TP_noté

26 Mars 2024

1 TP Noté Méthodes de Simulation

1.0.1 Manon CHARTRIN, Colin COËRCHON

[115]:
```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy.stats import norm,binom,gaussian_kde,t
from scipy.integrate import quad
from mpl_toolkits.mplot3d import Axes3D

def rand():
    return random.random()

def tirage_va_Discrete_Uniforme(valeurs, N=1):
    n=len(valeurs)
    cum_proba = np.concatenate(([0], np.cumsum(np.ones(n)*1/n)))
    res = np.zeros(N, dtype = int)

    for k in range(N):
        U = rand()
        for i in range(n):
            if U >= cum_proba[i] and U < cum_proba[i + 1]:
                res[k] = valeurs[i]
    if N==1: return res[0]
    else: return res

def tirage_va_Discrete(valeurs, proba, N=1):
    cum_proba = np.concatenate(([0], np.cumsum(proba)))
    res = np.zeros(N)

    for k in range(N):
        U = rand()
        for i in range(len(valeurs)):
            if U >= cum_proba[i] and U < cum_proba[i + 1]:
                res[k] = valeurs[i]
```

```
    if N==1 : return res[0]
    else: return res

def simuler_gaussienne_bidimensionnelle():
    u1 = rand()
    u2 = rand()
    R = -2*math.log(u1)
    Theta = u2*2*math.pi
    return [math.sqrt(R)*math.cos(Theta), math.sqrt(R)*math.sin(Theta)]

def simuler_loi_normale(mu = 0, sigma = 1):
    x = simuler_gaussienne_bidimensionnelle()[0]
    return sigma*x + mu
```

2 Exercice 1

2.0.1 Question 1

Soit X une variable aléatoire prenant ses valeurs dans {-5;-7;2;6} avec :

$$\begin{cases} p1 = P(X = -5) = 1/4 \\ p2 = P(X = -7) = 1/4 \\ p3 = P(X = 2) = 1/8 \\ p4 = P(X = 6) = 3/8 \end{cases}$$

On calcule la fonction de répartition F_X de X et son inverse F_X^{-1} définie sur $u \in]0, 1[$:

$$F_X(x) = \begin{cases} 0 & \text{si } x < -7 \\ \frac{1}{4} & \text{si } x \in [-7, -5[\\ \frac{1}{2} & \text{si } x \in [-5, 2[\\ \frac{5}{8} & \text{si } x \in [2, 6[\\ 1 & \text{si } x \geq 6 \end{cases}$$

$$F_X^{-1}(u) = \begin{cases} -7 & \text{si } u \in]0, \frac{1}{4}] \\ -5 & \text{si } u \in]\frac{1}{4}, \frac{1}{2}] \\ 2 & \text{si } u \in]\frac{1}{2}, \frac{5}{8}] \\ 6 & \text{si } u \in]\frac{5}{8}, 1[\end{cases}$$

[116]:
```
def loi_discrete(E, P):
    F = np.cumsum(P)
    i = 0
    u = np.random.rand()
```

```
    while u > F[i] and i < len(E) - 1:
        i += 1

    return E[i]
```

2.0.2 Question 2

Soit $\mu = E[X]$ et $\sigma^2 = Var(X)$

On va calculer l'espérance et la variance de notre loi discrète :

2.0.3 Espérance :

$E[X] = -5 * \frac{1}{4} - 7 * \frac{1}{4} + 2 * \frac{1}{8} + 6 * \frac{3}{8} = \frac{-10-14+2+18}{8} = \frac{-24+20}{8}$

Donc $\mu = E[X] = -\frac{1}{2}$

2.0.4 Variance :

$Var(X) = E[X^2] - E[X]^2$

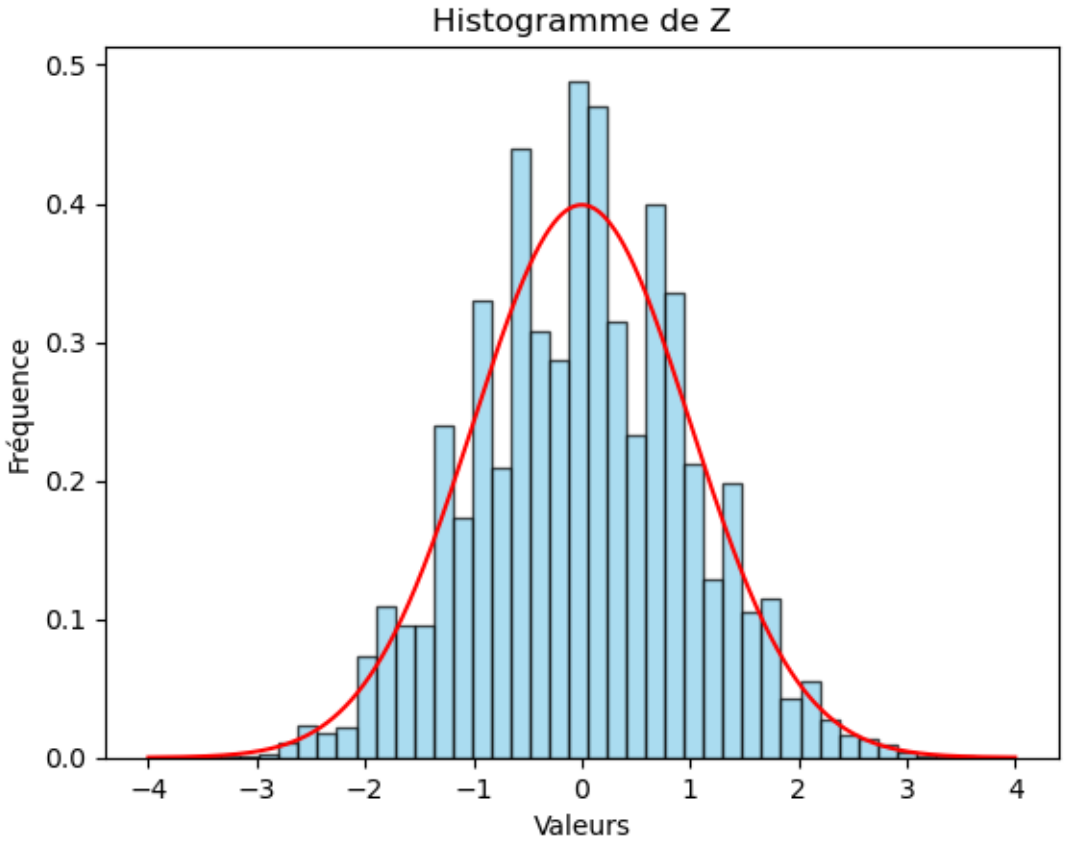
On commence par calculer $E[X^2]$:

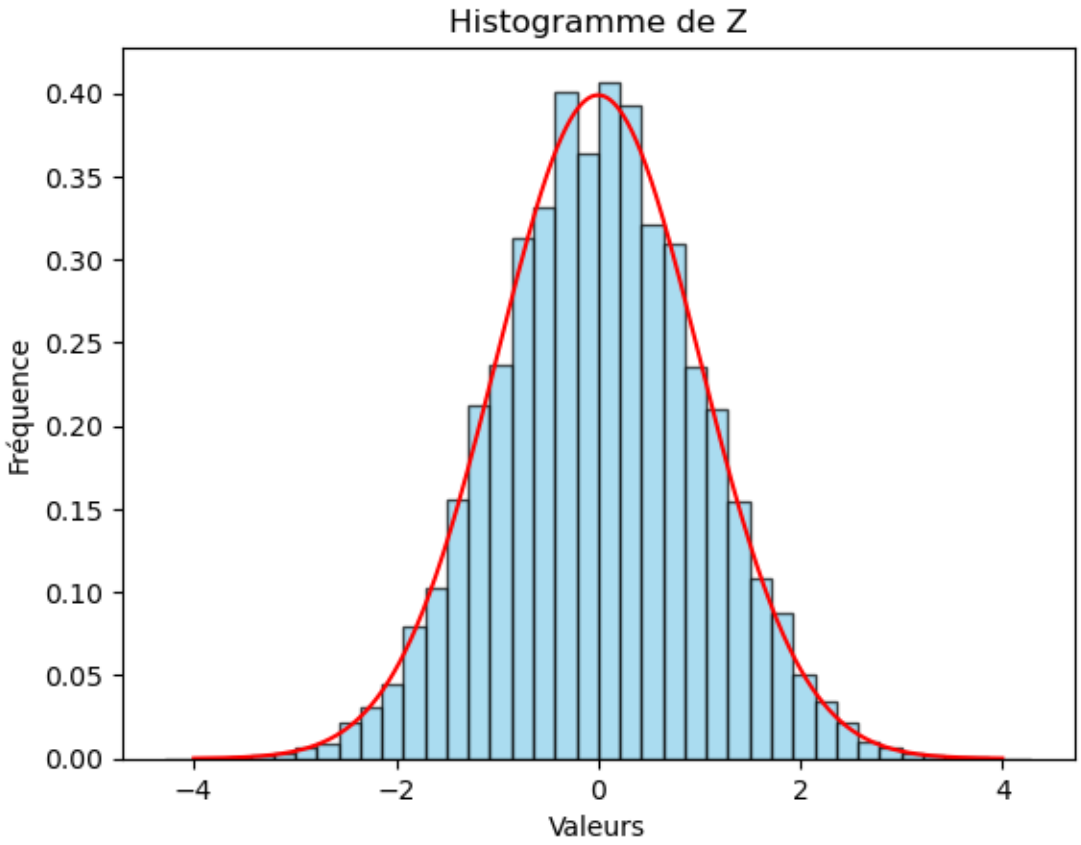
$E[X] = 25 * \frac{1}{4} + 49 * \frac{1}{4} + 4 * \frac{1}{8} + 36 * \frac{3}{8} = \frac{50+98+4+108}{8} = \frac{260}{8} = 32.5$

Ainsi, $Var(X) = E[X^2] - E[X]^2 = 32.5 - 0.25 = 32.25$

[118]:
```
# Valeurs prises par la variable aléatoire
E = [-5, -7, 2, 6]
# les poids associés à ces valeurs
P = [1/4, 1/4, 1/8, 3/8]


# densité d'une loi normale centrée réduite
def densite_normale_standard(x):
    return (1 / np.sqrt(2 * np.pi)) * np.exp(-x**2 / 2)


# mu et var ont été calculés juste au dessus :
mu = -0.5
var = 32.25

N = 10**5

petit_n = [10, 50]
for t in petit_n:
    plt.clf()

    Z = []
    for _ in range(N):
```

```
        somme = sum(loi_discrete(E,P) for _ in range(t))
        moy_emp = somme / t
        Z.append((moy_emp - mu) / (math.sqrt(var / t)))

    # On trace l'histogramme
    plt.hist(Z, bins=40, color='skyblue', edgecolor='black', density=True,␣
↪alpha=0.7)


    # On trace la densité de la loi normale standard sur le même graphe
    x = np.linspace(-4, 4, 1000)
    plt.plot(x, densite_normale_standard(x), color='red', label='Densité normale␣
↪standard')


    # Ajout des légendess et du titre
    plt.xlabel('Valeurs')
    plt.ylabel('Fréquence')
    plt.title('Histogramme de Z')

    plt.show()
```





