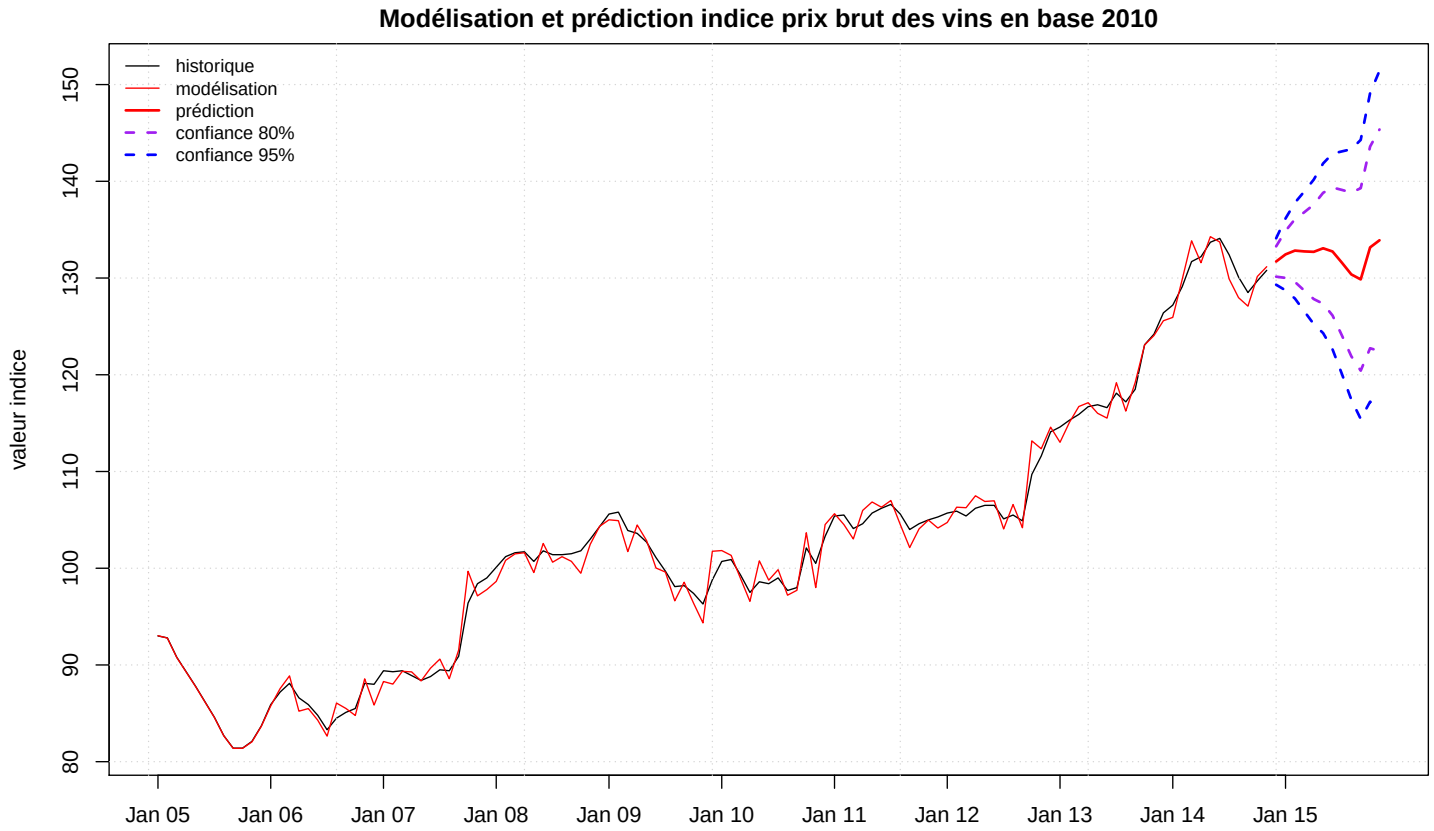

```
Box-Ljung test p-value: 0.6365. Kruskal-Wallis test p-value: 0.1044
```

En regardant le graphique des résidus, leur lag-plot, et l'ACF et la PACF, on voit qu'il n'y a aucune structure (aucune corrélation) dans les résidus. Ils ressemblent à du bruit blanc. Ceci est confirmé par le Box-Ljung test qui montre que les résidus sont indépendants (l'hypothèse nulle de ce test est l'indépendance, et elle n'est pas rejetée). Cela signifie que le modèle capture la grande majorité du signal présent dans la série observée. De plus, l'histogramme et le Normal Q-Q plot des résidus montrent que les résidus suivent une loi Normale. Le Kruskal-Wallis test ne rejette pas non plus l'hypothèse nulle que les résidus sont Normalement distribués. Le modèle est donc adéquat.

