Modéliser la volatilité d'une série financière avec un modèle GARCH univariés

avec R – une application au Bitcoin

Introduction

La volatilité est au cœur de l'analyse financière, que ce soit à l'échelle microéconomique ou macroéconomique. Pourtant, le concept de volatilité contient plusieurs mesures : l'écart-type, la variance, l'écart intra-journalier entre le prix maximum et le prix minimum, la somme quadratique des rendements intra-journaliers…

Parfois, il peut arriver que ces mesures aient des inconvénients relativement importants, et la modélisation statistique peut permettre de trouver une issue. Notamment, les modèles HMA, EWMA et ARCH/GARCH peuvent fournir des mesures de la volatilité, en modélisant les séries. Nous allons ici nous concentrer sur les modèles GARCH univariés, qui peuvent être vus comme une extension des modèles ARMA. Pour effectuer cette modélisation de la variance, le package **rugarch** permet (relativement) simplement de générer des modèles GARCH.

Nous proposons ici de fournir un exemple détaillé d'application de ce package (et de plein d'autres), avec l'application de la modélisation de la variance au Bitcoin. Plein d'études ont déjà étudié la variance du Bitcoin et des cryptomonnaies de manière générale, y compris avec des modèles GARCH. Ici, il ne s'agit pas de prolonger ces études déjà existantes, mais de fournir un code avec des données reproductibles et en libre-accès pour s'entraîner sur des données.

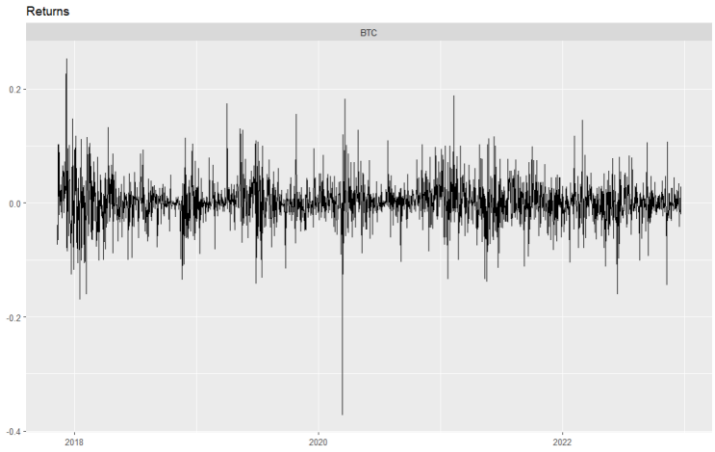
Données

Pour cela, nous utiliserons les données de la cryptomonnaie majeure, le Bitcoin. Les données sont récoltées sur le site Yahoo Finance entre le 9 novembre 2017 et le 20 décembre 2022, pour avoir suffisamment de données sans consommer trop de capacités computationnelles (les calculs peuvent être assez longs). Les données du Bitcoin sont stockées avec celles de cinq autres cryptomonnaies dans la database **cryptos**, qu'il convient d'importer et de transformer pour ne garder que la série qui nous intéresse. Nous précisons également dès le début l'ensemble des packages qui seront nécessaires pour l'analyse.

```
require(stargazer)
require(rugarch)
require(tidyr)
require(dplyr)
require(ggplot2)
require(PerformanceAnalytics)
require(forecast)
require(urca)
require(vars)
require(aTSA)
cryptos<-read.csv(file.choose())
crypto<-subset(cryptos, select=-c(X,XRP, ETH, LTC, BNB, ADA))
```

Une fois les données importées et les packages chargés, nous pouvons tracer la série de rendements du Bitcoin à l'aide de la commande suivante :

