Modélisation GARCH de la volatilité

Chapitre 21 : Étude de cas sur données réelles — Python

Jaouad Madkour

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

Plan du chapitre

Plan du chapitre

Avant de commencer

Le même travail, l'autre chaîne

Pourquoi refaire

Non pour répéter, mais pour **contrôler**. Deux bibliothèques écrites par des équipes différentes, deux optimiseurs différents : si elles concluent la même chose, le résultat ne doit rien à l'outil.

C'est la forme la plus simple de reproductibilité, et la seule qui soit à la portée d'un étudiant.

Les bibliothèques

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

```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
from arch import arch_model
from arch.univariate import ConstantMean, GARCH, StudentsT
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.stats.diagnostic import acorr_ljungbox, het_arch
from scipy import stats
```

Obtenir les données

Voie 1 — Google Finance par Google Sheets

```
=GOOGLEFINANCE("INDEXSP:.INX"; "close"; DATE(2015;1;1); TODAY(); "DAILY")
```

Puis export CSV, et en Python :

```
don = pd.read_csv("donnees.csv", parse_dates=["Date"], dayfirst=True)
don = don.dropna().rename(columns={"Date": "date", "Close": "prix"})
P = don["prix"].astype(float)
```

Voie 2 — téléchargement programmatique

```
brut = yf.download("^GSPC", start="2015-01-01", end="2025-01-01",
                   auto_adjust=True, progress=False)
P = brut["Close"].dropna()
```

Les deux voies (suite)

Le rappel du chapitre précédent

Google Finance n'expose aucune interface de programmation : `yfinance` est le seul moyen d'automatiser le téléchargement. Les cotations de clôture sont les mêmes.

`auto_adjust=True` corrige des dividendes et des divisions d'actions — indispensable sur actions individuelles, sans effet sur un indice de prix.

Un contrôle avant de continuer

```
print(P.index.min().date(), "→", P.index.max().date(), " ", len(P), "cotations")
print("valeurs manquantes :", P.isna().sum())
print("jours dupliqués    :", P.index.duplicated().sum())
```

Trois vérifications qui évitent des heures perdues

La **période** obtenue est-elle celle demandée ? Une erreur de symbole renvoie parfois une série vide sans lever d'exception.

Le **nombre de cotations** : environ 252 par an. Nettement moins signale des jours manquants.

Les **doublons** : rares mais destructeurs, ils faussent toutes les autocorrélations.

Étape 1 — Des prix aux rentabilités

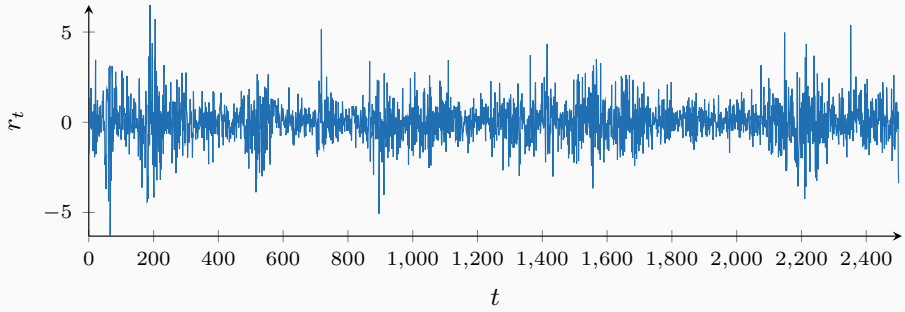
```
r = 100 * np.log(P).diff().dropna()
T = len(r)

print("n %d   moyenne %.4f   écart-type %.4f   annualisé %.1f %%"
      % (T, r.mean(), r.std(), r.std()*np.sqrt(252)))
print("asymétrie %.2f   kurtosis %.2f"
      % (stats.skew(r), stats.kurtosis(r, fisher=False)))
print("Jarque-Bera : %.1f  p %.3g" % stats.jarque_bera(r))
```

Le piège de scipy, une dernière fois

Sans `fisher=False`, `stats.kurtosis` renvoie l'excès. On lirait 3 au lieu de 6 et l'on conclurait à des queues normales.