Prédiction

```
predict=forecast.Arima(model,h=12) # 12 prochains mois
par(mfrow=c(1,1))
par(mar=c(2.5,4,2,0.5))
plot(price,type="l",xlab="",xaxt="n",ylab="valeur indice",
main="Modélisation et prédiction indice prix brut des vins en base 2010",xlim=range(1:(length(price)+12)),
ylim=range(price,predict$upper[,2]))
axis(1,at=c(which(wine[,"mois"]==1),121),
labels=c("Jan 05","Jan 06","Jan 07","Jan 08","Jan 09","Jan 10","Jan 11","Jan 12","Jan 13","Jan 14","Jan 15"))
grid()
# valeurs modélisées
lines(price+res,lwd=1,col="red")
# valeurs prédites
lines(predict$mean,lwd=2,col="red")
# interval de confiance 80%
lines(x=120:131,predict$lower[,1],lwd=2,lty=2,col="purple")
lines(x=120:131,predict$upper[,1],lwd=2,lty=2,col="purple")
# interval de confiance 95%
lines(x=120:131,predict$lower[,2],lwd=2,lty=2,col="blue")
lines(x=120:131,predict$upper[,2],lwd=2,lty=2,col="blue")
legend("topleft",legend=c("historique","modélisation","prédiction","confiance 80%","confiance 95%"),
```

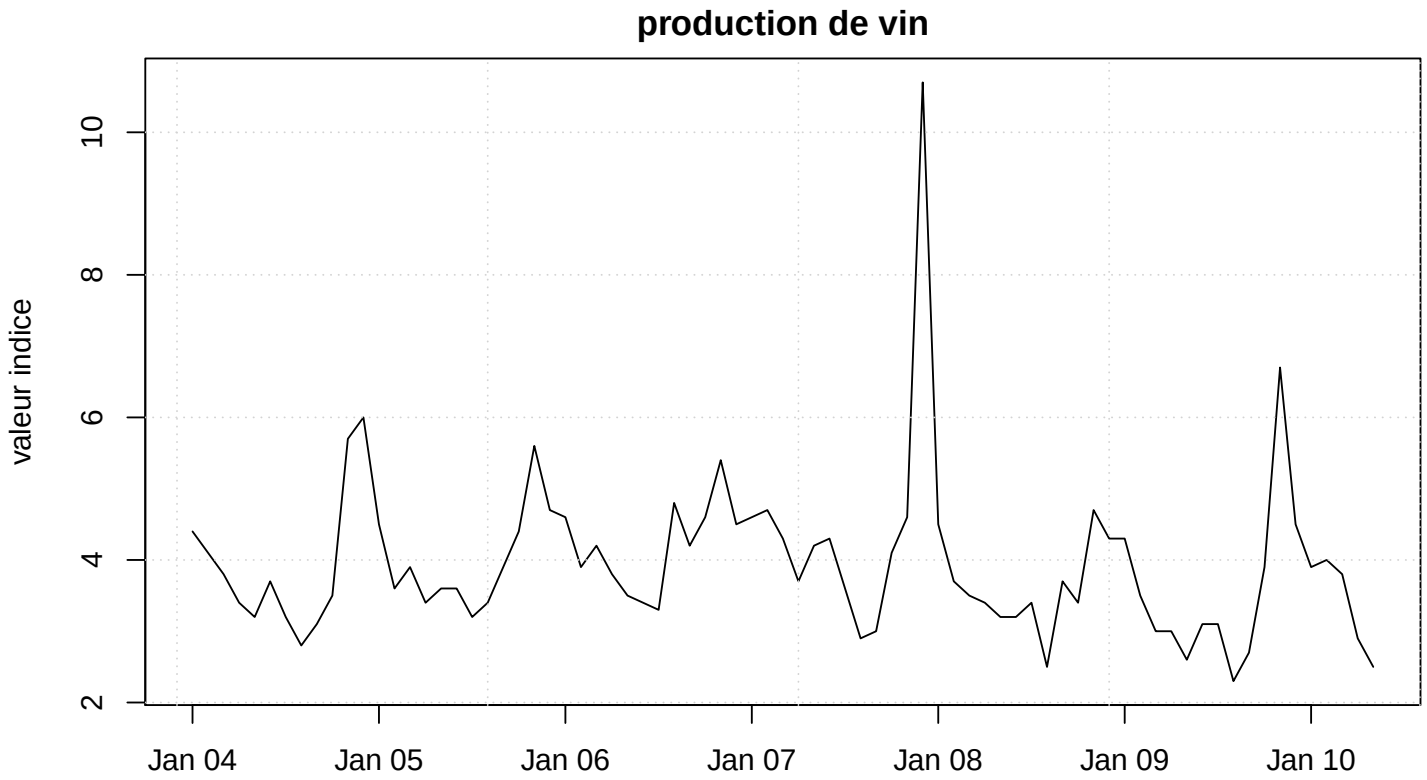
```
lty=c(1,1,1,2,2),lwd=c(1,1,2,2,2),col=c("black","red","red","purple","blue"),bty="n",cex=0.85)
```