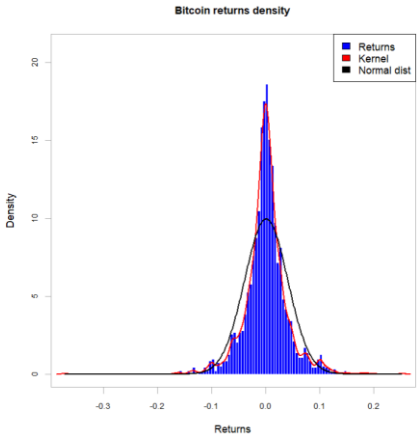
```
crypto2 = {
  crypto %>% dplyr::mutate(Date_1 =
as.Date(as.character(crypto[,1]))) %>%
    tibble %>% dplyr::select(-1) %>%
    pivot_longer(cols = -Date_1,names_to = "variables") %>%
    ggplot(aes(x = Date_1,y = value)) +
    geom_line() +
    facet_wrap(~variables,scales = "free_y") +
    labs(title = "Returns", x= "", y = "")}
crypto2
```



On peut déjà faire une première observation : la volatilité du Bitcoin est relativement importante. Les rendements sont marqués par des pics assez fréquents et regroupés en clusters de volatilité. Les statistiques descriptives peuvent nous fournir davantage d'informations, notamment concernant la distribution de la série.

```
stargazer(crypto, type="text")
chart.Histogram(crypto[,2],
                method=c("add.density", "add.normal"),
                colorset=c('blue', 'red', 'black'),
                main="Bitcoin returns density")
legend("topright",
       legend=c("Returns", "Kernel", "Normal dist"),
       fill=c('blue', 'red', 'black')
```

```
==========================================
Statistic   N   Mean  St. Dev.  Min    Max
------------------------------------------
BTC       1,867 0.001  0.040   -0.372 0.252
------------------------------------------
```



On observe que les rendements journaliers du Bitcoin sont leptokurtiques, c'est-à-dire qu'ils ont un pic plus haut que celui d'une loi normale, et des queues plus épaisses.

Processus ARIMA

Afin d'étudier la série de rendements du Bitcoin, il s'agit dans un premier temps d'étudier le processus AR(I)MA, à savoir le processus auto-régressif (AR) et de moyennes mobiles (MA), et potentiellement le nombre de différences nécessaires pour rendre le processus stationnaire (I).

Avant d'étudier le processus ARMA, nous pouvons estimer la stationnarité de la série à l'aide du test de Dickey-Fuller grâce au code suivant :

```
urdf<-ur.df(crypto$BTC, type="none")
summary(urdf)
```

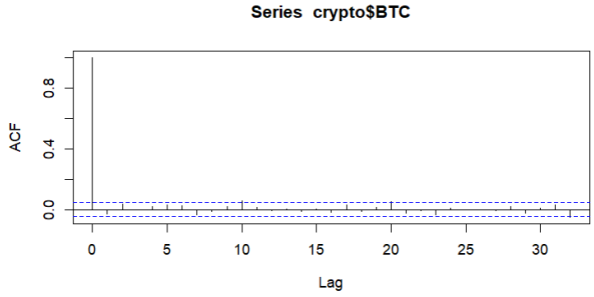
```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression none


Call:
lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max
-0.37163 -0.01589  0.00118  0.01813  0.25697

Coefficients:
          Estimate Std. Error t value Pr(>|t|)
z.lag.1    -0.99156    0.03315 -29.910   <2e-16 ***
z.diff.lag -0.03490    0.02313  -1.509    0.132
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.03998 on 1863 degrees of freedom
Multiple R-squared:  0.5143,    Adjusted R-squared:  0.5138
F-statistic: 986.5 on 2 and 1863 DF,  p-value: < 2.2e-16


Value of test-statistic is: -29.9097

Critical values for test statistics:
      1pct  5pct 10pct
tau1 -2.58 -1.95 -1.62
```

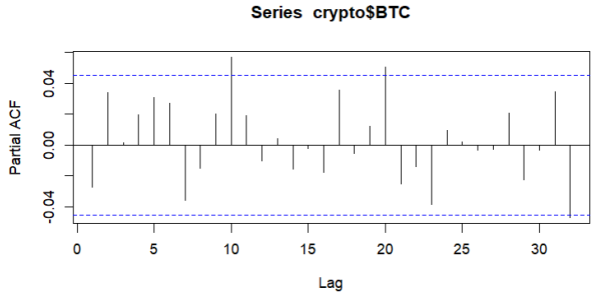
L'information importante à retenir est la suivante : la série est stationnaire (-29.9097<-2.58), il n'y a donc pas besoin de la transformer en différence.