On remarque que plus n est grand et plus l'histogramme se rapproche de la densité théorique

2.1 Exercice 2

2.1.1 Question 1

$C_0 = E[g(X)]$ avec X qui suit une loi normale centrée réduite et où $g(x) = e^{-rT} max(x_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}x} - K, 0)$

On veut montrer que $C_0 = x_0\phi(d1) - Ke^{-rT}\phi(d2)$ avec ϕ la fonction de répartition de la loi normale centrée réduite
et où $d1 = \frac{log(x_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$ et $d2 = d1 - \sigma\sqrt{T}$

On note $f(x)$ la densité de la fonction normale centrée réduite et $v = \frac{log(K/x_0)-(r-\sigma^2/2)T}{\sigma\sqrt{T}}$

$$
\begin{aligned}
C_0 = E[g(X)] &= \int_{\mathbb{R}} g(x) f(x)\, dx \\
&= \int_{\mathbb{R}} x_0 e^{-\frac{\sigma^2}{2}T+\sigma\sqrt{T}x} f(x) 1_{\{x\geq v\}}\, dx - Ke^{-rT}\int_{\mathbb{R}} f(x) 1_{\{x\geq v\}}\, dx \\
&= x_0 \int_v^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{\sigma^2}{2}T+\sigma\sqrt{T}x-\frac{x^2}{2}}\, dx - Ke^{-rT}\int_v^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx \\
&= x_0 \int_v^{+\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-1}{2}(x-\sigma\sqrt{T})^2}\, dx - Ke^{-rT}\int_v^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx
\end{aligned}
$$

On fait le changement de variable $y = x - \sigma\sqrt{T}$.

Ainsi :

$$
C_0 = x_0 \int_{v-\sigma\sqrt{T}}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}\, dy - Ke^{-rT}\int_v^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx
$$

Or, pour la loi normale centrée réduite, $f(x) = f(-x)$. Donc :

$$
\int_{-\infty}^{-x} f(x)\, dx = \int_x^{+\infty} f(x)\, dx
$$

Ainsi :

$$
C_0 = x_0\phi(-(v-\sigma\sqrt{T})) - Ke^{-rT}\phi(-v)
$$

En réorganisant :

$$
C_0 = x_0\phi(\sigma\sqrt{T}-v) - Ke^{-rT}\phi(-v)
$$

Or :

$$
\sigma\sqrt{T} - v = \frac{\sigma^2 T}{\sigma\sqrt{T}} - \frac{\log(K/x_0) - (r-\frac{\sigma^2}{2})T}{\sigma\sqrt{T}}
$$

Ce qui donne :

$$
\sigma\sqrt{T} - v = \frac{\sigma^2 T - \log(K/x_0) + rT - \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}
$$

En simplifiant :

$$
\sigma\sqrt{T} - v = \frac{-\log(K/x_0) + rT + \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}
$$

Et finalement :

$$\sigma\sqrt{T} - v = \frac{\log(x_0/K) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = d_1$$

On a donc $\sigma\sqrt{T} - v = d_1$ d'où $-v = d1 - \sigma\sqrt{T} = d2$ d'après la définition de d2 de l'énoncé.

On a montré avant le calcul de $\sigma\sqrt{T} - v$ que $C_0 = x_0\phi(\sigma\sqrt{T} - v) - Ke^{-rT}\phi(-v)$

Donc : $C_0 = x_0\phi(d1) - Ke^{-rT}\phi(d2)$

2.1.2 Question 2

[ ]:
```
x0 = 25
T = 1
K = 25
r = 0.03

def g(x,sigma):
    expr = x0 * math.exp((r-(sigma**2 / 2))*T + sigma*math.sqrt(T)*x) - K
    expr = np.max([expr, 0])
    return math.exp(-r*T) * expr

def simuler_Vec_C0(sigma,N):
    X = [simuler_loi_normale() for _ in range(N)]
    C0 = [g(Xk,sigma) for Xk in X]
    return C0

def fonction_C0(sigma):
    d2 = (math.log(x0/K) + T*(r-sigma**2 / 2))/(sigma*math.sqrt(T))
    d1 = d2 + sigma*math.sqrt(T)
    return x0 * norm.cdf(d1) - K * math.exp(-r*T) * norm.cdf(d2)

def calculer_variance_empirique(sigma,N):
    X = [simuler_loi_normale() for _ in range(N)]
    C0 = [g(Xk,sigma) for Xk in X]
    M = np.mean(C0)
    Vec_Var = [(g(Xk,sigma) - M)**2 for Xk in X]
    return 1/(N-1) * np.sum(Vec_Var)

def variance_empirique(X, N):
    M = np.mean(X)
    Vec_Var = [(Xk - M)**2 for Xk in X]
    return 1/(N-1) * np.sum(Vec_Var)

for sigma in [0.1, 0.95]:
    print("Pour sigma =", sigma, "on a :")
    C0 = fonction_C0(sigma)
```

```
    print("C0 =", round(C0,3))
    for N in [1000, 10000]:
        Vec_C0 = simuler_Vec_C0(sigma,N)
        C0_exp = np.mean(Vec_C0)
        Erreur = abs(C0_exp - C0)
        Var_emp = variance_empirique(Vec_C0, N)
        print("Pour N =", N, "on a : Erreur =", round(Erreur,3), "et Variance␣
↪empirique =", round(Var_emp,3))
    print("")
```

