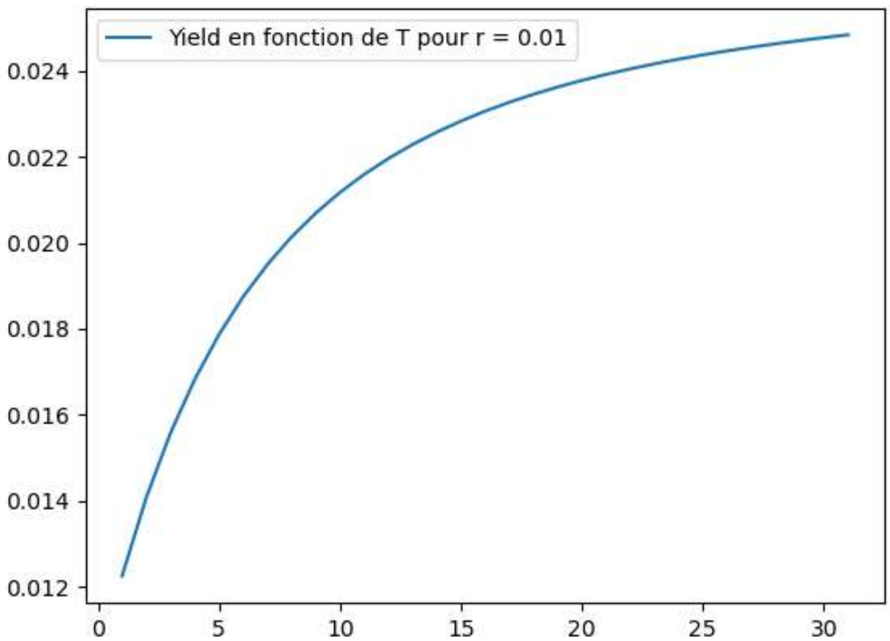
# Vasicek

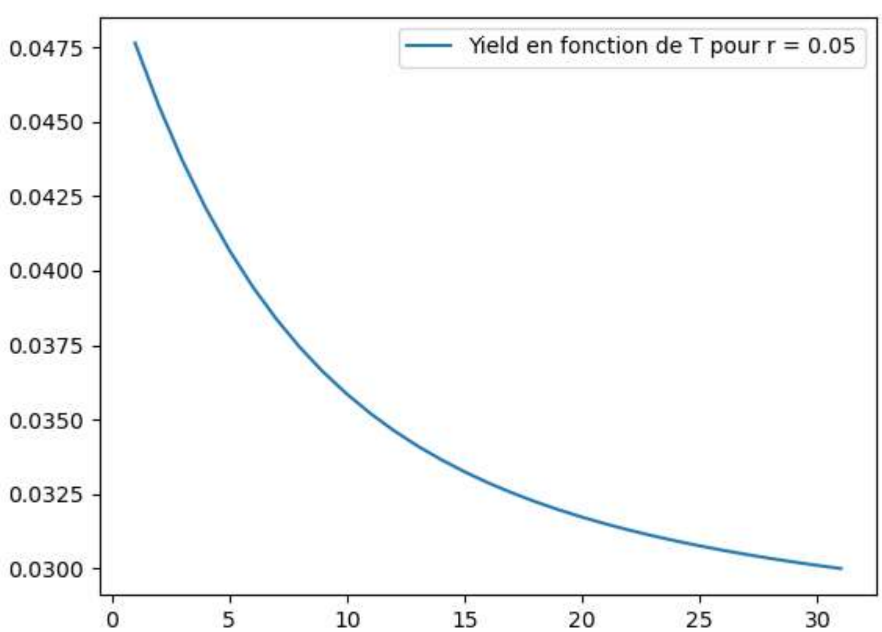
## Yield curve

Ces graphes représentent les courbes de Yields pour différents r donnés :

