Rapport rédigé par Massinissa SMAÏL, à l'attention de Mme. KORTCHEMSKI

TP n°3 : Calibration du modèle de Vasicek

Enoncés, codes, réponses et conclusions

Année scolaire 2023-2024 *ING3, Génie Mathématique, Option MMF*

Sommaire

Introduction

Dans ce TP, notre attention se portera sur la calibration des courbes des taux d'intérêt. Notre objectif principal est de calibrer ces paramètres de manière à ce que la courbe des rendements que nous allons générer se rapproche au maximum des taux observés sur le marché financier. Nous examinerons la correspondance entre les taux théoriques obtenus à partir de la courbe tracée avec les paramètres calibrés et les taux réels du marché. Si cette concordance est confirmée, nous pourrons conclure que notre calibration est efficace.

Pour réaliser cette calibration, nous nous baserons l'algorithme de Levenberg-Marquardt. Nous nous appuierons aussi sur le modèle de Vasicek, qui offre une dynamique pour les taux "r" en fonction de trois paramètres, qui sont précisément ceux que nous cherchons à calibrer. Dans un premier temps, nous effectuerons cette calibration les prix des obligations zéro-coupon et les rendements du marché. Par la suite, nous répéterons le processus en utilisant les taux "r" du marché.

Partie 1 – Modèle de Vasicek

L'objectif de cette partie est de vérifier les inégalités suivantes :

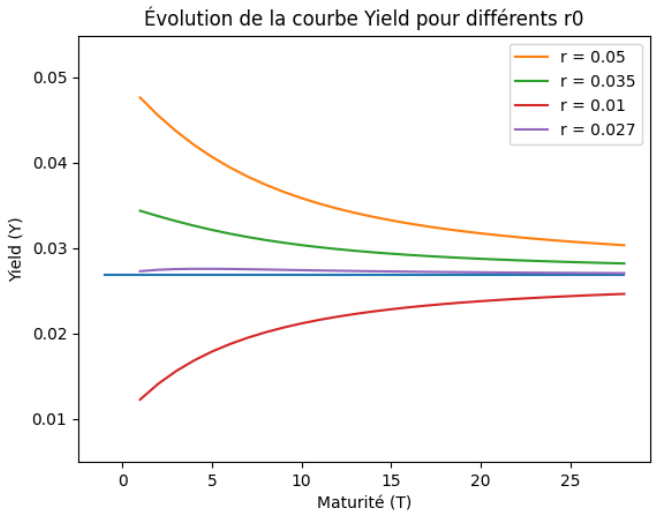
$$\lim_{T\to 0} Y(0,T) = r_0, \lim_{T\to\infty} Y(0,T) = \frac{\eta}{\gamma} - \frac{1}{2}(\frac{\sigma}{\gamma})^2$$

Pour cela, nous avons implémenté le code suivant :

```
N = 30
t = 0
gamma = 0.25
eta = 0.25 * 0.03
sigma = 0.02
r = 0.05
r1 = 0.035
r2 = 0.01
rs = 0.027

T = np.zeros(N)
lim = np.zeros(N)
B = np.zeros(N)
A = np.zeros(N)
Y = np.zeros(N)
Y1 = np.zeros(N)
Y2 = np.zeros(N)
Y3 = np.zeros(N)

limite = (eta / gamma) - 0.5 * (sigma / gamma) ** 2
```