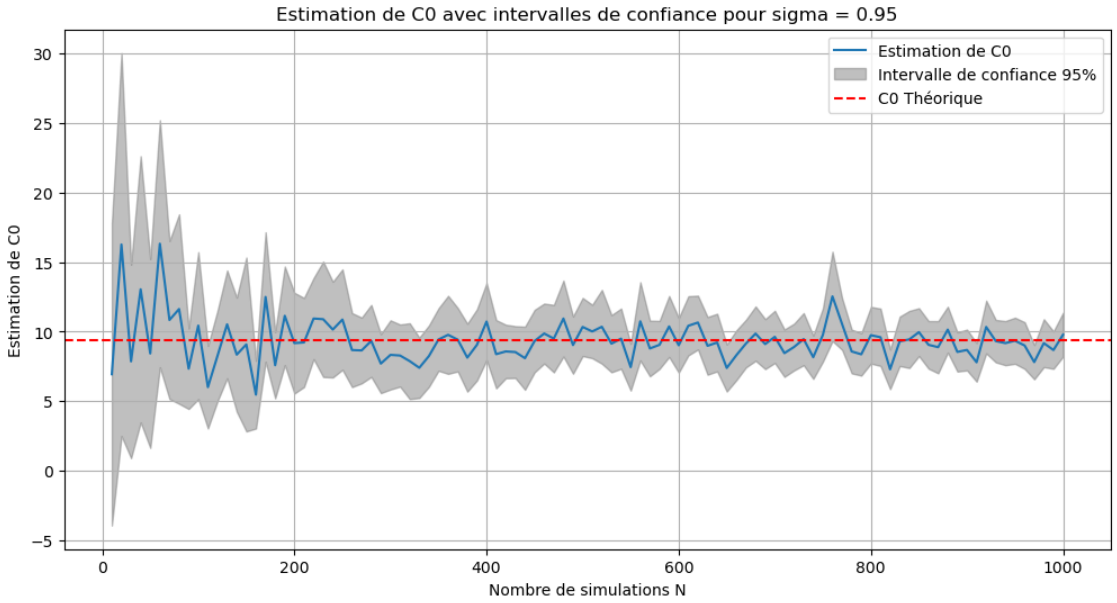
```
Pour sigma = 0.1 on a :
C0 = 1.395
Pour N = 1000 on a : Erreur = 0.002 et Variance empirique = 3.229
Pour N = 10000 on a : Erreur = 0.033 et Variance empirique = 3.259

Pour sigma = 0.95 on a :
C0 = 9.369
Pour N = 1000 on a : Erreur = 0.56 et Variance empirique = 688.43
Pour N = 10000 on a : Erreur = 0.036 et Variance empirique = 964.463
```

On remarque que la variance empirique modifiée augmente quand σ et N augmentent.

2.1.3 Question 3

[ ]:
```
sigma = 0.95
N_max = 1000
step = 10  # le pas de N
alpha = 0.05

def intervalle_confiance(C0, s, N, alpha=0.05):
    t_quantile = t.ppf(1 - alpha/2, df=N-1)
    marge_erreur = t_quantile * s / np.sqrt(N)
    return C0 - marge_erreur, C0 + marge_erreur

Y_N = []
intervalle_inf = []
intervalle_sup = []

for N in range(10, N_max + 1, step):
    X = np.array([simuler_loi_normale() for _ in range(N)])
    C0 = np.array([g(x, sigma) for x in X])
    M = np.mean(C0)
    Y_N.append(M)

    Var_emp = variance_empirique(C0, N)
    s_N = math.sqrt(Var_emp)
```

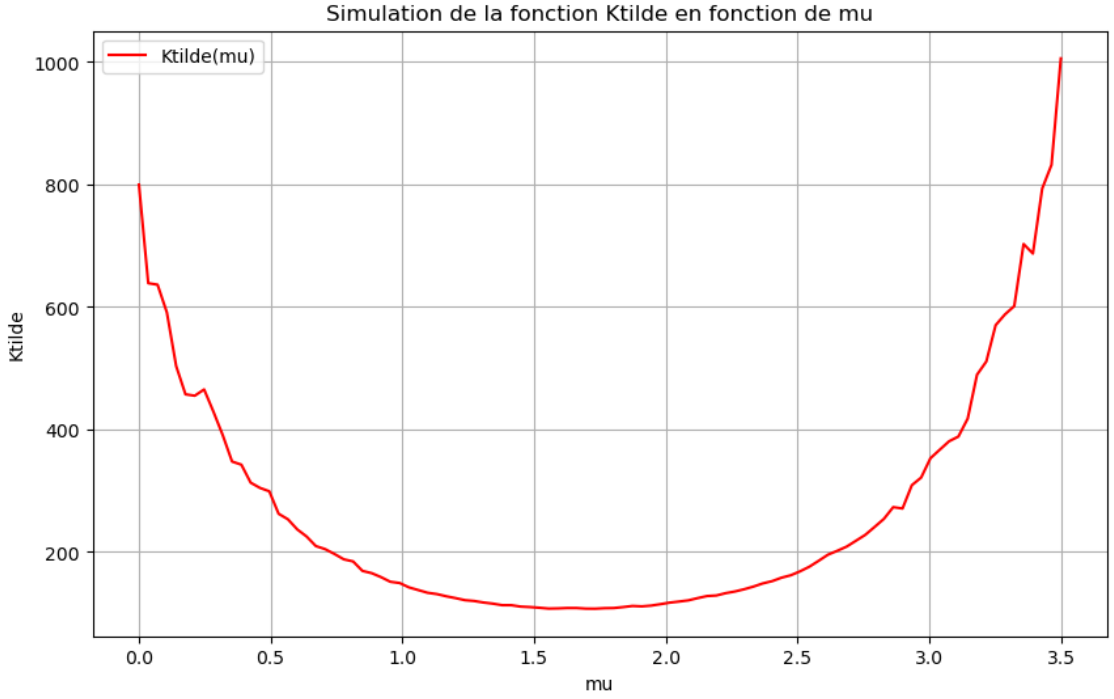
```
    inf, sup = intervalle_confiance(M, s_N, N, alpha)
    intervalle_inf.append(inf)
    intervalle_sup.append(sup)

C0_theorique = fonction_C0(sigma)

# Affichage des résultats
plt.figure(figsize=(12, 6))
plt.plot(range(10, N_max + 1, step), Y_N, label='Estimation de C0')
plt.fill_between(range(10, N_max + 1, step), intervalle_inf, intervalle_sup,␣
 ↪color='grey', alpha=0.5,
                 label='Intervalle de confiance 95%')
plt.axhline(y=C0_theorique, color='r', linestyle='--', label='C0 Théorique')
plt.xlabel('Nombre de simulations N')
plt.ylabel('Estimation de C0')
plt.title(f'Estimation de C0 avec intervalles de confiance pour sigma = {sigma}')
# plt.ylim(4, 14)
plt.legend()
plt.grid(True)
plt.show()
```