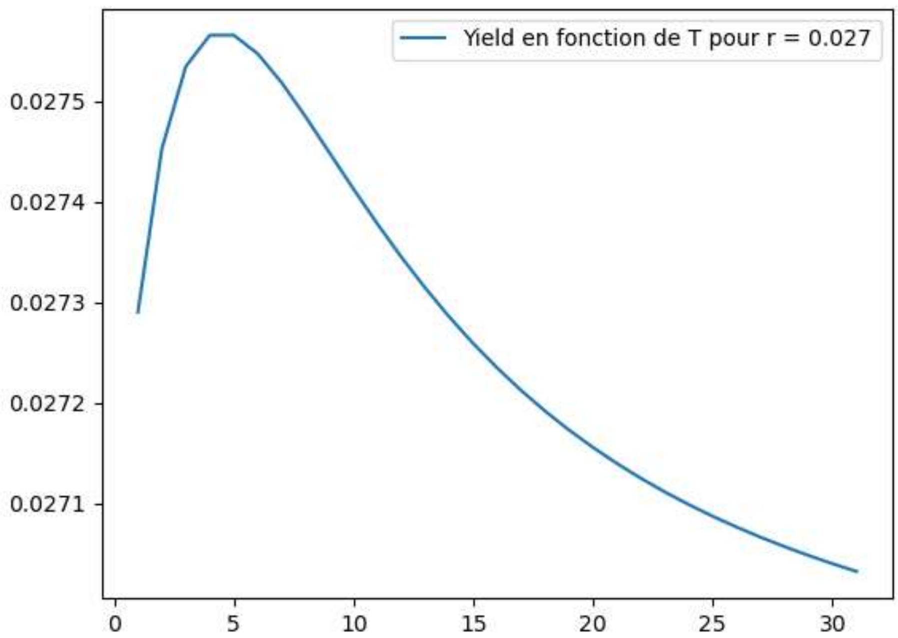
### Typical :




### Inverted



## Slightly




```
def B(t,T,gamma):

    Tau = T - t
    return (1-math.exp(-gamma * Tau))/gamma

def A(t, T, gamma, etha, sigma):

    Tau = T - t
    return (B(t, T, gamma) - Tau) * (gamma * etha - 0.5 *
(sigma**2))/(gamma**2) - ((sigma * B(t, T, gamma))**2)/(4 * gamma)


def Y(r, t, T, gamma, etha, sigma):

    Tau = T - t
    if (T - t) == 0:
        return 1
    else:
        return - (A(t, T, gamma, etha, sigma) - r * B(t, T, gamma)) / Tau

def Yield():

    gamma = 0.25
    etha = 0.25 * 0.03
    sigma = 0.02
```

```python
    r1 = 0.01
    r2 = 0.027
    r3 = 0.05

    T = np.zeros(31)
    Yield1 = np.zeros(31)
    Yield2 = np.zeros(31)
    Yield3 = np.zeros(31)

    for i in range(31):
        T[i] = i + 1
        Yield1[i] = Y(r1, 0, T[i], gamma, etha, sigma)
        Yield2[i] = Y(r2, 0, T[i], gamma, etha, sigma)
        Yield3[i] = Y(r3, 0, T[i], gamma, etha, sigma)

    #plt.plot(T, Yield2, label="Yield en fonction de T pour r = 0.027")
    #plt.plot(T, Yield1, label="Yield en fonction de T pour r = 0.01")
    plt.plot(T, Yield3, label="Yield en fonction de T pour r = 0.05")
    plt.legend()
    plt.show()

    print(Yield1[0])
    print(Yield1[30])
    limite_infinie = etha/gamma - 0.5 * ((sigma / gamma) ** 2)
    print(limite_infinie)
    if Yield1[30] - limite_infinie < 0.01:
        print("converge vers limite infine ")
```

Le programme retourne ses valeurs-ci pour r1 = 0.01. Yield1 correspond à r1.

**Yield[1] = 0.012248558821245528**

**Yield1[30] 0.02483946559940831**

**L est égale: 0.026799999999999997**

**Yield converge vers r lorsque T tend 0**

**Yield converge vers L lorsque T tend vers + infini**

On remarque que la limite de **Yield** lorsque **T** tend vers **0** est bien **r1** dans ce cas.