Modèle avec variable extérieure

Dans cette section, on essaye d'améliorer la modélisation en prenant en compte l'impact de la production de vin sur les prix. Cet impact n'est pas immédiat mais se répercute au cours des années suivantes, plus ou moins rapidement en fonction des stocks disponibles, de la demande etc... Dans ce qui suit, on fait l'hypothèse que la production de vin affecte les prix avec une année de retard. On peut donc utiliser les chiffres de la production de 2004 pour modéliser l'évolution des prix en 2005, et ainsi de suite. Les données sont disponibles sur cette page. Notre étude doit se restreindre à la période 2005-2011 car les données de production ne sont plus disponibles après Mai 2010.

```
setwd(wd)
production = read.table("production.txt", header = TRUE)
head(production)

  année mois valeur
1  2004    1    4.4
2  2004    2    4.1
3  2004    3    3.8
4  2004    4    3.4
5  2004    5    3.2
6  2004    6    3.7

par(mfrow = c(1, 1))
par(mar = c(2.5, 4, 2, 0.5))
plot(production[, 3], type = "l", xlab = "", xaxt = "n", ylab = "valeur indice",
    main = "production de vin")
axis(1, at = c(which(production[, "mois"] == 1)), labels = c("Jan 04", "Jan 05",
    "Jan 06", "Jan 07", "Jan 08", "Jan 09", "Jan 10"))
grid()
```



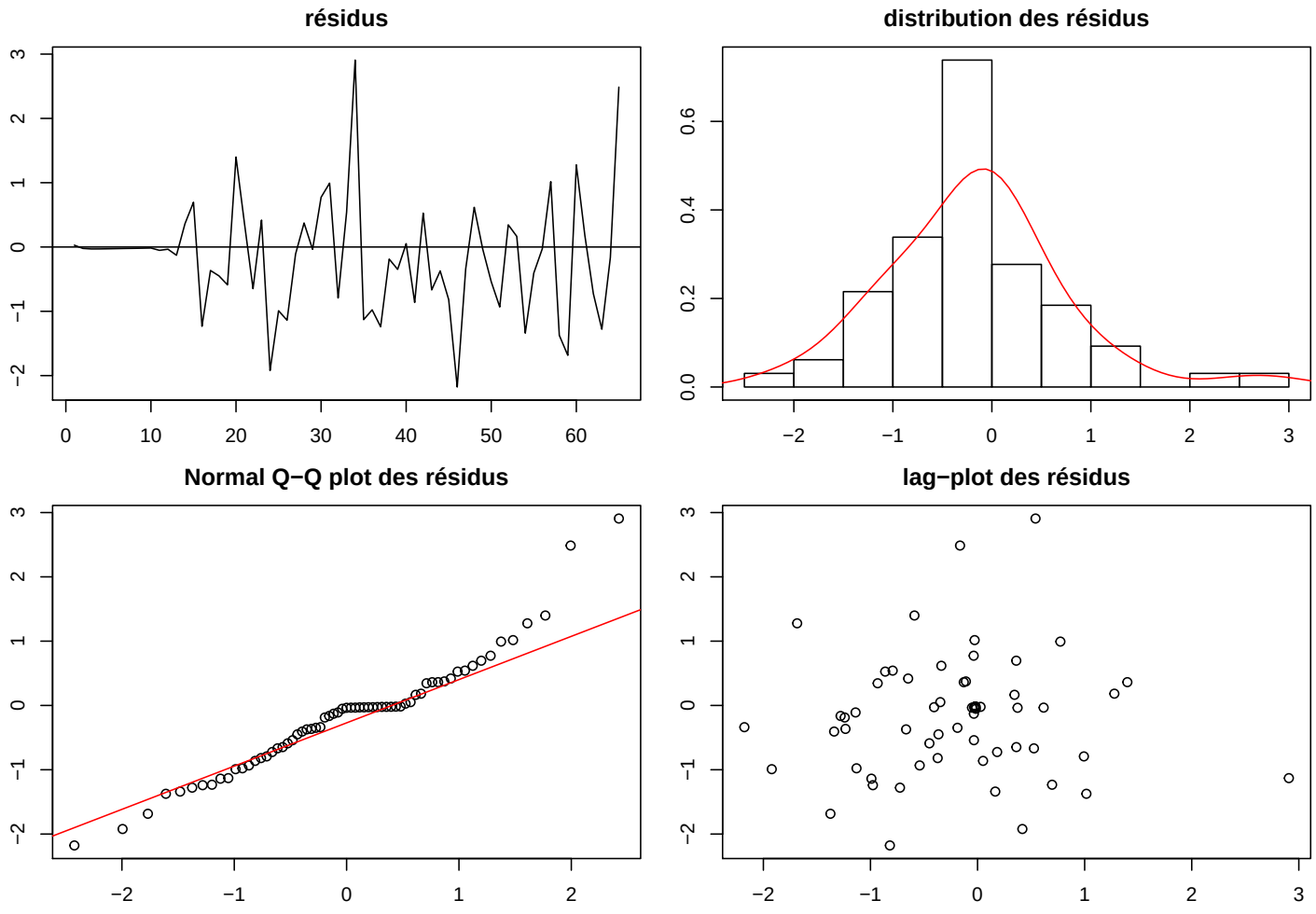
Tout d'abord, comme nous ne modélisons ici qu'une version tronquée de la série temporelle de départ, il est nécessaire de retrouver le meilleur modèle SARIMA sans variable extérieure. En effet les ordres et l'estimation des paramètres du modèle dépendent des valeurs observées.