2.1.4 Question 4. c)

[ ]:
```
sigma = 0.95
N = 10000

def fonction_K(mu, x, sigma):
```

```
    return (g(mu + x, sigma))**2 * math.exp(-mu*(2*x+mu))

def fonction_Ktilde(mu, sigma, N):
    X = [simuler_loi_normale() for _ in range(N)]
    K = [fonction_K(mu, X[k], sigma) for k in range(N)]
    return np.mean(K)

mus = np.linspace(0, 3.5, 100)

Ktilde = [fonction_Ktilde(mu, sigma, N) for mu in mus]

plt.figure(figsize=(10, 6))
plt.plot(mus, Ktilde, label='Ktilde(mu)', color='red')
plt.title("Simulation de la fonction Ktilde en fonction de mu")
plt.xlabel("mu")
plt.ylabel("Ktilde")
plt.legend()
plt.grid(True)
plt.show()
```



Graphiquement, on a l'impression que la fonction $\mu \longmapsto \widetilde{K}(\mu)$ atteint son minimum au point $\mu^* \approx 1.6$.

2.1.5 Question 4. d)

[ ]:
```
def g_xi(mu, xi, sigma):
    return g(mu + xi, sigma)

def g_xi_prime(mu, xi, sigma):
    return x0 * sigma * np.sqrt(T) * np.exp(-sigma**2 * T / 2 + sigma * np.
 ↪sqrt(T) * (xi + mu))

def l_xi(mu, xi):
    return math.exp(-mu * (2 * xi + mu))

def fonction_derivee_K(mu, xi, sigma):
    gxi = g_xi(mu, xi, sigma)
    gxip = g_xi_prime(mu, xi, sigma)
    lx = l_xi(mu, xi)
    return 2 * lx * gxi * (gxip - (xi + mu) * gxi)

def fonction_derivee_K_2(mu, xi, sigma):
    gxi = g_xi(mu, xi, sigma)
    gxip = g_xi_prime(mu, xi, sigma)
    lx = l_xi(mu, xi)
    term1 = gxi**2 * (2 * (xi + mu)**2 - 1)
    term2 = gxip * (gxip + (sigma * np.sqrt(T) - 4 * (xi + mu)) * gxi)
    return 2 * lx * (term1 + term2)

def algo_Newton_Raphson(mu0, sigma, N, epsilon, n_max):
    Xi = [simuler_loi_normale() for _ in range(N)]
    i = 0
    mu1 = mu0
    while i < n_max :
        Ktilde_derivee1 = np.mean([fonction_derivee_K(mu1, Xik, sigma) for Xik␣
 ↪in Xi])
        Ktilde_derivee2 = np.mean([fonction_derivee_K_2(mu1, Xik, sigma) for Xik␣
 ↪in Xi])
        mu2 = mu1 - (1/Ktilde_derivee2) * Ktilde_derivee1
        if abs(mu2-mu1) < epsilon:
            break
        else:
            mu1 = mu2
        i+=1
    return mu2

sigma = 0.95
mu0 = 2     # On commence l'algorithme à mu0 = 2 par choix arbitraire. N'importe␣
 ↪quel choix de mu0 dans A = [0, 3.3] fonctionnera.
N = 10000
```

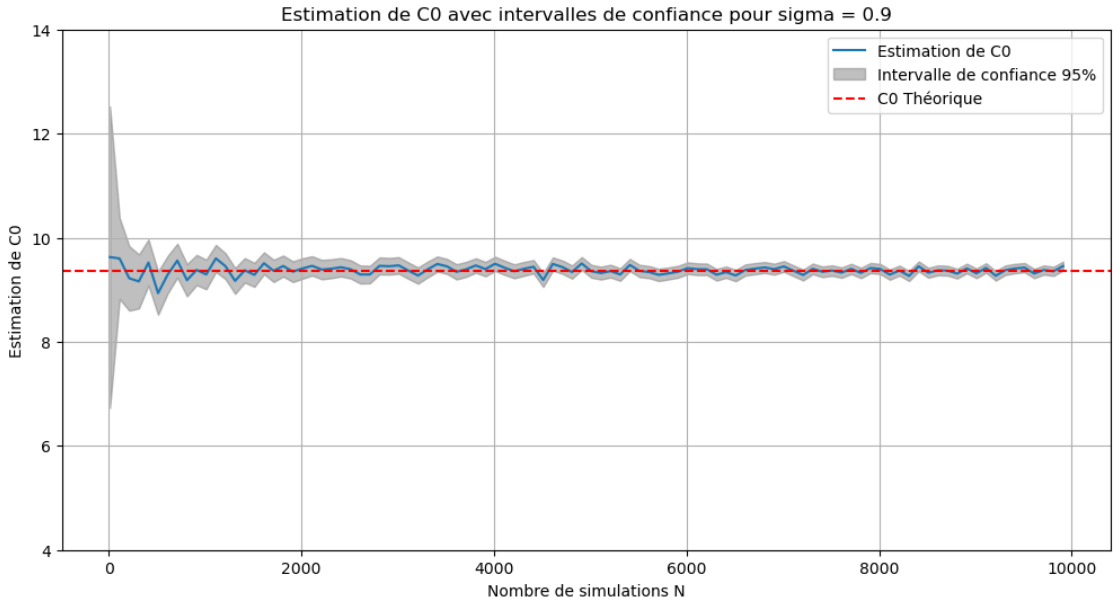
```
epsilon = 10**(-6)      # Critère d'arrêt arbitraire
n_max = 500             # Critère d'itérations max arbitraire

mu_optimal = algo_Newton_Raphson(mu0, sigma, N, epsilon, n_max)
print("mu optimal =", mu_optimal)
```

```
mu optimal = 1.6628626382796252
```