Nous traçons ensuite le corrélogramme de la série, ainsi que le corrélogramme partiel, afin d'observer la mémoire de la série.

```
par(mfrow=c(2,1))
```

```
acf(crypto$BTC)
pacf(crypto$BTC)
```





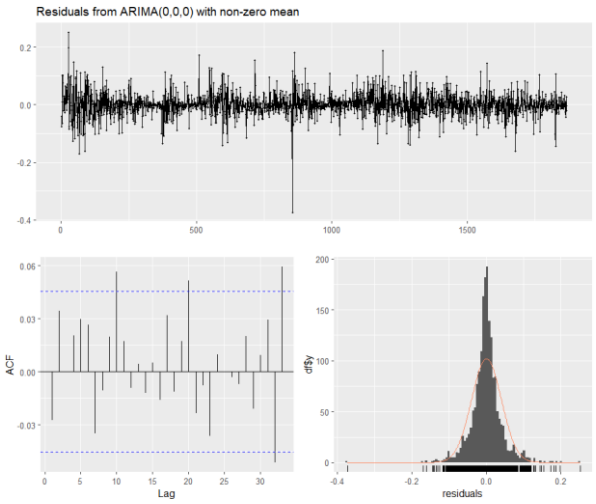
Il semble que le processus ait une mémoire, car certains lags dépassent la valeur critique (on peut le voir surtout sur le PACF). Afin de modéliser correctement la série, nous pouvons déterminer l'ordre AR(p), MA(q) ou ARMA(p,q) optimal.

```
model<-arima(crypto[,2], order=c(0,0,0))
ar1<-arima(crypto[,2], order=c(1,0,0))
ar2<-arima(crypto[,2], order=c(2,0,0))
ar3<-arima(crypto[,2], order=c(3,0,0))
ar4<-arima(crypto[,2], order=c(4,0,0))
ar5<-arima(crypto[,2], order=c(5,0,0))

ma1<-arima(crypto[,2], order=c(0,0,1))
ma2<-arima(crypto[,2], order=c(0,0,2))
ma3<-arima(crypto[,2], order=c(0,0,3))
ma4<-arima(crypto[,2], order=c(0,0,4))
ma5<-arima(crypto[,2], order=c(0,0,5))

arma11<-arima(crypto[,2], order=c(1,0,1))
arma12<-arima(crypto[,2], order=c(1,0,2))
arma21<-arima(crypto[,2], order=c(2,0,1))
arma22<-arima(crypto[,2], order=c(2,0,2))

arima_models<-cbind(model=model$aic,ar1=ar1$aic, ar2=ar2$aic,
ar3=ar3$aic, ar4=ar4$aic, ar5=ar5$aic,
                    ma1=ma1$aic, ma2=ma2$aic, ma3=ma3$aic,
ma4=ma4$aic, ma5=ma5$aic ,
                    arma11=arma11$aic,arma12= arma12$aic,
arma21=arma21$aic,arma22= arma22$aic)
arima_models
```

```
min(arima_models)
```