Regarder

```
fig, ax = plt.subplots(2, 1, figsize=(11, 5), sharex=True)
ax[0].plot(P.index, P, lw=0.7, color="grey");          ax[0].set_ylabel("prix")
ax[1].plot(r.index, r, lw=0.5, color="steelblue");     ax[1].set_ylabel("r (%)")
```



Étape 2 — Stationnarité

```
print("ADF  sur r      : stat %.2f  p %.4f" % adfuller(r, autolag="AIC")[:2])
print("KPSS sur r      : stat %.3f  p %.3f" % kpss(r, regression="c", nlags="auto")[:
print("ADF  sur ln(P)  : stat %.2f  p %.4f" % adfuller(np.log(P), autolag="AIC")[:2])
```

Le verdict attendu

$\ln P_t$: ADF ne rejette pas — série $I(1)$.

r_t : ADF rejette nettement, KPSS ne rejette pas — série $I(0)$.

La différenciation est donc **validée par un test**, non supposée.

Étape 3 — L’équation de moyenne

Étape 3 — L'équation de moyenne

```
print(acorr_ljungbox(r, lags=[20], return_df=True).round(4))

res = []
for p in range(3):
    for q in range(3):
        try:
            m = ARIMA(r, order=(p, 0, q)).fit()
            res.append((p, q, m.aic, m.bic))
        except Exception:
            pass
tab = pd.DataFrame(res, columns=["p", "q", "AIC", "BIC"]).sort_values("BIC")
print(tab.head(3).round(2))

p_opt, q_opt = int(tab.iloc[0]["p"]), int(tab.iloc[0]["q"])
moy = ARIMA(r, order=(p_opt, 0, q_opt)).fit()
e = moy.resid
```

Le test ARCH

```
print("LB sur e   :", acorr_ljungbox(e,    lags=[20], return_df=True).values[0].round
print("LB sur e²  :", acorr_ljungbox(e**2, lags=[20], return_df=True).values[0].round
print("ARCH-LM    :", het_arch(e, nlags=12)[:2])
```

Le contraste, invariable

Sur e_t : non rejeté. Sur e_t^2 : statistique de plusieurs centaines.

C'est le moment exact où le cours ARMA s'arrête et où celui-ci commence.

Étape 4 — Le modèle de variance

Étape 4 — Le modèle de variance

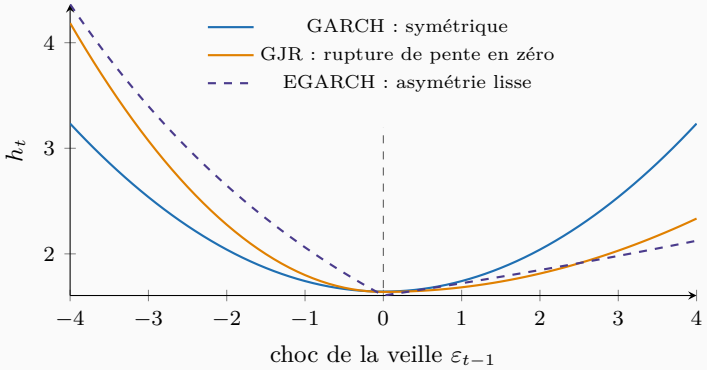
```
def ajuster(o, loi):
    return arch_model(r, mean="ARX", lags=p_opt, vol="GARCH",
                      p=1, o=o, q=1, dist=loi).fit(disp="off")

g1 = ajuster(0, "normal")     # GARCH gaussien
g2 = ajuster(0, "t")          # GARCH Student
g3 = ajuster(1, "t")          # GJR Student

for nom, m in [("GARCH-norm", g1), ("GARCH-t", g2), ("GJR-t", g3)]:
    print("%-12s logL %9.1f   BIC %9.1f" % (nom, m.loglikelihood, m.bic))
print(g3.summary())
```

Le classement attendu

Le BIC décroît à chaque étape : la loi de Student gagne beaucoup, l'asymétrie gagne encore.