En notant **L** la quantité dans le TP, on a aussi **Yield** converge vers **L** lorsque **T** tend vers **plus l'infini**.

NB : Les mêmes résultats ont été trouvé pour r =0.05 et r = 0.027.

# Calibration de Yield Curve :

## Calibration à t=0 :

A t = 0 et pour r = 0.023,

On initialise les valeurs de sigma, gamma et etha à 1.

L'algorithme de **Levenberg-Marquart** permet de rendre les valeurs calibrées de ces paramètres, tout en minimisant la somme des résidus au carré dûe à la diférence entre les valeurs du marché donnés à **t=0** et les valeurs théoriques calculées à partir de **T** et les dernières valeurs de paramètres obtenues à chaque itération.

```
def Vasicek():

    r = 0.023
    epsilon = 0.000000001
    Lamda = 0.01



    Y_Mar = [0.035, 0.041, 0.0439, 0.046, 0.0484, 0.0494, 0.0507, 0.0514,
0.052, 0.0523] ## Vecteur de taille 10
    T = np.array([3, 6, 9, 12, 15, 18, 21, 24, 27, 30]) ## Vecteur de taille
10
    Y_th =[]
    Res = []
    J = np.zeros((10, 3))



    Identite = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    D = np.array([[1, 1, 1]]).T
    etha = 1
    gamma = 1
    sigma_carre = 1

    k = 0
    while (np.linalg.norm(D) > epsilon) :