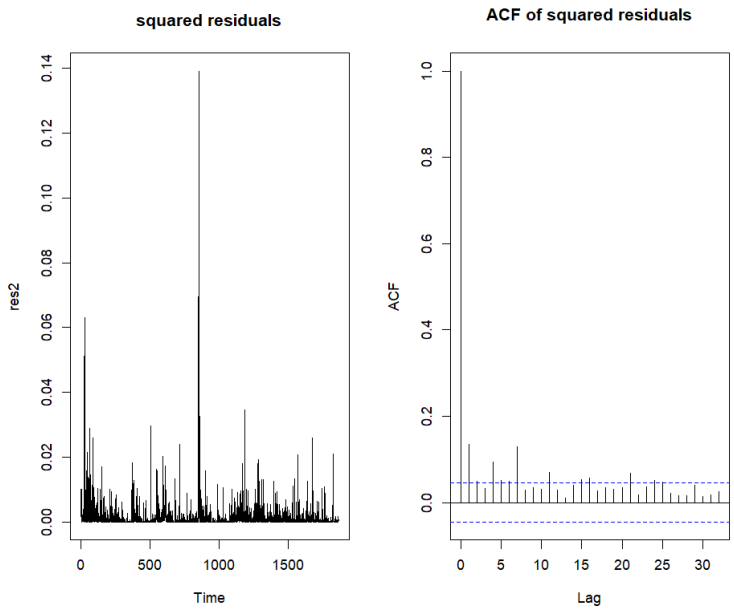
Le modèle qui semble minimiser le critère AIC et le modèle ARMA(0,0). Cependant, le Bitcoin, comme la plupart des séries financières, est caractérisé par une volatilité relativement importante et non-constante. On parle alors d'hétéroscedasticité, et sa présence peut expliquer le fait que les résidus du modèle optimal ne soient pas « parfaits ».

```
checkresiduals(model)
```



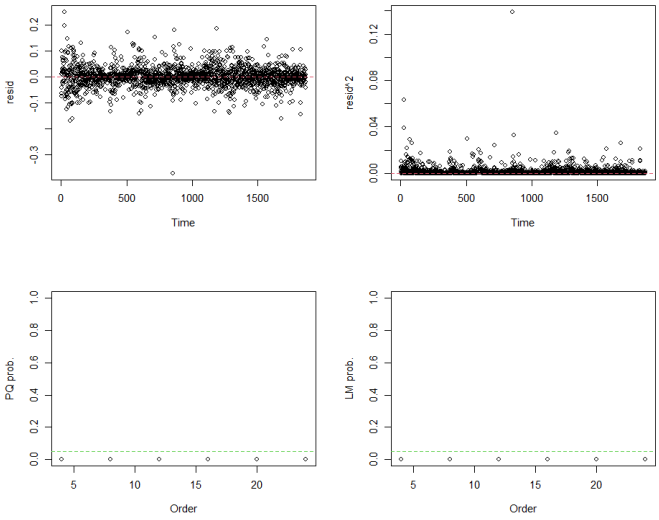
La détection de l'hétéroscédasticité peut se faire par plusieurs méthodes. Notamment, il est possible de la détecter grâce à l'autocorrélogramme : si les résidus au carrés estimés par modèle ARMA optimal sont autocorrélés, cela permet d'affirmer que la série est hétéroscedastique.

```
res2<-model$residuals^2
par(mfrow=c(1,2))
plot(res2, main="squared residuals")
acf(res2, main="ACF of squared residuals")
```



L'autre test de détection de l'hétéroscedasticité, plus formel, est le test ARCH du package **aTSA**.

```
arch.test(model)
```




```
ARCH heteroscedasticity test for residuals
alternative: heteroscedastic

Portmanteau-Q test:
     order    PQ  p.value
[1,]     4  56.5 1.58e-11
[2,]     8  98.8 0.00e+00
[3,]    12 113.1 0.00e+00
[4,]    16 127.8 0.00e+00
[5,]    20 135.5 0.00e+00
[6,]    24 151.9 0.00e+00
Lagrange-Multiplier test:
     order   LM p.value
[1,]     4 3138       0
[2,]     8 1311       0
[3,]    12  852       0
[4,]    16  627       0
[5,]    20  496       0
[6,]    24  403       0
```