Lire les résultats

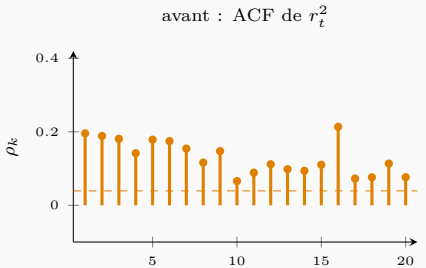
```
a, gm, b = g3.params["alpha[1]"], g3.params["gamma[1]"], g3.params["beta[1]"]
pers = a + gm/2 + b
print("persistance %.4f   demi-vie %.1f jours   nu %.2f"
      % (pers, np.log(0.5)/np.log(pers), g3.params["nu"]))
print("impact d'un choc négatif / positif : %.2f" % ((a+gm)/a))
```

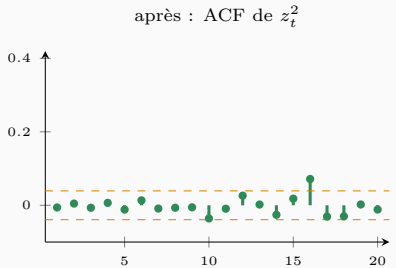


Étape 5 — Valider

Étape 5 — Valider

```
z = g3.std_resid.dropna()
print(acorr_ljungbox(z,    lags=[20], return_df=True).values[0].round(3))
print(acorr_ljungbox(z**2, lags=[20], return_df=True).values[0].round(3))
print("kurtosis de z : %.2f" % stats.kurtosis(z, fisher=False))

nu = g3.params["nu"]
qt = stats.t.ppf((np.arange(1, len(z)+1)-0.5)/len(z), nu) / np.sqrt(nu/(nu-2))
plt.scatter(qt, np.sort(z), s=2); plt.plot([-6, 6], [-6, 6], color="orange")
```



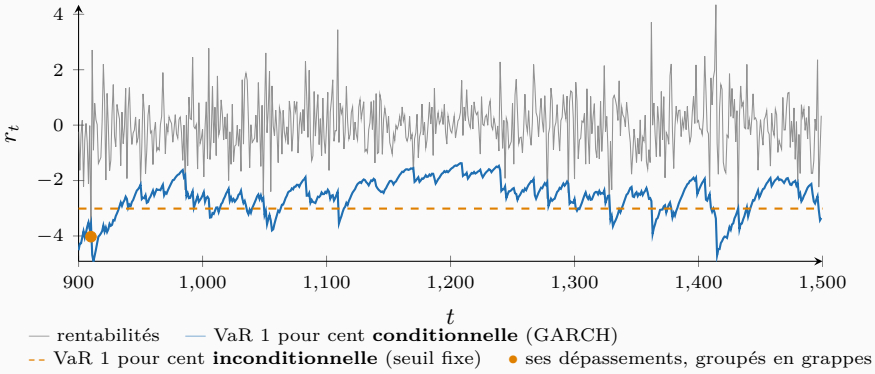


Étape 6 — VaR et backtesting

Les deux mesures

```
q1    = np.quantile(z, 0.01)
h     = g3.conditional_volatility
mu_t  = r - g3.resid                    # moyenne conditionnelle ajustée
var_c = mu_t + h * q1                   # VaR conditionnelle filtrée
var_u = np.quantile(r, 0.01)            # seuil fixe

dep_c = (r < var_c).astype(int).values
dep_u = (r < var_u).astype(int).values
print("dépassements : conditionnelle %d, fixe %d, attendu %.0f"
      % (dep_c.sum(), dep_u.sum(), 0.01*T))
```

