TP2

10 Mars 2024

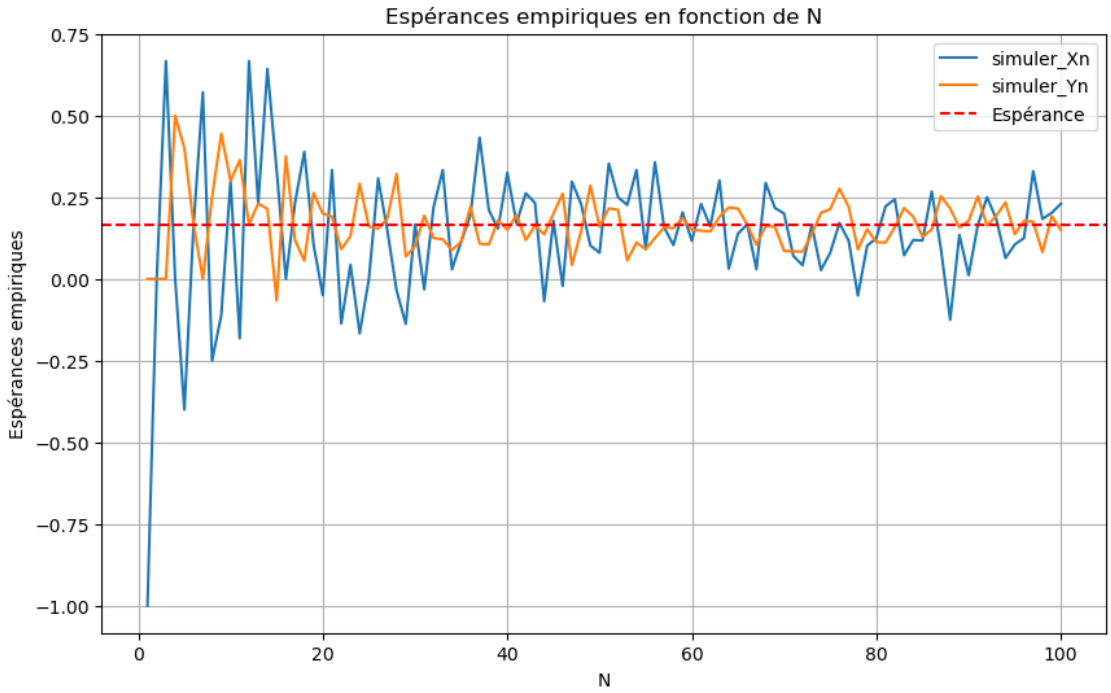
[2]:
```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy.stats import norm,binom,gaussian_kde,t
from mpl_toolkits.mplot3d import Axes3D

def rand():
    return random.random()

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
```

0.1 Exercice 1

0.1.1 Question 3

[3]:
```
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
```

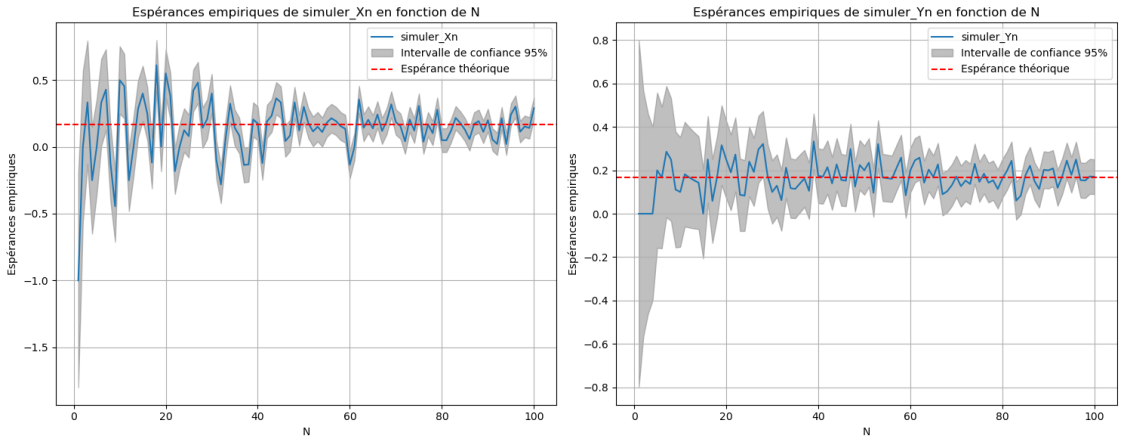
```python
def simuler_Xn(n):
    v = [-1,0,1]
    p = [1/3,1/6,1/2]
    X = tirage_va_Discrete(v,p,n)
    return np.sum(X)/n

def simuler_Yn(n):
    v = [-1,0,1]
    p = [1/60,4/5,11/60]
    Y = tirage_va_Discrete(v,p,n)
    return np.sum(Y)/n


N_values = np.arange(1, 101)

Xn_values = [simuler_Xn(N) for N in N_values]
Yn_values = [simuler_Yn(N) for N in N_values]

plt.figure(figsize=(10, 6))
plt.plot(N_values, Xn_values, label='simuler_Xn')
plt.plot(N_values, Yn_values, label='simuler_Yn')
plt.axhline(y=1/6, color='r', linestyle='--', label='Espérance')
plt.xlabel('N')
plt.ylabel('Espérances empiriques')
plt.title('Espérances empiriques en fonction de N')
plt.legend()
plt.grid(True)
plt.show()
```