2.1.6 Question 4. e)

[ ]:
```
sigma = 0.95
N_max = 10000
step = 100
alpha = 0.05
mu_opti = mu_optimal    # Souvent vers 1.65

def psi(x, mu, sigma):
    return g(x, sigma) * math.exp(-(1/2)*mu*(2*x - mu))

def intervalle_confiance(C0, s, N, alpha=0.05):
    t_quantile = t.ppf(1 - alpha/2, df=N-1)
    marge_erreur = t_quantile * s / np.sqrt(N)
    return C0 - marge_erreur, C0 + marge_erreur

Z_N = []
intervalle_inf = []
intervalle_sup = []

for N in range(10, N_max + 1, step):
    X = np.array([simuler_loi_normale() for _ in range(N)])
    Z = mu_opti + X
    C0 = np.array([psi(z, mu_opti, sigma) for z in Z])
    M = np.mean(C0)
    Z_N.append(M)

    Var_emp = variance_empirique(C0, N)
    s_N = math.sqrt(Var_emp)

    inf, sup = intervalle_confiance(M, s_N, N, alpha)
    intervalle_inf.append(inf)
    intervalle_sup.append(sup)

C0_theorique = fonction_C0(sigma)

plt.figure(figsize=(12, 6))
plt.plot(range(10, N_max + 1, step), Z_N, label='Estimation de C0')
```

```
plt.fill_between(range(10, N_max + 1, step), intervalle_inf, intervalle_sup,␣
↪color='grey', alpha=0.5,
                 label='Intervalle de confiance 95%')
plt.axhline(y=C0_theorique, color='r', linestyle='--', label='C0 Théorique')
plt.xlabel('Nombre de simulations N')
plt.ylabel('Estimation de C0')
plt.title('Estimation de C0 avec intervalles de confiance pour sigma = 0.9')
plt.ylim(4, 14)
plt.legend()
plt.grid(True)
plt.show()
```

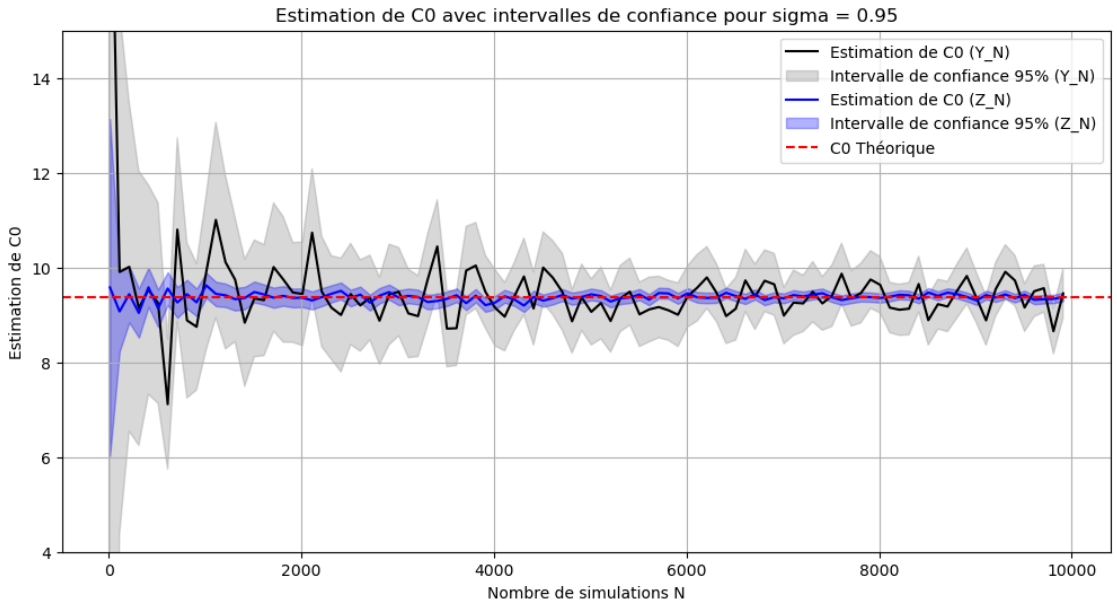


On peut alors comparer la simulation avec $\left(\psi\left(X_1^{\mu^*}\right), \ldots, \psi\left(X_N^{\mu^*}\right)\right)$ avec la simulation naïve de $(g(X_1), \ldots, g(X_N))$:

[ ]:
```
sigma = 0.95
N_max = 10000
step = 100  # Le pas de N
alpha = 0.05
mu_opti = mu_optimal

Y_N = []
intervalle_inf_Y = []
intervalle_sup_Y = []
Z_N = []
intervalle_inf_Z = []
intervalle_sup_Z = []
```

```
for N in range(10, N_max + 1, step):
    X = np.array([simuler_loi_normale() for _ in range(N)])

    # Simulation pour Y_N
    C0_Y = np.array([g(x, sigma) for x in X])
    M_Y = np.mean(C0_Y)
    Y_N.append(M_Y)
    Var_emp_Y = variance_empirique(C0_Y, N)
    s_N_Y = math.sqrt(Var_emp_Y)
    inf_Y, sup_Y = intervalle_confiance(M_Y, s_N_Y, N, alpha)
    intervalle_inf_Y.append(inf_Y)
    intervalle_sup_Y.append(sup_Y)

    # Simulation pour Z_N
    Z = mu_opti + X
    C0_Z = np.array([psi(z, mu_opti, sigma) for z in Z])
    M_Z = np.mean(C0_Z)
    Z_N.append(M_Z)
    Var_emp_Z = variance_empirique(C0_Z, N)
    s_N_Z = math.sqrt(Var_emp_Z)
    inf_Z, sup_Z = intervalle_confiance(M_Z, s_N_Z, N, alpha)
    intervalle_inf_Z.append(inf_Z)
    intervalle_sup_Z.append(sup_Z)

C0_theorique = fonction_C0(sigma)

plt.figure(figsize=(12, 6))
# Y_N plot
plt.plot(range(10, N_max + 1, step), Y_N, label='Estimation de C0 (Y_N)',␣
 ↪color='black')
plt.fill_between(range(10, N_max + 1, step), intervalle_inf_Y, intervalle_sup_Y,␣
 ↪color='grey', alpha=0.3,
                 label='Intervalle de confiance 95% (Y_N)')
# Z_N plot
plt.plot(range(10, N_max + 1, step), Z_N, label='Estimation de C0 (Z_N)',␣
 ↪color='blue')
plt.fill_between(range(10, N_max + 1, step), intervalle_inf_Z, intervalle_sup_Z,␣
 ↪color='blue', alpha=0.3,
                 label='Intervalle de confiance 95% (Z_N)')
plt.axhline(y=C0_theorique, color='r', linestyle='--', label='C0 Théorique')
plt.xlabel('Nombre de simulations N')
plt.ylabel('Estimation de C0')
plt.title(f'Estimation de C0 avec intervalles de confiance pour sigma = {sigma}')
plt.ylim(4, 15)
plt.legend()
plt.grid(True)
```

```
plt.show()
```



Nous avons pris un μ qui minimisait $\mu \longmapsto \widetilde{K}(\mu)$. Ainsi, cela nous a permis d'approximer le mimimum de $\mathbb{E}\left[\psi^2(X^\mu)\right] = \mathbb{E}\left[K(\mu, \xi)\right]$ et donc de réduire drastiquement la variance de notre estimateur !

Effectivement, on remarque bien que l'estimation de C_0 avec Z_N est bien meilleure que l'estimation avec Y_N puisque sa variance est bien plus faible que celle de Y_N.

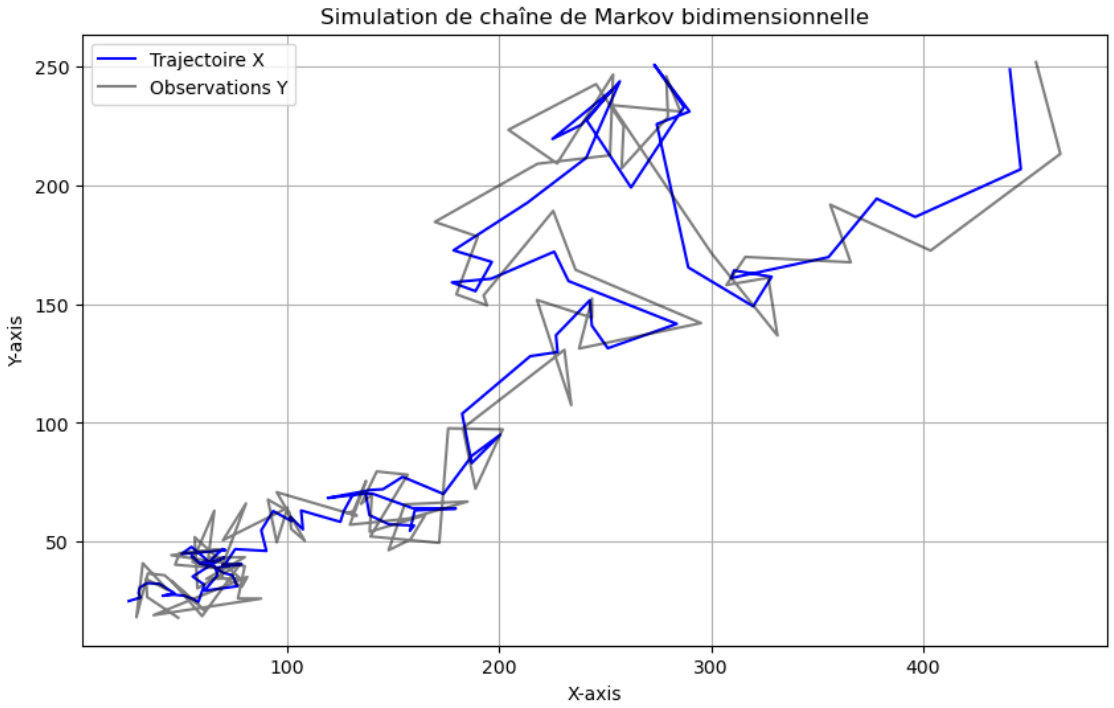
Pour conclure, notre travail de réduction de la variance a été très utile ici, puisque Z_N est un excellent estimateur de C_O pour des valeurs de N très petites, tandis que Y_N fournit des résultats encore assez approximatifs pour des grandes valeurs de N ($N > 10000$).