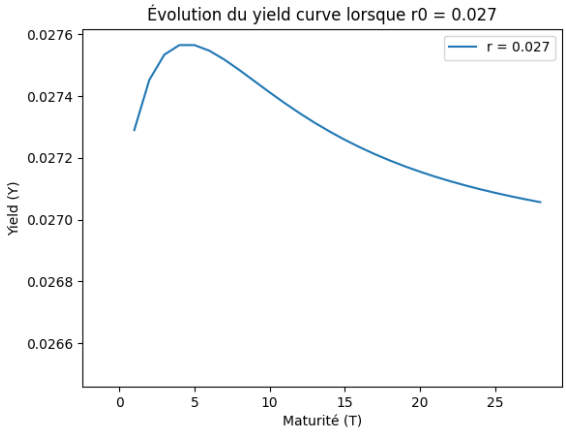
```
for i in range(N):
    T[i] = i - 1
    lim[i] = limite
    B[i] = (1 / gamma) * (1 - np.exp(-gamma * T[i]))
    A[i] = (B[i] - (T[i] - t)) * ((eta * gamma - (sigma ** 2 / 2)) / gamma ** 2) - (sigma * B[i]) ** 2 / (4 * gamma)

    if (T[i] - t) != 0:# On vérifie que le dénominateur est différent de 0, afin d'éviter une erreur
        Y[i] = -(A[i] - r * B[i]) / (T[i] - t)
        Y1[i] = -(A[i] - r1 * B[i]) / (T[i] - t)
        Y2[i] = -(A[i] - r2 * B[i]) / (T[i] - t)
        Y3[i] = -(A[i] - rs * B[i]) / (T[i] - t)
    else:# Gérer le cas où le dénominateur vaut effectivement 0
        Y[i] = np.nan
        Y1[i] = np.nan
        Y2[i] = np.nan
        Y3[i] = np.nan

plt.plot(T,lim)
plt.plot(T, Y, label='r = 0.05')
plt.plot(T, Y1, label='r = 0.035')
plt.plot(T, Y2, label='r = 0.01')
plt.plot(T, Y3, label='r = 0.027')
plt.legend()
plt.xlabel('Maturité (T)')
plt.ylabel('Yield (Y)')
plt.title("Évolution de la courbe Yield pour différents r0")
plt.show()

plt.figure()
plt.plot(T, Y3, label='r = 0.027')
plt.legend()
plt.xlabel('Maturité (T)')
plt.ylabel('Yield (Y)')
plt.title("Évolution du yield curve lorsque r0 = 0.027")
plt.show()
```

Grâce au code ci-dessus, nous traçons les courbes de rendements en fonction du temps à partir du modèle de Vacisek pour différentes valeurs de r_o.



Nous observons que pour toutes les valeurs de r_o implémentées, les courbes convergent vers la limite asymptotique théorique (représenté par la courbe bleue). Ce qui valide la deuxième égalité que nous souhaitions vérifier.

Nous isolons maintenant la courbe de rendement pour $r_o = 0.027$, nous obtenons la représentation suivante :



Ce graphique montre que pour T qui tend vers $+\infty$, la limite de cette courbe de rendement est égale à 0.027, qui est aussi la valeur de r_o. Autrement dit, la première égalité que nous souhaitions est bien vérifiée.

Partie 2 & 3 – Calibrage de la courbe de rendement

Nous utilisons l'algorithme de Levenberg-Marquart afin d'obtenir les paramètres minimums de β_1, β_2 et β_3 qui nous permettrons ensuite de tracer la courbe Y calibré, fonction de T et de β.

L'énoncé nous fournit les égalités $\beta^1 \equiv \eta, \quad \beta^2 \equiv \sigma^2, \quad \beta^3 = \gamma$ suivantes :

Nous traçons d'abord Y marché à partir du tableau fourni en énoncé :

i	1	2	3	4	5	6	7	8	9	10
maturité T_i	3	6	9	12	15	18	21	24	27	30
yield Y_i^{market}	0.035	0.041	0.0439	0.046	0.0484	0.0494	0.0507	0.0514	0.052	0.0523

Grâce au code suivant, nous calibrons la courbe de rendement :

```
t = 0
r0 = 0.023

eta = 1
sigma = 1
gamma = 1
sigma_carre = sigma ** 2
beta = [eta, sigma_carre, gamma]

epsilon = 10 ** (-9)
lambda_1 = 0.01

d = [1, 1, 1]
J = np.array([np.zeros(3) for i in range(10)])

Yt_final = np.zeros(10)
Res = np.zeros(10)
T = np.linspace(3, 30, 10)

Y_market = [0.035, 0.041, 0.0439, 0.046, 0.0484, 0.0494, 0.0507, 0.0514, 0.052, 0.0523]
```