        #print(np.linalg.norm(D))


        for p in range(10):

            Y_th.append(Y(r, 0, T[p], gamma, etha, cmath.sqrt(sigma_carre)))
            Res.append(Y_Mar[p] - Y_th[p])
            Res_Matrix = np.array([Res]).T ## changer en matrice
            J[p][0] = derivee_etha(0, T[p], gamma)
            J[p][1] = derivee_sigma_carre(0, T[p], gamma)
            J[p][2] = derivee_gamma(r, 0, T[p], gamma, etha,
cmath.sqrt(sigma_carre))

        D = - (np.linalg.inv(J.T@ J+ Lamda * Identite))@ J.T @ Res_Matrix  ##
```

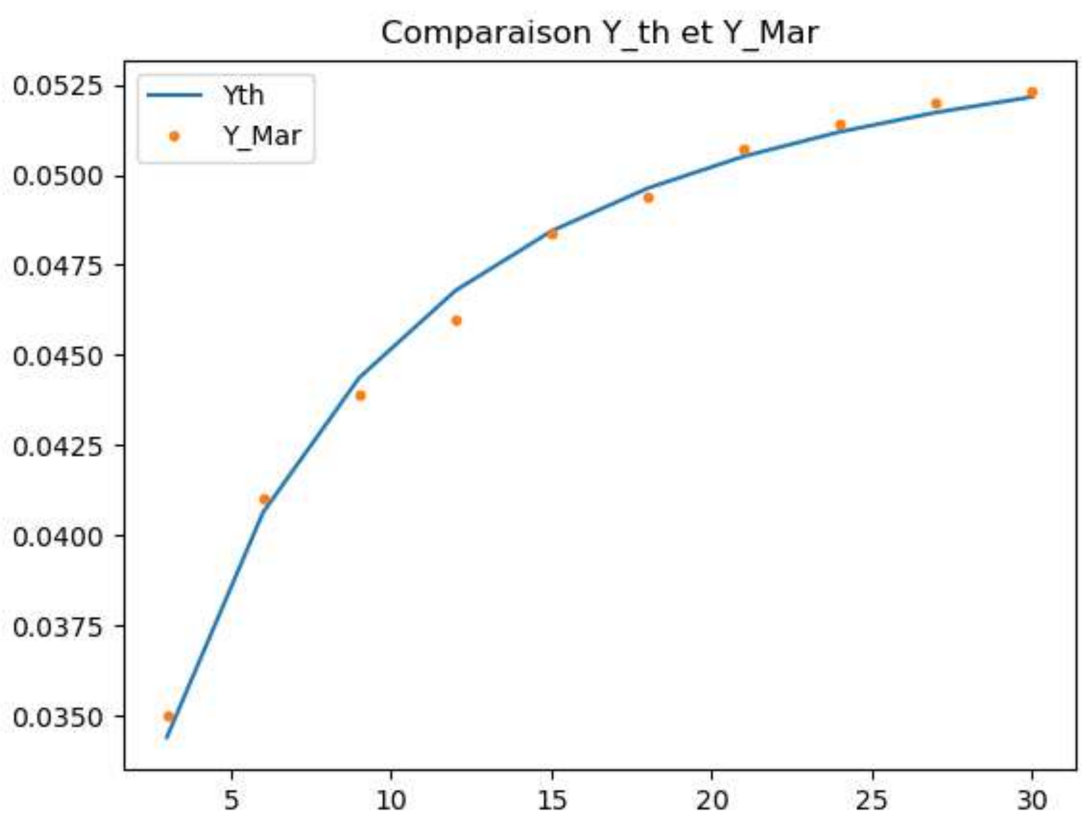
```
Produit matriciel

        J = np.zeros((10, 3))
        Y_th.clear()
        Res.clear()

        etha += D[0][0]
        sigma_carre += D[1][0]
        gamma += D[2][0]

        k = k + 1
    #Beta = [etha, sigma_carre, gamma]
    print(k)
    print("etha",etha,)
    print("sigma", cmath.sqrt(sigma_carre))
    print("gamma",gamma)


    ### Comparer Y_th et Y_Mar
    for i in range(10):
        Y_th.append(Y(r, 0, T[i], gamma, etha, cmath.sqrt(sigma_carre)))
    plt.plot(T, Y_th, label="Yth" )
    plt.plot(T, Y_Mar, '.', label="Y_Mar")
    plt.legend()
    plt.title("Comparaison Y_th et Y_Mar")
    plt.show()
```