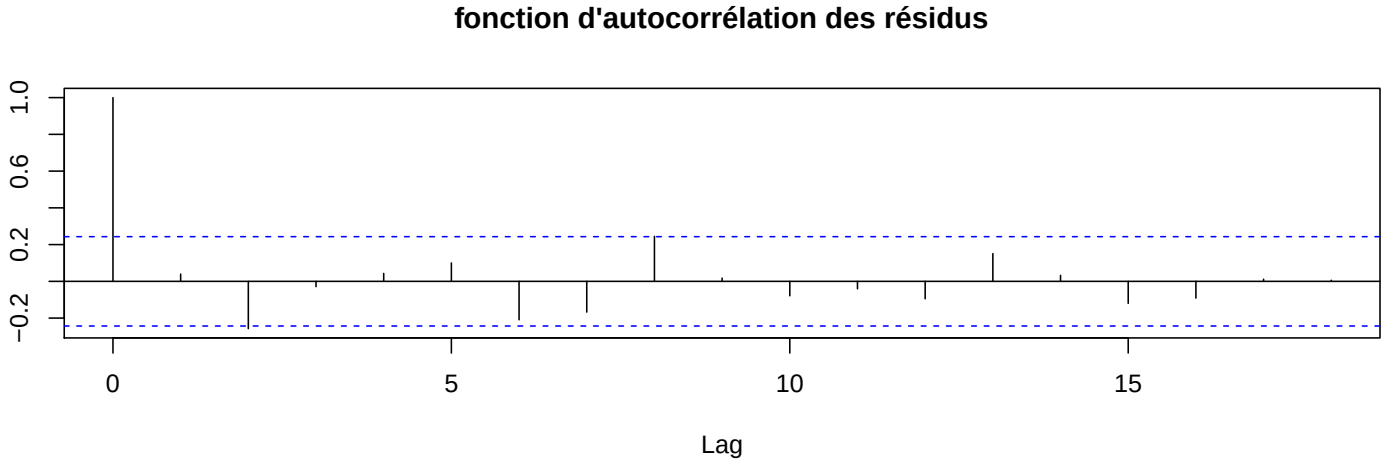
```
output2 = cbind(expand.grid(p = 0:3, q = 0:3, s.p = 0:3, s.q = 0:3), as.numeric(vector(length = 256)))
colnames(output2)[5] = "AIC"

cl = makeCluster(4)
registerDoParallel(cl)

output2[, 5] = foreach(i = 1:length(output2[, 1]), .packages = "forecast", .combine = "c",
    .export = c("price", "output2")) %dopar% {
    tryCatch({
        Arima(price[1:65], order = c(output2[i, 1], 2, output2[i, 2]), seasonal = list(order = c(output2[i,
            3], 1, output2[i, 4]), period = 12), method = "ML")$aic
    }, error = function(e) {
        NA
    })
}

stopCluster(cl)

output2 = na.omit(output2)
output2 = output2[order(output2[, 5]), ]
head(output2)

   p q s.p s.q   AIC
53 0 1   3   0 191.9
57 0 2   3   0 192.4
55 2 1   3   0 192.5
61 0 3   3   0 192.6
69 0 1   0   1 193.1
58 1 2   3   0 193.3

tail(output2)

    p q s.p s.q   AIC
161 0 0   2   2 214.0
225 0 0   2   3 214.0
17  0 0   1   0 215.7
33  0 0   2   0 215.9
```

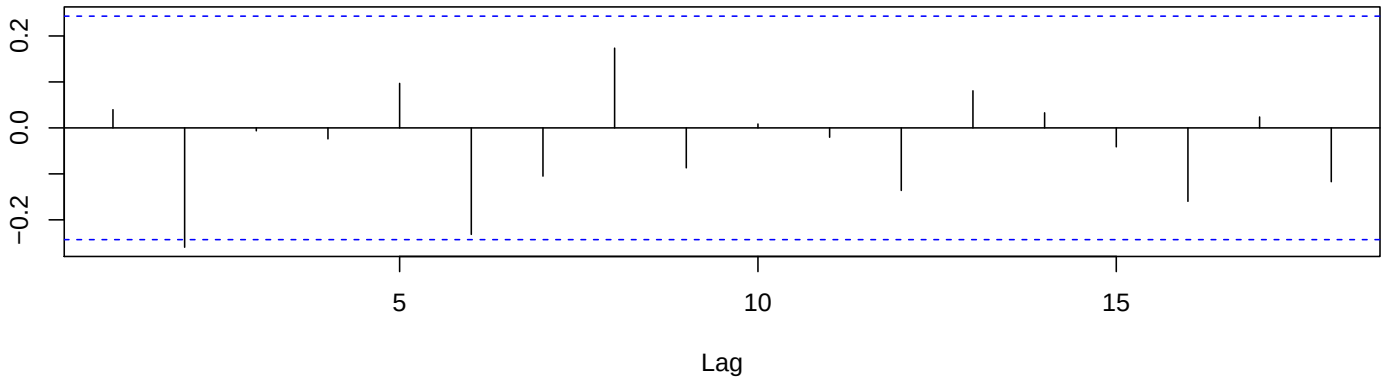
```
2    1 0    0    0 217.7
1    0 0    0    0 226.4
```