0.1.2 Question 4

[4]:
```
# Génération des valeurs de N
N_values = np.arange(1, 101)

# Calcul des valeurs de simuler_Xn et simuler_Yn pour chaque N
Xn_values = [simuler_Xn(N) for N in N_values]
Yn_values = [simuler_Yn(N) for N in N_values]


# Calcul des intervalles de confiance à 95% pour simuler_Xn
IC1_Xn = [Xn_values[i] - 1.96 * math.sqrt(1/(6*N)) for i, N in␣
 ↪enumerate(N_values)]
IC2_Xn = [Xn_values[i] + 1.96 * math.sqrt(1/(6*N)) for i, N in␣
 ↪enumerate(N_values)]

# Calcul des intervalles de confiance à 95% pour simuler_Yn
IC1_Yn = [Yn_values[i] - 1.96 * math.sqrt(1/(6*N)) for i, N in␣
 ↪enumerate(N_values)]
IC2_Yn = [Yn_values[i] + 1.96 * math.sqrt(1/(6*N)) for i, N in␣
 ↪enumerate(N_values)]

# Création de la figure avec deux sous-graphiques
plt.figure(figsize=(15, 6))
```

```
# Premier sous-graphique (Xn)
plt.subplot(1, 2, 1)
plt.plot(N_values, Xn_values, label='simuler_Xn')
plt.fill_between(N_values, IC1_Xn, IC2_Xn, color='grey', alpha=0.5,␣
 ↪label='Intervalle de confiance 95%')
plt.axhline(y=1/6, color='r', linestyle='--', label='Espérance théorique')
plt.xlabel('N')
plt.ylabel('Espérances empiriques')
plt.title('Espérances empiriques de simuler_Xn en fonction de N')
plt.legend()
plt.grid(True)

# Deuxième sous-graphique (Yn)
plt.subplot(1, 2, 2)
plt.plot(N_values, Yn_values, label='simuler_Yn')
plt.fill_between(N_values, IC1_Yn, IC2_Yn, color='grey', alpha=0.5,␣
 ↪label='Intervalle de confiance 95%')
plt.axhline(y=1/6, color='r', linestyle='--', label='Espérance théorique')
plt.xlabel('N')
plt.ylabel('Espérances empiriques')
plt.title('Espérances empiriques de simuler_Yn en fonction de N')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



[5]:
```
list(enumerate(np.arange(1, 11)))
```

[5]:
```
[(0, 1),
 (1, 2),
 (2, 3),
 (3, 4),
 (4, 5),
 (5, 6),
 (6, 7),
 (7, 8),
 (8, 9),
 (9, 10)]
```

0.2 Exercice 2

0.2.1 Question 2

[5]:
```
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_CO_MC(sigma,N):
    X = [simuler_loi_normale() for _ in range(N)]
    CO = [g(Xk,sigma) for Xk in X]
    return np.mean(CO)

def fonction_CO(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_erreur_absolue(sigma,N,CO):
    CO_exp = simuler_CO_MC(sigma,N)
    return abs(CO_exp - CO)