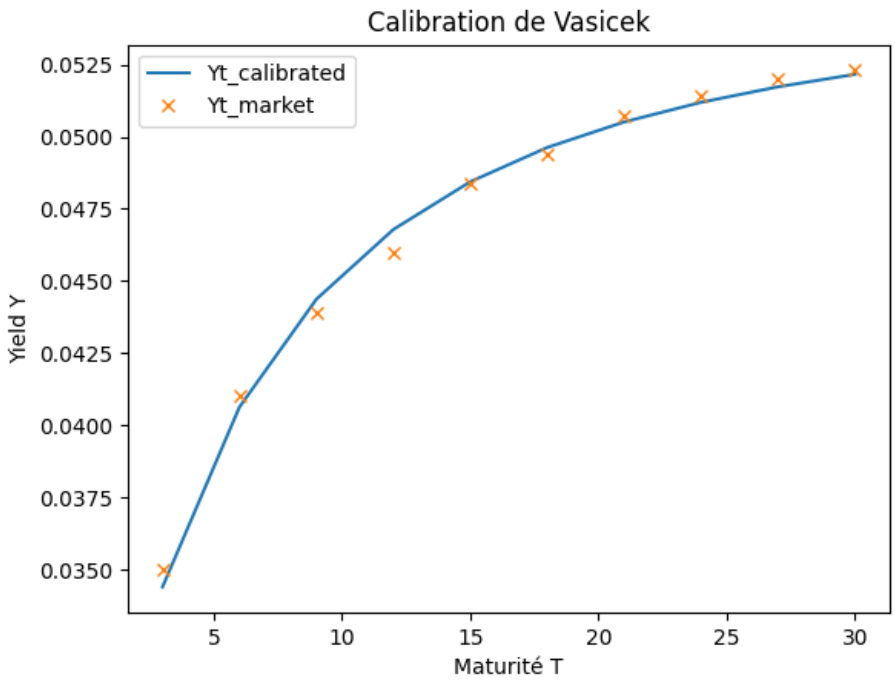
```
def fonction_B(t, T, Gamma):
    return (1 - exp(-Gamma * (T - t))) / Gamma


def fonction_A(t, T, Eta, Sigma, Gamma):
    return (fonction_B(t, T, Gamma) - (T - t)) * (Eta * Gamma - 0.5 * Sigma ** 2) / (Gamma ** 2) - (Sigma ** 2) * (
        fonction_B(t, T, Gamma)) ** 2 / (4 * Gamma)


while LA.norm(d) > epsilon: #Levenberg-Marquart
    for i in range(10):
        B = fonction_B(t, T[i], beta[2])
        A = fonction_A(t, T[i], beta[0], cmath.sqrt(beta[1]), beta[2])
        DB = ((T[i] - t) * exp(-(T[i] - t) * beta[2]) - B) / beta[2]
        DA = (beta[0] * (DB * beta[2] - B) + (T[i] - t) * beta[0] - beta[1] / 2 * (DB - 2 * B / beta[2]) - (T[i] - t) *
              beta[1] / beta[2] - beta[1] * B / 4 * (2 * beta[2] * DB - B)) / beta[2] ** 2

        J[i][0] = (B - (T[i] - t)) / (T[i] - t) / beta[0]
        J[i][1] = -((B - (T[i] - t)) / (2 * beta[2]) + (B ** 2) / 4) / ((T[i] - t) * beta[2])
        J[i][2] = (DA - r0 * DB) / (T[i] - t)
        Yt = -(A - r0 * B) / (T[i] - t)
        Res[i] = Y_market[i] - Yt

    d = np.dot(-LA.inv(np.dot(J.transpose(), J) + lambda_1 * np.identity(3)), np.dot(J.transpose(), Res))
    beta = beta + d.transpose()
    eta = eta + d[0]
    sigma_carre = sigma_carre + d[1]
    gamma = gamma + d[2]
```

```
plt.plot(T, Yt_final, label='Yt_calibré')
plt.plot(T, Y_market, 'x', label='Yt_marché')
plt.legend()
plt.xlabel('Maturité (T)')
plt.ylabel('Yield (Y)')
plt.title('Calibration de Vasicek')
plt.show()

print('Eta = ', eta)
print('Sigma² = ', sigma_carre)
print('Gamma = ', gamma)
```

A partir du code précédent, et en prenant comme conditions initiales **t = 0 et r_0 = 0.023**, nous obtenons les paramètres suivants :

```
Eta =  0.015375669687660663
Sigma² =  0.001418209435075013
Gamma =  0.21539706938385134
```

Ainsi que le graphique suivant :



Nous remarquons que notre modèle calibré est crédible puisqu'il passe par tous les points du marché

Nous effectuons la même démarche mais cette fois-ci avec des points de marchés et des conditions initiales qui diffèrent de ceux que nous avons défini précédemment.