Tant au niveau du corrélogramme des résidus que du test ARCH, la présence d'hétéroscedasticité est confirmée : les résidus élevés au carré sont autocorrélés, et la p-value du test Portmanteau et du test LM sont systématiquement significatives (<0.05). La présence d'hétéroscedasticité (ou effets ARCH) ayant été confirmée, cela justifie l'utilisation d'un modèle GARCH pour modéliser la variance de la série. Par ailleurs, la mise en place d'un modèle GARCH implique que nous allons modéliser l'impact de la variance passée sur la variance actuelle

Modèles GARCH

Afin de modéliser un modèle GARCH, il faut d'abord spécifier le modèle (GARCH standard, GARCH exponentiel…) ainsi que les lois de distribution conditionnelles qui lui correspondent (une dizaine de lois de distributions sont détaillées dans le package).

Etant donné que spécifier chaque modèle et chaque distribution conditionnelle est relativement long, la totalité du code est disponible dans les documents associés. Un exemple de spécification de modèle GARCH est présent ci-après, pour un modèle GARCH standard avec une distribution Normale.

```
spec.sgarch.norm<-ugarchspec(mean.model = list(armaOrder = c(0,0)),
variance.model = list(garchOrder = c(1,1), model ="sGARCH"),
distribution.model ="norm")

fit.sgarch.norm<-ugarchfit(spec=spec.sgarch.norm, data=crypto[,2])
```

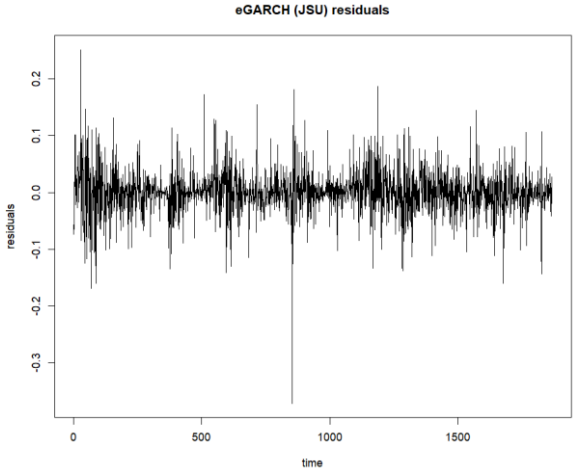
Une fois que la totalité des GARCH ont été spécifiés, il faut comparer les AIC de chaque modèle adapté à notre série de Bitcoin. Le modèle associé à la loi de distribution conditionnelle qui minimisent le critère AIC sera le modèle considéré comme étant le modèle optimal pour modéliser la variance. Le

code fournit également le code pour comparer les AIC. Au final, on peut regrouper les résultats dans le tableau suivant :

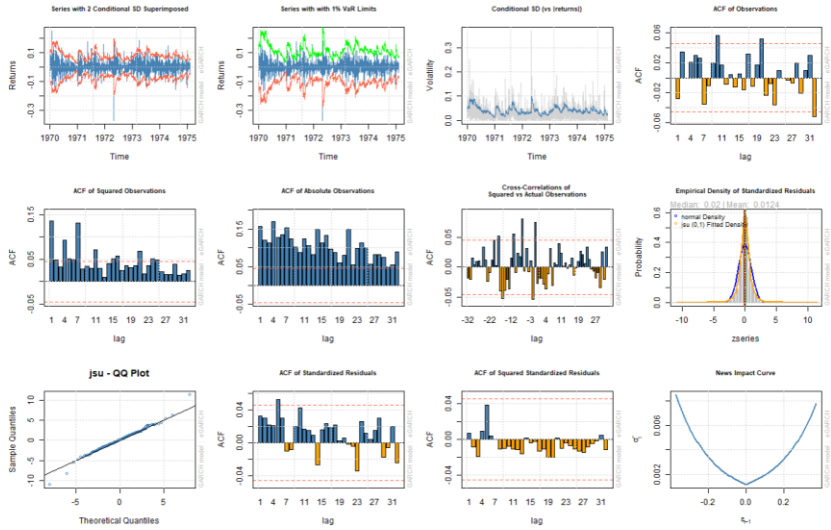
```
Bitcoin returns
====================================
      sGARCH  eGARCH  gjrGARCH tGARCH
------------------------------------
norm  -3.7033 -3.7031 -3.7067  -3.7015
snorm -3.7024 -3.7021 -3.7057  -3.7005
std   -3.941  -3.9508 -3.9399  -3.948
sstd   -3.94  -3.9497 -3.9389  -3.9469
ged   -3.936  -3.9394 -3.9351   2.103
sged  -3.9353 -3.939  -3.9344  2.3327
nig   -3.9437 -3.9486 -3.9426  -3.9469
ghyp  -3.9429 -3.9494 -3.9418  -3.947
jsu   -3.9445 -3.9515 -3.9434  -3.9491
------------------------------------
```