Le meilleur modèle est un SARIMA(0,2,1)(3,1,0) de période 12. On examine ses résidus.

```
setwd(wd)
source("check.residuals.ts.R")

model2 = Arima(price[1:65], order = c(0, 2, 1), seasonal = list(order = c(3,
    1, 0), period = 12), method = "ML")
summary(model2)

Series: price[1:65]
ARIMA(0,2,1)(3,1,0)[12]

Coefficients:
         ma1    sar1    sar2    sar3
      -0.814  -0.643  -0.574  -0.712
s.e.   0.092   0.190   0.203   0.128

sigma^2 estimated as 1.04:  log likelihood=-90.94
AIC=191.9   AICc=193.2   BIC=201.5

Training set error measures:
                    ME  RMSE    MAE     MPE   MAPE    MASE
Training set -0.1663 0.904 0.6438 -0.1703 0.6775 0.09437

# résidus
res2 = residuals(model2)
# examen des résidus
check.residuals.ts(res2)
```








```
Box-Ljung test p-value: 0.7438. Kruskal-Wallis test p-value: 0.008146
```

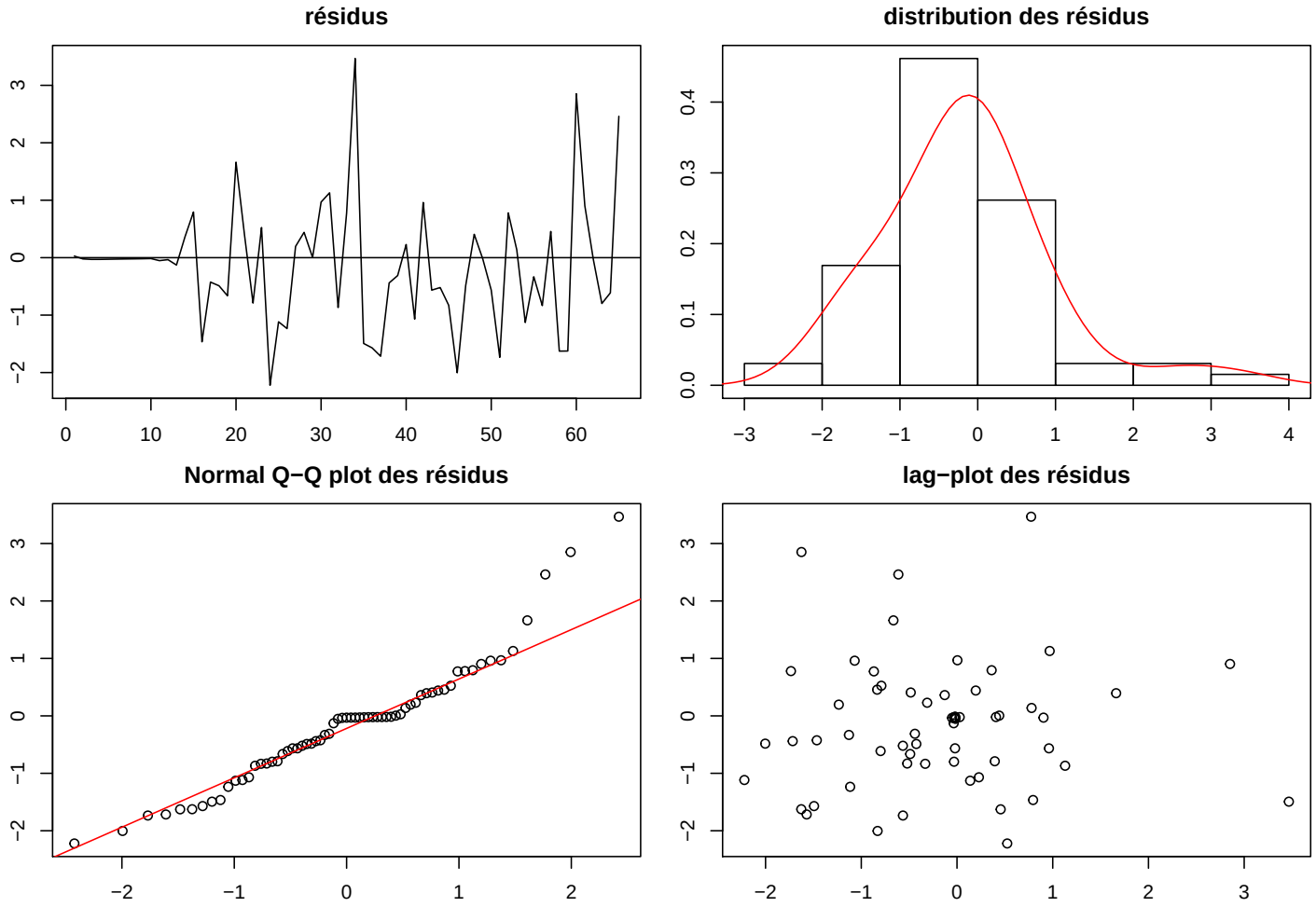
Le modèle est adéquat. A présent on cherche le meilleur modèle SARIMA prenant en compte la production comme variable extérieure.

