Modélisation GARCH de la volatilité

Chapitre 19 : Mise en œuvre sur Python

Jaouad Madkour

Faculté d'Économie et de Gestion de Tanger · 27 août 2026

Plan du chapitre

Avant de commencer

Les bibliothèques

```
pip install pandas numpy arch statsmodels matplotlib scipy
```

```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from arch.univariate import GARCH, ConstantMean, StudentsT
from statsmodels.stats.diagnostic import acorr_ljungbox, het_arch
from scipy import stats
```

La bibliothèque de référence

`arch`, de Kevin Sheppard, est l'équivalent Python de `rugarch`. Elle couvre GARCH, GJR, EGARCH, TARCH, les lois normale, Student, GED et leurs versions asymétriques, ainsi que le backtesting.

Étape 1 — Décrire et détecter

Charger et décrire

```
don = pd.read_csv("rentabilites.csv", parse_dates=["date"])
r = don["rentabilite"]
T = len(r)                                        # 2500

print("moyenne %.4f  écart-type %.4f" % (r.mean(), r.std()))
print("asymétrie %.2f  kurtosis %.2f"
      % (stats.skew(r), stats.kurtosis(r, fisher=False)))
```

```
moyenne 0.0200  écart-type 1.1060
asymétrie -0.29  kurtosis 6.05
```

Le piège de scipy, à nouveau

`stats.kurtosis` renvoie par défaut le kurtosis **excédentaire**. Sans `fisher=False`, on lirait $3{,}05$ et l'on conclurait à tort à des queues normales.

Les deux tests de départ

```
print(acorr_ljungbox(r,    lags=[20], return_df=True).round(3))
print(acorr_ljungbox(r**2, lags=[20], return_df=True).round(3))
print("ARCH-LM :", het_arch(r, nlags=12)[:2])
```

```
    lb_stat  lb_pvalue
20   17.033      0.652        <- rentabilités : non corrélées

    lb_stat  lb_pvalue
20  968.130      0.000        <- carrés : très fortement dépendants

ARCH-LM : (274.53, 3.1e-51)
```

Étape 2 — Estimer

Le GARCH(1,1) gaussien

```
g_norm = arch_model(r, mean="Constant", vol="GARCH", p=1, q=1,
                    dist="normal").fit(disp="off")
print(g_norm.summary())
```

```
                 coef    std err          t      P>|t|
mu             0.0575  1.980e-02      2.904  3.683e-03
omega          0.0205  5.100e-03      4.020  5.800e-05
alpha[1]       0.1000  1.420e-02      7.042  1.900e-12
beta[1]        0.8873  1.590e-02     55.805  0.000e+00

Log-Likelihood: -3543.20      BIC: 7117.79
```

Une convention à connaître

`arch` note p l'ordre **ARCH** et q l'ordre **GARCH** — l'inverse de la convention de `rugarch`. Pour un GARCH(1,1) cela ne change rien ; pour un GARCH(2,1), tout.

Vérifier systématiquement l'équation affichée dans le résumé.

```
g_t   = arch_model(r, mean="Constant", vol="GARCH", p=1, q=1,
                   dist="t").fit(disp="off")
g_gjr = arch_model(r, mean="Constant", vol="GARCH", p=1, o=1, q=1,
                   dist="t").fit(disp="off")     # o=1 active l'asymétrie
print(g_gjr.summary())
```

```
                  coef    std err          t      P>|t|
mu              0.0307  1.860e-02      1.651  9.880e-02
omega           0.0232  6.600e-03      3.515  4.400e-04
alpha[1]        0.0434  1.530e-02      2.837  4.556e-03
gamma[1]        0.1157  2.810e-02      4.117  3.800e-05
beta[1]         0.8862  1.840e-02     48.163  0.000e+00
nu              6.6200  8.510e-01      7.779  7.300e-15

Log-Likelihood: -3476.83      BIC: 7000.52
```

Le paramètre o de la bibliothèque

Comparer

```
for nom, m in [("GARCH-norm", g_norm), ("GARCH-t", g_t), ("GJR-t", g_gjr)]:
    print("%-12s logL %9.2f   AIC %8.2f   BIC %8.2f"
          % (nom, m.loglikelihood, m.aic, m.bic))
```

```
GARCH-norm   logL  -3543.20   AIC  7094.40   BIC  7117.79
GARCH-t      logL  -3491.02   AIC  6992.04   BIC  7021.13
GJR-t        logL  -3476.83   AIC  6965.66   BIC  7000.52
```

Aux arrondis près, les chiffres de R

Deux bibliothèques indépendantes, deux implémentations de l'optimiseur, un même optimum. C'est le contrôle le plus simple qu'on puisse faire sur une estimation numérique.