Le meilleur modèle, selon le critère AIC, est le eGARCH(1,1) associé à la loi de distribution Johnson's SU. On peut visualiser les résidus avec le code suivant

```
par(mfrow=c(1,1))
plot(fit.egarch.jsu@fit[["residuals"]],
     type="l",
     main="eGARCH (JSU) residuals",
     ylab="residuals",
     xlab="time")
```



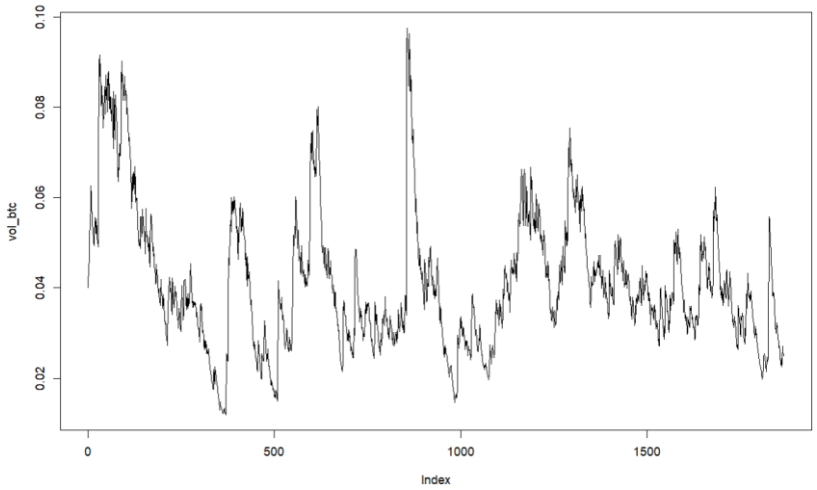
Un ensemble de graphiques (graphiques 10) permet de retracer les grandes lignes des résultats du modèle GARCH estimé. Notamment, on peut observer que notre modèle modélise assez bien la volatilité de la série. Par ailleurs, la distribution estimée colle relativement bien à la densité de la distribution, ce qui peut nous permettre de confirmer le choix du modèle.



Modélisation de la volatilité

Une fois le choix du modèle confirmé, il est possible de visualiser la variance modélisée de la série de Bitcoin, avec la commande :

```
vol_btc<-fit.egarch.jsu@fit[["sigma"]]
par(mfrow=c(1,1))
plot(vol_btc, type="l")
```



On peut relever que la modélisation de la variance est relativement compliquée mais permet de ne pas perdre d'observation, comme ça peut être le cas quand on utilise les écart-types estimés dans des fenêtres roulantes. A noter que la série obtenue est celle de l'écart-type (sigma), qu'on peut élever au carré pour obtenir la série de variance (ou en modifiant `[["sigma"]]` en `[["var"]]` dans la commande précédente

Vous savez maintenant comment modéliser la variance d'une série financière avec des modèles GARCH. Nous ne nous sommes pas arrêté sur l'interprétation des résultats du modèle GARCH parce qu'ils sont trop nombreux, mais plein de ressources en ligne fournissent les explications.