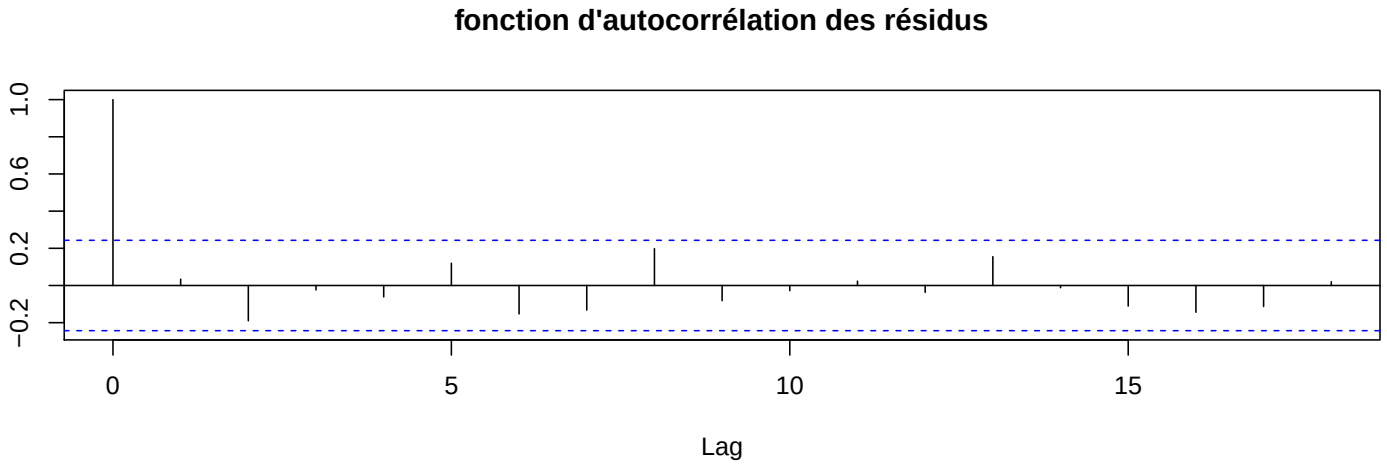
```
output.x = cbind(expand.grid(p = 0:3, q = 0:3, s.p = 0:3, s.q = 0:3), as.numeric(vector(length = 256)))
colnames(output.x)[5] = "AIC"

cl = makeCluster(4)
registerDoParallel(cl)

output.x[, 5] = foreach(i = 1:length(output.x[, 1]), .packages = "forecast",
    .combine = "c", .export = c("price", "output.x", "production")) %dopar%
    {
        write.table(i, file = "progress.txt")
        tryCatch({
            Arima(price[1:65], order = c(output.x[i, 1], 2, output.x[i, 2]),
                seasonal = list(order = c(output.x[i, 3], 1, output.x[i, 4]),
                  period = 12), xreg = production[1:65, 3], method = "ML")$aic
        }, error = function(e) {
            NA
        })
    }

stopCluster(cl)

output.x = na.omit(output.x)
output.x = output.x[order(output.x[, 5]), ]
head(output.x)

    p q s.p s.q   AIC
69  0 1   0   1 143.6
53  0 1   3   0 143.9
57  0 2   3   0 144.8
73  0 2   0   1 145.1
70  1 1   0   1 145.3
133 0 1   0   2 145.3

tail(output.x)

    p q s.p s.q   AIC
161 0 0   2   2 159.2
241 0 0   3   3 159.7
33  0 0   2   0 160.0
225 0 0   2   3 160.4
2   1 0   0   0 161.0
1   0 0   0   0 166.7
```