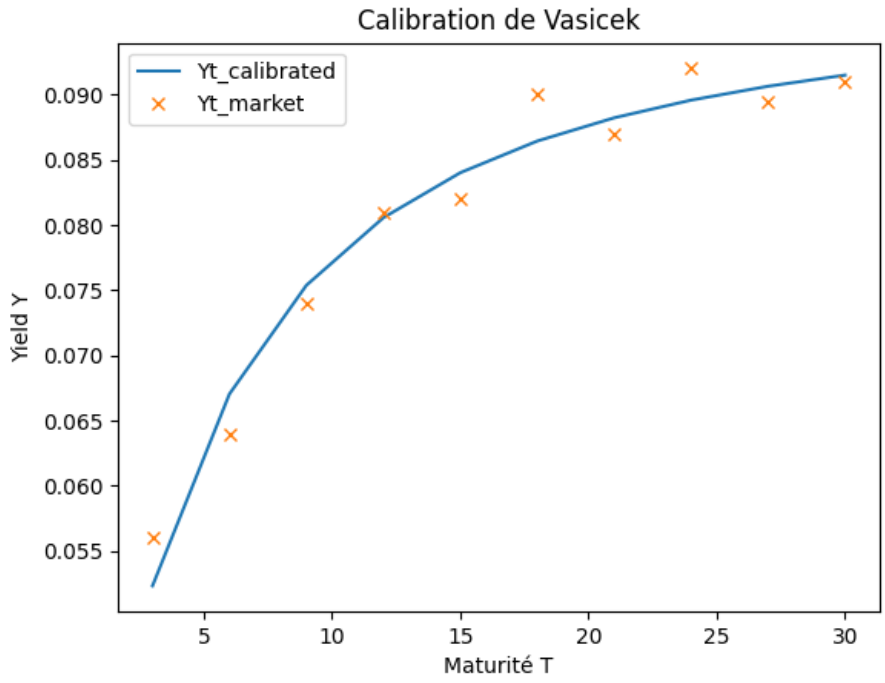
Nous avons le tableau suivant :

i	1	2	3	4	5	6	7	8	9	10
maturité T_i	3	6	9	12	15	18	21	24	27	30
yield Y_i^{market}	0.056	0.064	0.074	0.081	0.082	0.09	0.087	0.092	0.0895	0.091

Le code utilisé est aussi le même que celui que nous avons utilisé précédemment, à la différence de la liste Y_market dont on a modifié les valeurs.

Comme demandé, nous prenons cette fois-ci comme conditions initiales **t = 1 et r_0 = 0.04**, nous obtenons les paramètres finaux et le graphique suivant :

```
Eta =  0.025439844343425447
Sigma² =  0.0009662822667774977
Gamma =  0.23328211568064652
```



Comme précédemment, nous remarquons notre courbe est bien calibrée puisqu'elle est proche de tous les points du marché.

Partie 4 – Calibration avec les données historiques

Toujours à partir du modèle Vasicek, nous allons cette fois calibrer notre modèle avec les dates historiques.

Pour cette partie nous avons implémenté le code suivant :

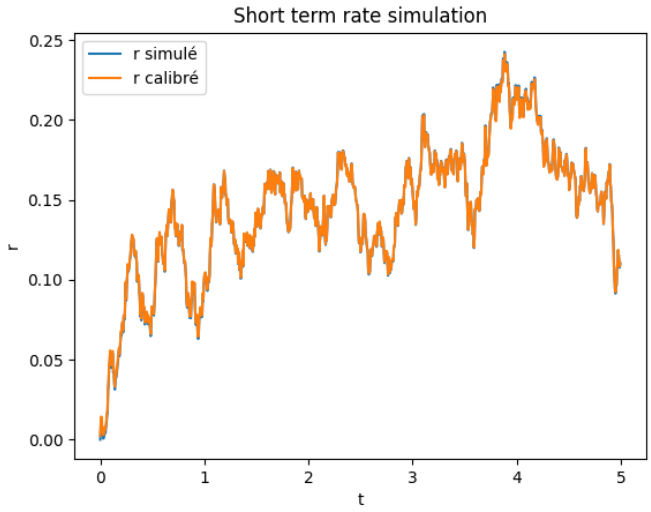
```
r0 = 0
N = 1000
T = 5
delta_T = T / N

Eta = 0.6
Sigma = 0.08
Gamma = 4

r = [r0]
t = [0]

beta = [1, 1]
d = [1, 1]

lambda_1 = 0.1
epsilon = 10 ** (-4)

J = np.array([np.zeros(2) for i in range(N)])
r_final = np.zeros(N)
Res = np.zeros(N)
```