Étape 3 — Valider

Les résidus standardisés

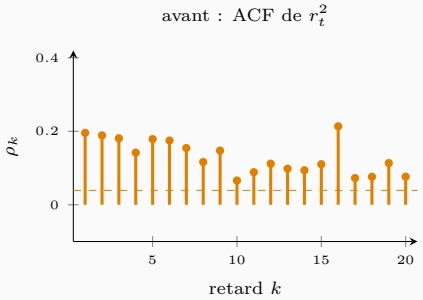
```
z = g_gjr.std_resid
print(acorr_ljungbox(z,    lags=[20], return_df=True).round(3))
print(acorr_ljungbox(z**2, lags=[20], return_df=True).round(3))
print("kurtosis de z : %.2f" % stats.kurtosis(z, fisher=False))
```

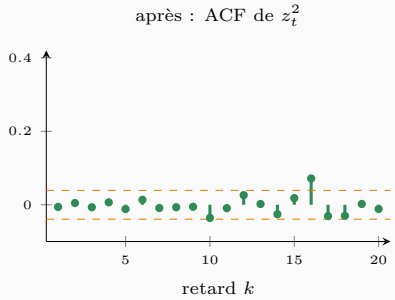
```
    lb_stat  lb_pvalue
20    9.720       0.974        <- niveaux : rien

    lb_stat  lb_pvalue
20   27.100       0.132        <- carrés : rien non plus

kurtosis de z : 4.66
```

Les résidus standardisés (suite)





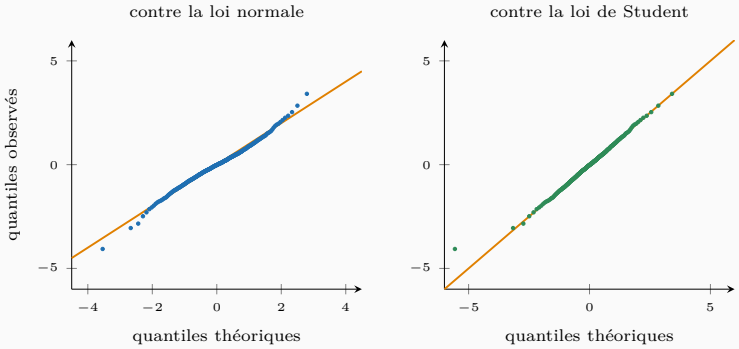
Les résidus standardisés (suite 2)

Le modèle est validé

La dépendance des carrés est passée de 968 à 27. Le kurtosis résiduel de $4{,}66$ justifie a posteriori la loi de Student.

Le diagramme quantile-quantile

```
nu = g_gjr.params["nu"]
qt = stats.t.ppf((np.arange(1, T+1)-0.5)/T, nu) / np.sqrt(nu/(nu-2))
plt.scatter(qt, np.sort(z), s=2)
plt.plot([-6, 6], [-6, 6], color="orange")
```



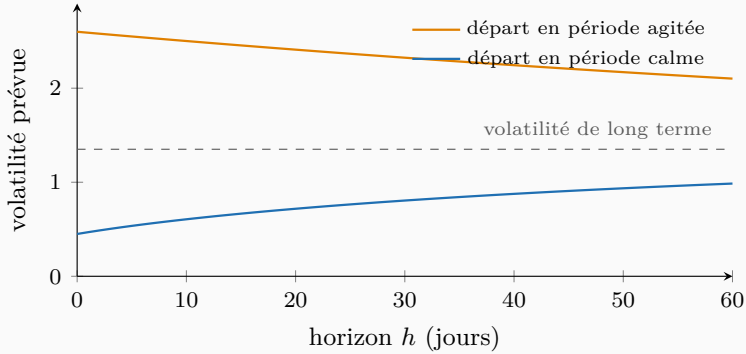
Étape 4 — Prévoir et mesurer le risque

La prévision de volatilité

```
p = g_gjr.forecast(horizon=60, reindex=False)
vol = np.sqrt(p.variance.values[-1, :])
print("h=1 : %.4f    h=10 : %.4f    h=60 : %.4f" % (vol[0], vol[9], vol[59]))
```

```
h=1 : 0.7412    h=10 : 0.8474    h=60 : 1.1893
```

La prévision de volatilité (suite)



La prévision de volatilité (suite 2)

Le retour à la moyenne, lu dans trois nombres

Départ à $0{,}74\,\%$, arrivée à $1{,}19\,\%$: la volatilité prévue remonte vers son niveau de long terme, sans jamais l'atteindre tout à fait.