Le programme retourne les valeurs ci-dessous.

**etha (0.015375669366269313)**

**sigma (0.03765912128124674)**

**gamma (0.2153970729250022)**



Ensuite, on compare Y_Mar à Y_théorique. On remarque que les valeurs de Y_Mar sont très proches du modèle théorique de Vsicek pour les valeus de sigma, gamma et etha calibrées.

## Calibration à t =1

On refait le même travail pout t = 1.

Dans ce cas les valeurs calibrées sont :

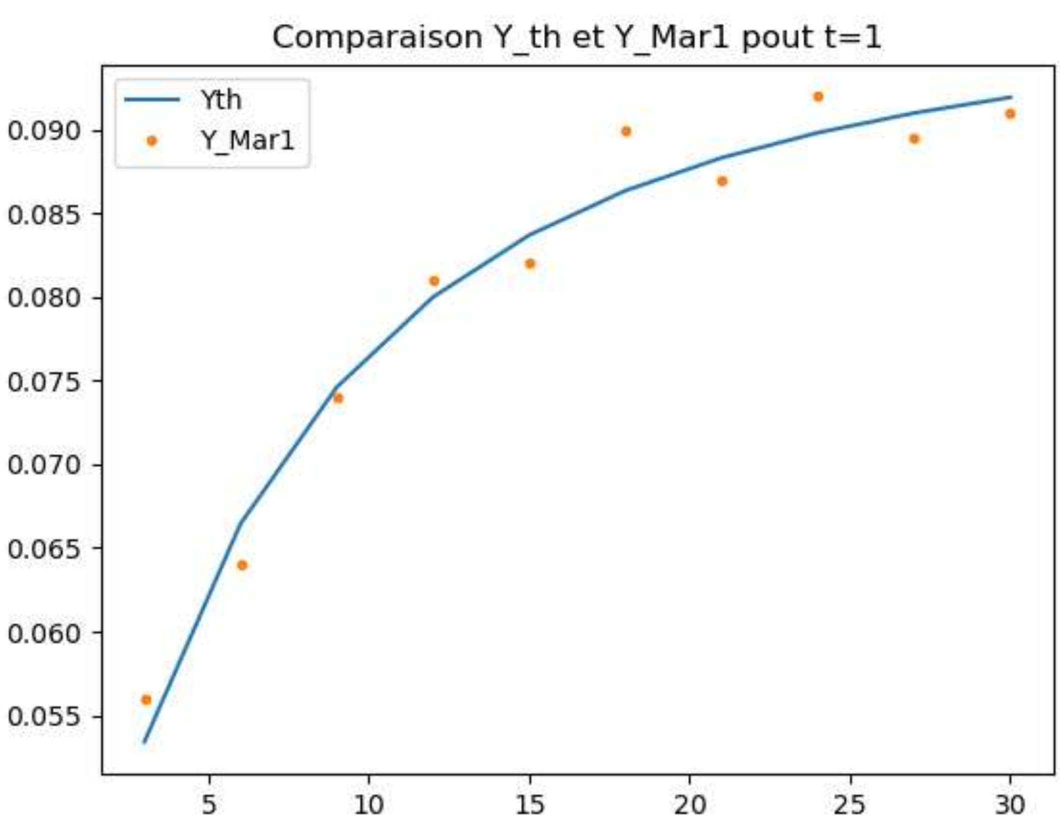
**etha (0.02025152675101919)**

**sigma (0.05046657279765208)**

**gamma (0.2908727757231634**)

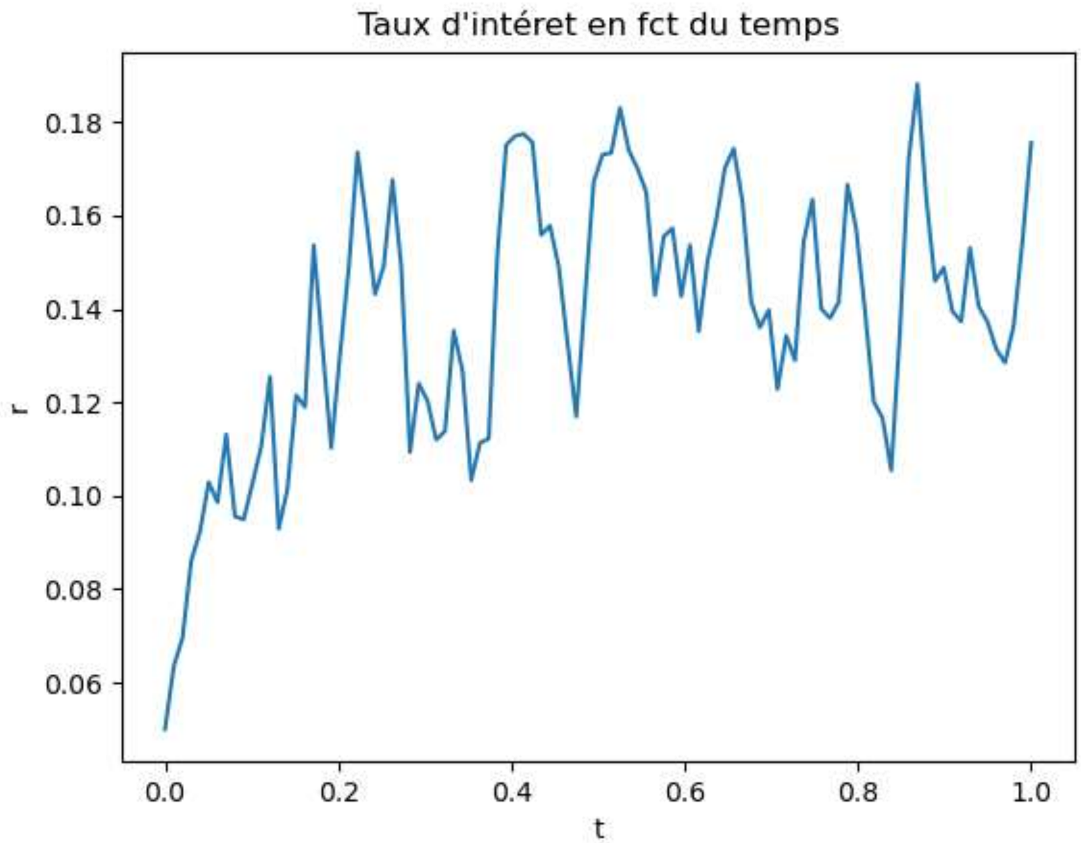
```
r = 0.04
epsilon = 0.000000001
Lamda = 0.01



Y_Mar = [0.035, 0.041, 0.0439, 0.046, 0.0484, 0.0494, 0.0507, 0.0514, 0.052,
0.0523] ## Vecteur de taille 10
Y_Mar1 = [0.056, 0.064, 0.074, 0.081, 0.082, 0.09, 0.087, 0.092, 0.0895,
0.091]
T = np.array([3, 6, 9, 12, 15, 18, 21, 24, 27, 30]) ## Vecteur de taille 10
Y_th =[]
Res = []
J = np.zeros((10, 3))



Identite = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
D = np.array([[1, 1, 1]]).T
etha = 1
gamma = 1
sigma_carre = 1