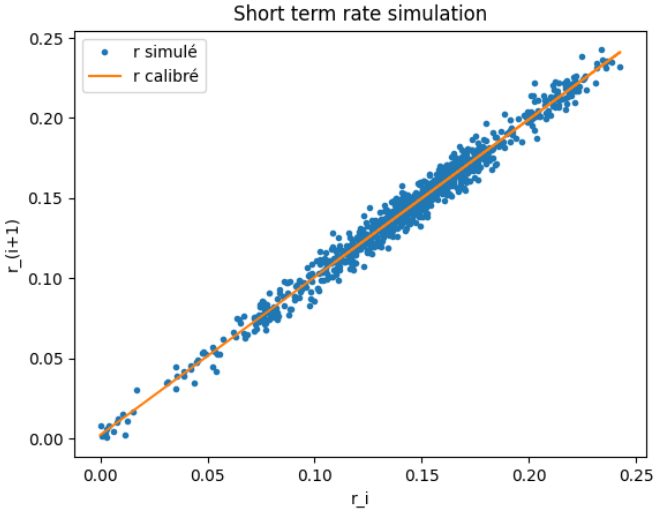
```
while LA.norm(d) > epsilon:
    t = [0]
    r = [r0]
    for i in range(N - 1):
        t.append(t[-1] + delta_T)
        r.append(r[-1] * exp(-Gamma * delta_T) + Eta / Gamma * (1 - exp(-Gamma * delta_T)) + sqrt(
            (Sigma ** 2) / (2 * Gamma) * (1 - exp(-2 * Gamma * delta_T))) * rm.gauss(0, 1))
        J[i][0] = -r[i]
        J[i][1] = -1
        Res[i] = r[-1] - (beta[0] * r[-2] + beta[1])

    d = np.dot(-LA.inv(np.dot(J.transpose(), J) + lambda_1 * np.identity(2)), np.dot(J.transpose(), Res))
    beta = beta + d.transpose()

for i in range(N):
    r_final[i] = beta[0] * r[i] + beta[1]

print('Les coefficients de la regression : [a b] = ' + str(beta))

D = sqrt(np.var(Res))
print('D = ' + str(D))

Gamma_cal = -log(beta[0]) / delta_T
print('Gamma = ' + str(Gamma_cal))

Eta_cal = Gamma_cal * beta[1] / (1 - beta[0])
print('Eta = ' + str(Eta_cal))

Sigma_cal = D * sqrt(-2 * log(beta[0]) / (delta_T * (1 - beta[0] ** 2)))
print('Sigma = ' + str(Sigma_cal))
```

```
plt.plot(t, r, label='r simulé')
plt.plot(t, r_final, label='r calibré')
plt.legend()
plt.xlabel('t')
plt.ylabel('r')
plt.title('Short term rate simulation')
plt.show()

plt.plot(r[0:N - 1], r[1:N], '.', label='r simulé')
plt.plot(r, r_final, label='r calibré')
plt.legend()
plt.xlabel('r_i')
plt.ylabel('r_(i+1)')
plt.title('Short term rate simulation')
plt.show()
```

Nous obtenons les paramètres finaux et les graphs suivants :

```
Les coefficients de la regression : [a b] = [0.98497944 0.00227604]
D = 0.005750590759742569
Gamma = 3.0269027592764837
Eta = 0.45866114111590545
Sigma = 0.0819418166924979
```





Nous observons que notre calibration est efficace, car avec les paramètres calibrés, les courbes de taux simulés et calibrés sont presque superposées.