2.2 Exercice 3

2.2.1 Question 1

[ ]:
```
def simuler_chaine_Markov(n, nu, mu, delta, X0):
    X = np.zeros((n+1,2))
    Y = np.zeros((n+1,2))
    X[0] = X0
    for k in range(n):
        Zk = simuler_gaussienne_bidimensionnelle()
        Vk = simuler_gaussienne_bidimensionnelle()
        X[k+1] = X[k]*(1+mu) + nu * np.array([X[k,0] * Zk[0],X[k,1] * Zk[1]])
        Y[k+1] = X[k+1] + delta * np.array(Vk)      # Car Y[k] est définit pour␣
↪k >= 1
    return (X,Y)
```

```
nu = 0.1
mu = 0.03
X0 = [25, 25]
delta = 9
N = 100


X,Y = simuler_chaine_Markov(N, nu, mu, delta, X0)

plt.figure(figsize=(10, 6))

plt.plot(X[:, 0], X[:, 1], label='Trajectoire X', color='blue')
plt.plot(Y[1:, 0], Y[1:, 1], label='Observations Y', color='black', alpha = 0.5)
plt.title('Simulation de chaîne de Markov bidimensionnelle')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.show()
```



2.2.2 Question 3

Premièrement, on sait que : $\forall k \in \mathbb{N}^*$, $Y_k = X_k + V_k$ avec $V_k \sim \mathcal{N}\left(0, \delta^2 I_2\right)$.

Donc :

$$(Y_k \,|\, X_k = x_k) \sim \mathcal{N}(x_k, \delta^2\, I_2)$$

On en déduit alors la variable aléatoire $(Y_k \,|\, X_k = x_k)$ admet pour densité la fonction suivante :

$$y_k \longmapsto \frac{1}{\delta}\, \varphi_0 \left(\frac{\|y_k - x_k\|}{\delta} \right)$$

De plus, on rappelle cette hypothèse dans le modèle des chaînes de Markov cachées :

$$\forall k \in \mathbb{N}^*, \quad p\,(y_k \,|\, x_k, y_{0:k-1}) = p\,(y_n \,|\, x_k) \tag{1}$$

Ainsi, $\forall n \in \mathbb{N}^*$, on a :

$$\begin{aligned} p\,(y_{1:n} \,|\, x_{1:n}) &= p\,(y_n \,|\, x_{1:n}, y_{1:n-1})\; p\,(y_{1:n-1} \,|\, x_{1:n}) \\ &= p\,(y_n \,|\, x_{1:n})\; p\,(y_{1:n-1} \,|\, x_{1:n}) && \text{(par l'hypothèse (1))} \\ &= p\,(y_n \,|\, x_n)\; p\,(y_{1:n-1} \,|\, x_{1:n-1}) && \text{(car } \forall k \neq n,\, Y_n \perp X_k\text{)} \end{aligned}$$

\end{proof}

Par une relation de récurrence simple, on en déduit que $(Y_{1:k} \,|\, X_{1:k} = x_{1:k})$ admet pour densité la fonction suivante :

$$p\,(\cdot \,|\, x_{1:n}) \;:\; y_{1:n} \longmapsto \delta^{-n} \prod_{k=1}^{n} \varphi_0 \left(\frac{\|y_k - x_k\|}{\delta} \right)$$

Or, comme on part de $X_0 = x_0$ et non de X_1, on peut conclure que :

$$\boxed{p\,(y_{1:n} \,|\, x_{0:n}) = \delta_{x_0}\, \delta^{-n} \prod_{k=1}^{n} \varphi_0 \left(\frac{\|y_k - x_k\|}{\delta} \right)}$$

2.2.3 Question 4

[ ]:
```
n = 10
N = 10000
gamma = 36
X0 = [25, 25]
delta = 9

def distance(x,y):
    return math.sqrt((x[0] - y[0])**2 + (x[1] - y[1])**2)

def simuler_chaine_Markov_upgrade(n, nu, mu, delta, X0):
    X = np.zeros((n+1,2))
    Y = np.zeros((n+1,2))
    X[0] = X0
    proba = 1
```

```
    def loi_normale(x):
        return (1/math.sqrt(2*math.pi))*math.exp(-(x**2)/2)

    for k in range(n):
        Zk = simuler_gaussienne_bidimensionnelle()
        Vk = simuler_gaussienne_bidimensionnelle()
        X[k+1] = X[k]*(1+mu) + nu * np.array([X[k,0] * Zk[0],X[k,1] * Zk[1]])
        Y[k+1] = X[k+1] + delta * np.array(Vk)      # Car Y[k] est définit pour
↪k >= 1
        proba *= (1/delta)*loi_normale(distance(Y[k], X[k])/delta)

    resultats = np.array([np.array(X[1:]),np.array(Y[1:])])
    return resultats, proba

def simuler_N_chaine_markov_upgrade(N, n, nu, mu, delta, X0):
    l_resultats = np.zeros((N,2,n,2))
    l_proba = np.zeros(N)
    for i in range(N):
        res, pr = simuler_chaine_Markov_upgrade(n, nu, mu, delta, X0)
        l_resultats[i] = res
        l_proba[i] = pr
    return l_resultats, l_proba

simulations, l_proba = simuler_N_chaine_markov_upgrade(N, n, nu, mu, delta, X0)
w = l_proba / np.sum(l_proba)

def h(X):
    return np.prod([1 if distance(x, X0) <= gamma else 0 for x in X])

temps_moyen_empirique = np.sum([h(simulations[i, 0]) * w[i] for i in range(N)])
print("P(tau > n | y_{0:n}) =", np.round(temps_moyen_empirique,3))
```

```
P(tau > n | y_{0:n}) = 0.946
```