k = 0
while (np.linalg.norm(D) > epsilon) :

    #print(np.linalg.norm(D))


    for p in range(10):

        Y_th.append(Y(r, 1, T[p], gamma, etha, cmath.sqrt(sigma_carre)))
        Res.append(Y_Mar1[p] - Y_th[p])
        Res_Matrix = np.array([Res]).T ## changer en matrice
```

```
        J[p][0] = derivee_etha(1, T[p], gamma)
        J[p][1] = derivee_sigma_carre(1, T[p], gamma)
        J[p][2] = derivee_gamma(r, 1, T[p], gamma, etha,
cmath.sqrt(sigma_carre))

    D = - (np.linalg.inv(J.T@ J+ Lamda * Identite))@ J.T @ Res_Matrix  ##
Produit matriciel

    J = np.zeros((10, 3))
    Y_th.clear()
    Res.clear()

    etha += D[0][0]
    sigma_carre += D[1][0]
    gamma += D[2][0]

    k = k + 1
#Beta = [etha, sigma_carre, gamma]
print(k)
print("etha",etha,)
print("sigma", cmath.sqrt(sigma_carre))
print("gamma",gamma)


### Comparer Y_th et Y_Mar
for i in range(10):
    Y_th.append(Y(r, 1, T[i], gamma, etha, cmath.sqrt(sigma_carre)))
plt.plot(T, Y_th, label="Yth" )
plt.plot(T, Y_Mar1, '.', label="Y_Mar")
plt.legend()
plt.title("Comparaison Y_th et Y_Mar pout t=1")
plt.show()
```