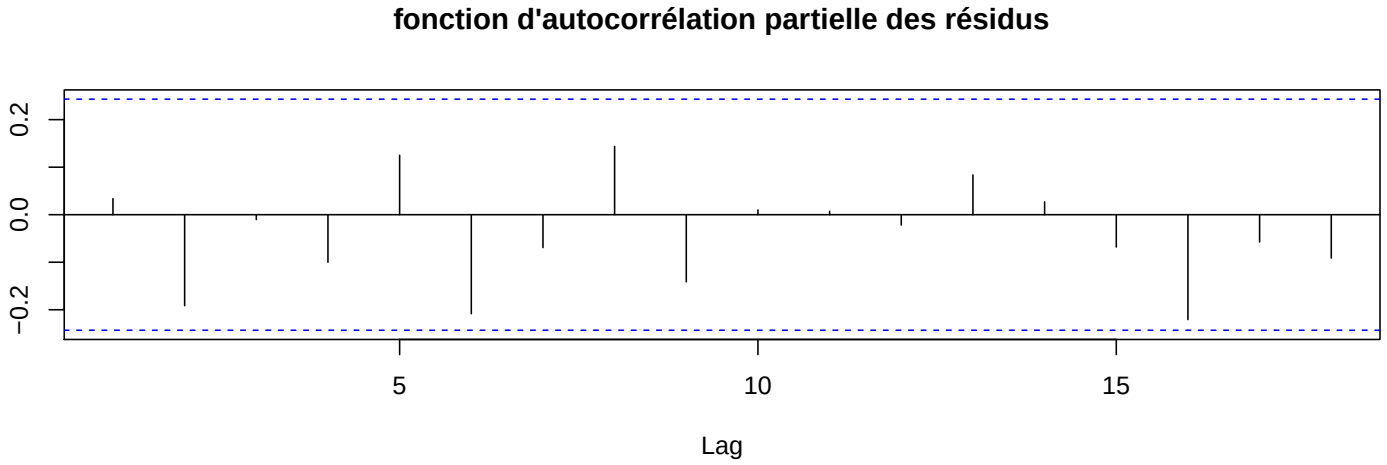
Le meilleur modèle est un SARIMA $(0,2,1)(0,1,1)$ de période 12. On remarque que grâce à l'apport de l'information apportée par la variable extérieure, les ordres du modèle sont plus bas. On examine les résidus.

```
setwd(wd)
source("check.residuals.ts.R")

# meilleur modèle
model.x = Arima(price[1:65], order = c(0, 2, 1), seasonal = list(order = c(0,
    1, 1), period = 12), xreg = production[1:65, 3], method = "ML")
summary(model.x)

Series: price[1:65]
ARIMA(0,2,1)(0,1,1)[12]

Coefficients:
          ma1     sma1  production[1:65, 3]
      -0.765  -0.934                0.040
s.e.   0.118   0.413                0.141

sigma^2 estimated as 2.01:  log likelihood=-67.81
```

```
AIC=143.6   AICc=144.5   BIC=151.3

Training set error measures:
                    ME RMSE    MAE      MPE   MAPE   MASE
Training set -0.1551 1.07 0.7684 -0.1587 0.8074 0.1126

# résidus
res.x = residuals(model.x)
# examen des résidus
check.residuals.ts(res.x)
```





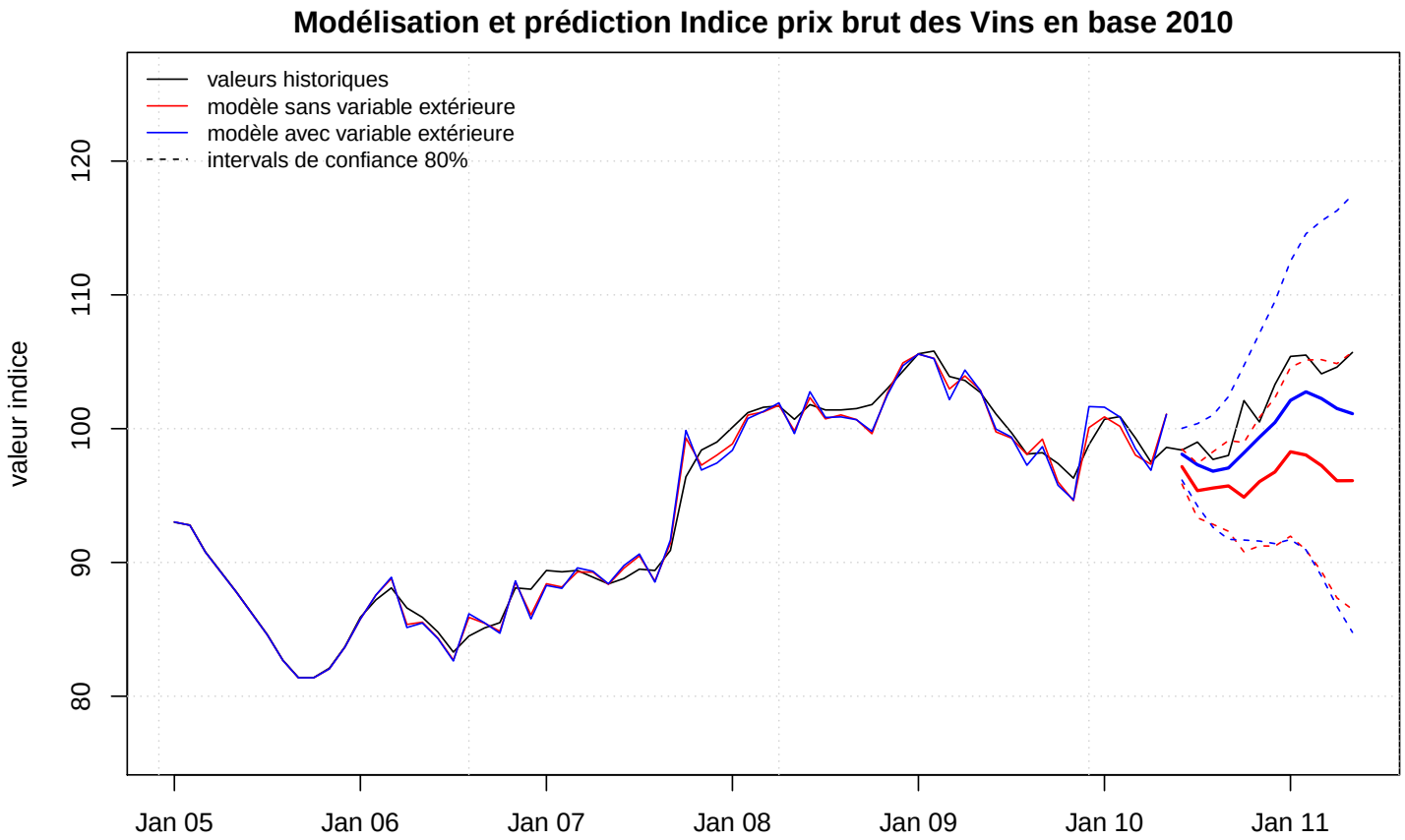



```
Box-Ljung test p-value: 0.78. Kruskal-Wallis test p-value: 0.02445
```