Les deux mesures (suite)



```
def backtest(d, alpha=0.01):
    n, n1 = len(d), int(d.sum()); pi = n1/n
    lg = lambda x: np.log(x) if x > 0 else 0.0
    lr_uc = -2*((n-n1)*np.log(1-alpha) + n1*np.log(alpha)
                - ((n-n1)*lg(1-pi) + n1*lg(pi)))
    n00 = n01 = n10 = n11 = 0
    for i in range(1, n):
        a, b = d[i-1], d[i]
        n00 += (a==0 and b==0); n01 += (a==0 and b==1)
        n10 += (a==1 and b==0); n11 += (a==1 and b==1)
    p01 = n01/max(n00+n01, 1); p11 = n11/max(n10+n11, 1)
    p   = (n01+n11)/(n00+n01+n10+n11)
    lr_ind = -2*(((n00+n10)*lg(1-p) + (n01+n11)*lg(p))
                 - (n00*lg(1-p01) + n01*lg(p01) + n10*lg(1-p11) + n11*lg(p11)))
    return lr_uc, lr_ind, lr_uc + lr_ind
```

Kupiec et Christoffersen (suite)

Les seuils

χ^2_1 à 5 % : $3{,}84$ pour LR_{uc} et LR_{ind}.

χ^2_2 à 5 % : $5{,}99$ pour LR_{cc}.

Le résultat typique sur dix ans d'indice

La VaR **inconditionnelle** rate le test d'indépendance de loin : ses dépassements se concentrent sur mars 2020 et sur l'automne 2008 si la période le couvre.

La VaR **conditionnelle** passe les deux tests. C'est la démonstration opérationnelle de tout le cours.

L’expected shortfall

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

Comparer R et Python

Ce qu'il faut vérifier après avoir fait les deux

Quantité	Écart acceptable
coefficients GARCH	3e décimale
log-vraisemblance	moins de $0{,}1$
persistance	4e décimale
nombre de dépassements	identique

Si l'écart est plus grand

Vérifier trois choses avant de soupçonner un bogue : l'**échantillon** est-il exactement le même — attention aux `dropna()` ; la **mise à l'échelle** est-elle la même — facteur 100 ; et l'**équation de moyenne** est-elle la même dans les deux chaînes.

Neuf fois sur dix, l'écart vient de là.

Le fil conducteur

Le fil conducteur

Ce que l'étudiant sait faire à ce stade

Prendre une série de prix qu'il n'a jamais vue, la transformer, tester sa stationnarité, ajuster son équation de moyenne, détecter et modéliser son hétéroscédasticité, valider le résultat, en tirer une VaR et démontrer qu'elle est correcte.

C'est exactement le contenu d'une fiche de poste de *risk analyst* junior.

Synthèse

Ce qu'il faut retenir

L'essentiel du chapitre 21

- `yfinance` pour automatiser, `GOOGLEFINANCE()` dans Sheets pour la voie manuelle.
- Toujours contrôler période, nombre de cotations et doublons avant de calculer.
- `arch_model(..., mean="ARX", lags=p, vol="GARCH", p=1, o=1, q=1, dist="t")` estime l'équation de moyenne et celle de variance **conjointement**.
- Validation : Ljung-Box sur z et sur z^2, QQ-plot contre la Student estimée.
- Le backtesting complet tient en une trentaine de lignes, sans bibliothèque dédiée.
- R et Python doivent coïncider à la troisième décimale ; sinon, chercher l'écart d'échantillon.

Le mot de la fin

Le cours est parti d'une observation de Mandelbrot sur le prix du coton et s'achève sur un script qui mesure le risque d'un indice réel et prouve que sa mesure tient. Entre les deux, une seule idée : la variance est un processus, non un paramètre.