# Calibration to historical dates

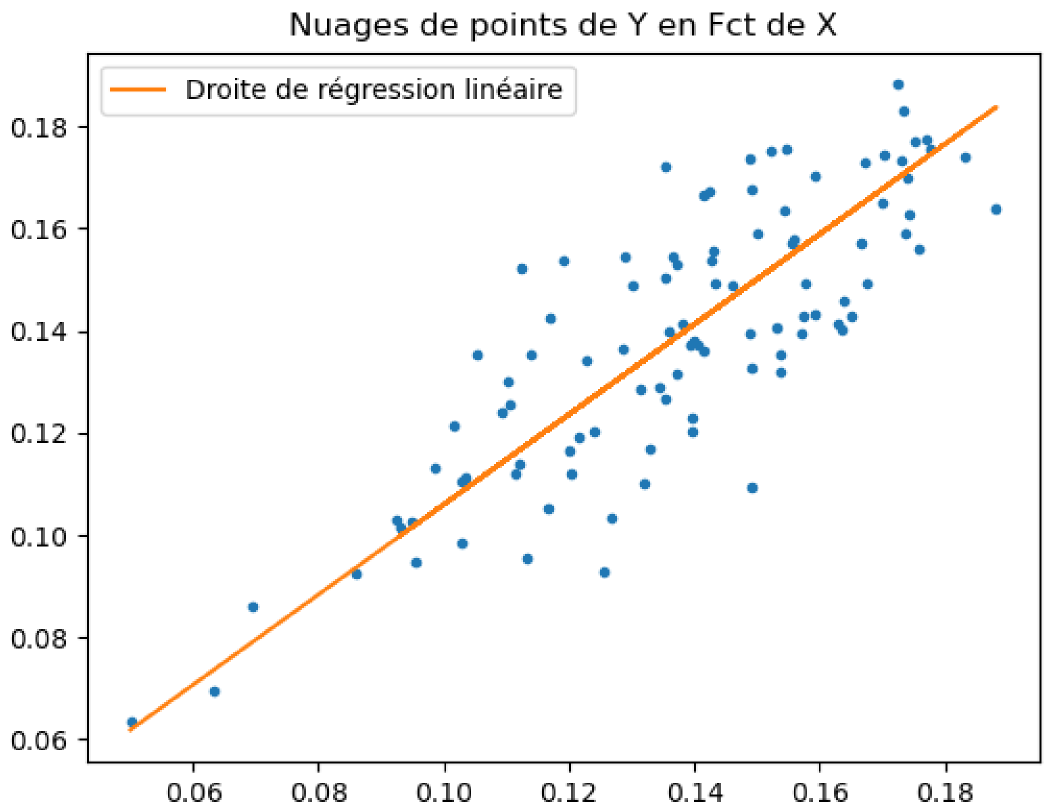
```
ef calibration_hist_dates():

    T = 5
    etha = 0.6
    gamma = 4
    sigma = 0.08
    N = 100
    delta_t = T/N
    X = []
    X_carre = []
    Y = []
    Y_th = []
    Z = []
    r = np.zeros(100)
    t = np.linspace(0, 1, 100)
    r[0] = 0.05

    for i in range(99):

        X.append(r[i])
        X_carre.append(r[i] ** 2)
        Y_th.append(X[i] * 0.8817867419193822 + 0.017808406981631)
        r[i+1] = r[i] * math.exp(-gamma * delta_t) + etha * (1 - math.exp(-
gamma * delta_t))/gamma + sigma * math.sqrt((1 - math.exp(-2 * gamma *
delta_t))/(2 * gamma)) * np.random.normal(0, 1)
        Y.append(r[i + 1])
        Z.append(r[i] * r[i + 1])


    plt.plot(t, r)
    plt.xlabel("t")
    plt.ylabel("r")
    plt.title("Taux d'intéret en fct du temps")
    plt.show()

    plt.plot(X, Y, '.')
    plt.plot(X, Y_th, label="Droite de régression linéaire")
    plt.legend()
    plt.title("Nuages de points de Y en Fct de X")
    plt.show()
```




A l’aide de la formule théoriques, on simule l’évolution du Taux d’intérêt en fonction du temps t.

On trouve la courbe ci-dessus.

## Nuage de points




On trace le nuage de point de Y en fonction de X avec X qui contient les valeurs de r[i] et Y qui contient r[i+1] (Voir le code ci-dessus).

En minimisant la somme du carré du résidu entre **r[i+1] et beta1 * r[i] + beta2** à l'aide de Leveberg-Marquart, on trouve les valeurs respectives de a=beta1 et b= beta2 (les paramètres de la droite de régression).

Fianlement, le programme retourne :

**La valeur de Beta1 est: 0.814384304725029**

**La valeur de Beta2 est: 0.028271495702542327**

```
D_carre = 0
for i in range(99):
    D_carre += 0.01 * (Y[i] - (beta1 * X[i] + beta2))**2
print("D au carré est:",D_carre)

gamma = - math.log(beta1) / delta_t
```

```
etha = gamma * beta2 / (1 - beta1)
sigma = cmath.sqrt(-D_carre * 2 * math.log(beta1) / (delta_t * 1 - beta1**2))
print("etha est : ",etha)
print("gamma est:", gamma)
print("sigma est :",sigma)
```

A l’aide du calcul de D_carre et les formules du TP, on trouve les valeurs de gamma, etha et sigma.

Le programme retourne alors :

**D au carré est: 0.00022525648684036285**

**etha est :  0.4954752456991101**

**gamma est: 3.518502200812601**

**sigma est : 0.011013689573376341**