Le modèle est adéquat. On compare à présent les deux modèles en termes de prédiction. En effet, un modèle performant doit non seulement reproduire fidèlement les observations de la série temporelle qui a été utilisée pour estimer ses paramètres, mais doit aussi être capable de fournir des prédictions réalistes de valeurs qu'il n'a jamais vues. On compare les prédictions des deux modèles pour la période Juin 2010-Mai 2011 (prédiction des 12 prochain mois). On passe les valeurs de production de vin de Juin 2009 à Mai 2010 à la fonction forecast.Arima() pour les prédictions du modèle avec variable extérieure.

```
predict2 = forecast.Arima(model2, h = 12)
predict.x = forecast.Arima(model.x, h = 12, xreg = production[66:77, 3])
par(mfrow = c(1, 1))
par(mar = c(2.5, 4, 2, 0.5))

plot(price[1:77], type = "l", xlab = "", xaxt = "n", ylab = "valeur indice",
    main = "Modélisation et prédiction Indice prix brut des Vins en base 2010",
    xlim = range(1:(length(price[1:77]))), ylim = range(predict2$upper[, 2],
        predict.x$upper[, 2], predict2$lower[, 2], predict.x$lower[, 2]))
axis(1, at = which(wine[1:77, "mois"] == 1), labels = c("Jan 05", "Jan 06",
    "Jan 07", "Jan 08", "Jan 09", "Jan 10", "Jan 11"))
grid()
# valeurs modélisées
lines(price[1:65] + res2, lwd = 1, col = "red")
lines(price[1:65] + res.x, lwd = 1, col = "blue")
# valeurs prédites
lines(predict2$mean, lwd = 2, col = "red")
lines(predict.x$mean, lwd = 2, col = "blue")
# intervals de confiance 80%
lines(x = 66:77, predict2$lower[, 1], lwd = 1, lty = 2, col = "red")
lines(x = 66:77, predict2$upper[, 1], lwd = 1, lty = 2, col = "red")
lines(x = 66:77, predict.x$lower[, 1], lwd = 1, lty = 2, col = "blue")
lines(x = 66:77, predict.x$upper[, 1], lwd = 1, lty = 2, col = "blue")
legend("topleft", legend = c("valeurs historiques", "modèle sans variable extérieure",
    "modèle avec variable extérieure", "intervals de confiance 80%"), lty = c(1,
    1, 1, 2), lwd = c(1, 1, 1, 1), col = c("black", "red", "blue", "black"),
```

```
    bty = "n", cex = 0.85)
```



On voit qu'à l'inverse du modèle sans variable extérieure, le modèle prenant en compte la variable extérieure (production de vin) a été capable de reproduire plus fidèlement l'augmentation de l'indice. L'erreur (definie par l'écart-type des résidus, ou RMSE) du modèle avec variable extérieure est moindre :

```
# erreur du modèle avec var. ext.
sqrt((1/12) * sum((predict.x$mean - price[66:77])^2))

[1] 2.606

# erreur du modèle sans var. ext.
sqrt((1/12) * sum((predict2$mean - price[66:77])^2))

[1] 6.171
```

Contact

antoine.tixier-1@colorado.edu