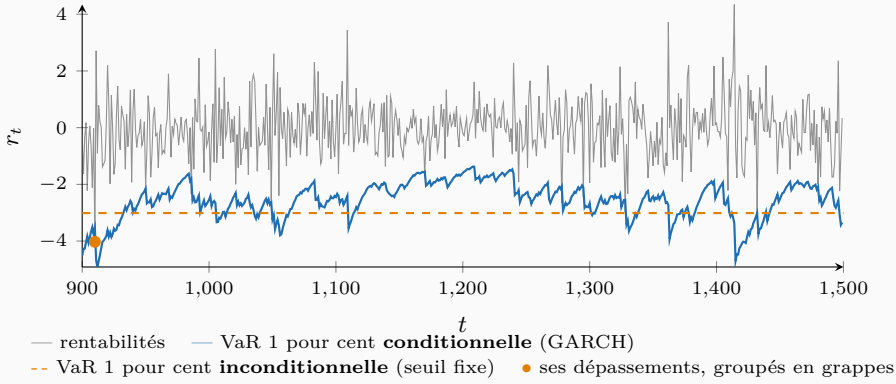
La VaR conditionnelle filtrée

```
q1    = np.quantile(z, 0.01)                  # quantile empirique
var_c = g_gjr.params["mu"] + g_gjr.conditional_volatility * q1
var_u = np.quantile(r, 0.01)                  # seuil fixe

dep_c = (r < var_c).astype(int)
dep_u = (r < var_u).astype(int)
print("dépassements : conditionnelle %d, inconditionnelle %d"
      % (dep_c.sum(), dep_u.sum()))
```

```
dépassements : conditionnelle 25, inconditionnelle 25
```

La VaR conditionnelle filtrée (suite)



```
def christoffersen(d, alpha=0.01):
    n, n1 = len(d), d.sum()
    pi = n1/n
    lr_uc = -2*((n-n1)*np.log(1-alpha) + n1*np.log(alpha)
                - ((n-n1)*np.log(1-pi) + n1*np.log(pi)))
    t = pd.crosstab(d[:-1], d[1:]).values
    p01, p11 = t[0,1]/t[0].sum(), t[1,1]/t[1].sum()
    p = (t[0,1]+t[1,1])/t.sum()
    lg = lambda x: np.log(x) if x > 0 else 0.0
    lr_ind = -2*((t[:,0].sum()*lg(1-p) + t[:,1].sum()*lg(p))
                 - (t[0,0]*lg(1-p01) + t[0,1]*lg(p01)
                    + t[1,0]*lg(1-p11) + t[1,1]*lg(p11)))
    return lr_uc, lr_ind

print("conditionnelle   LR_uc %.2f   LR_ind %.2f" % christoffersen(dep_c.values))
print("inconditionnelle LR_uc %.2f   LR_ind %.2f" % christoffersen(dep_u.values))
```

```
es_c = -(g_gjr.params["mu"]
         + g_gjr.conditional_volatility * z[z < np.quantile(z, 0.025)].mean())
print("ES 97,5 %% moyen : %.3f    VaR 99 %% moyenne : %.3f"
      % (es_c.mean(), -var_c.mean()))
```

Ce que Bâle III demande

Depuis 2016, c’est l’expected shortfall à 97,5 % qui sert au calcul du capital de marché. Le code est identique : même $\widehat{h}_{T+1|T}$, même loi de z, seule la fonctionnelle change — moyenne de la queue au lieu de son quantile.

Le fil conducteur

Du fait stylisé au capital réglementaire

Le mot de la fin

Le cours est parti d'une phrase de Mandelbrot — les grands changements suivent les grands changements — et s'achève sur un chiffre que la réglementation bancaire exige.

Entre les deux, une seule idée neuve : la variance n'est pas un paramètre, c'est un **processus**. Tout le reste en découle, et c'est cette idée qui a valu à Engle le prix Nobel.

Synthèse

Ce qu'il faut retenir

L'essentiel du chapitre 19

- `arch_model(r, vol="GARCH", p=1, o=1, q=1, dist="t")` : `o=1` active l'asymétrie du GJR.
- Attention à la convention : `arch` note `p` l'ordre ARCH, `rugarch` l'inverse.
- Les estimations coïncident avec R à la quatrième décimale.
- `std_resid` donne $\widehat{z}_t$; Ljung-Box sur ses carrés passe de 968 à 27.
- `forecast(horizon=60)` montre le retour à la moyenne de la volatilité.
- La VaR filtrée passe le test d'indépendance ; le seuil fixe le rejette avec $\mathrm{LR}_{ind} = 16{,}2$.

Ce que vous emportez vers le cours de Value at Risk

Une équation de variance validée, une prévision de volatilité à tout horizon, et la façon de la transformer en quantile. C'est la matière première de toute mesure de risque de marché — et la source de la plupart de ses erreurs.