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

for sigma in [0.03, 0.20, 0.40, 0.90, 0.1, 0.95]:
    print("Pour sigma =", sigma, "on a :")
```

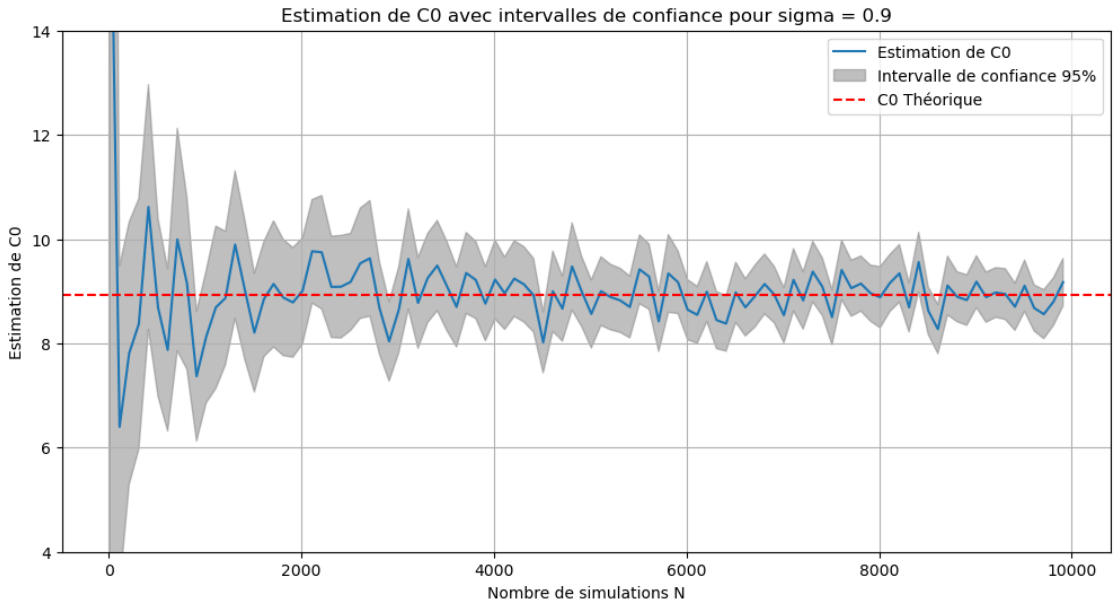
```
    C0 = fonction_C0(sigma)
    print("C0 =", round(C0,3))
    for N in [1000, 10000, 100000]:
        Erreur = calculer_erreur_absolue(sigma,N,C0)
        Var_emp = calculer_variance_empirique(sigma,N)
        print("Pour N =", N, "on a : Erreur =", round(Erreur,3), "et Variance␣
↪empirique =", round(Var_emp,3))
    print("")
```

```
Pour sigma = 0.03 on a :
C0 = 0.8
Pour N = 1000 on a : Erreur = 0.021 et Variance empirique = 0.422
Pour N = 10000 on a : Erreur = 0.014 et Variance empirique = 0.429
Pour N = 100000 on a : Erreur = 0.004 et Variance empirique = 0.427

Pour sigma = 0.2 on a :
C0 = 2.353
Pour N = 1000 on a : Erreur = 0.095 et Variance empirique = 13.039
Pour N = 10000 on a : Erreur = 0.058 et Variance empirique = 12.932
Pour N = 100000 on a : Erreur = 0.018 et Variance empirique = 12.388

Pour sigma = 0.4 on a :
C0 = 4.285
Pour N = 1000 on a : Erreur = 0.13 et Variance empirique = 53.426
Pour N = 10000 on a : Erreur = 0.017 et Variance empirique = 57.609
Pour N = 100000 on a : Erreur = 0.039 et Variance empirique = 58.777

Pour sigma = 0.9 on a :
C0 = 8.928
Pour N = 1000 on a : Erreur = 1.53 et Variance empirique = 390.798
Pour N = 10000 on a : Erreur = 0.052 et Variance empirique = 501.189
Pour N = 100000 on a : Erreur = 0.002 et Variance empirique = 579.424

Pour sigma = 0.1 on a :
C0 = 1.395
Pour N = 1000 on a : Erreur = 0.048 et Variance empirique = 3.306
Pour N = 10000 on a : Erreur = 0.018 et Variance empirique = 3.159
Pour N = 100000 on a : Erreur = 0.002 et Variance empirique = 3.196

Pour sigma = 0.95 on a :
C0 = 9.369
Pour N = 1000 on a : Erreur = 0.904 et Variance empirique = 562.03
Pour N = 10000 on a : Erreur = 0.055 et Variance empirique = 615.456
Pour N = 100000 on a : Erreur = 0.133 et Variance empirique = 687.938
```

0.2.2 Question 3

[7]:
```
sigma = 0.9
N_max = 10000
step = 100  # le pas de N
alpha = 0.05

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)

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('Estimation de C0 avec intervalles de confiance pour sigma = 0.9')
plt.ylim(4, 14)
```

```
plt.legend()
plt.grid(True)
plt.show()
```

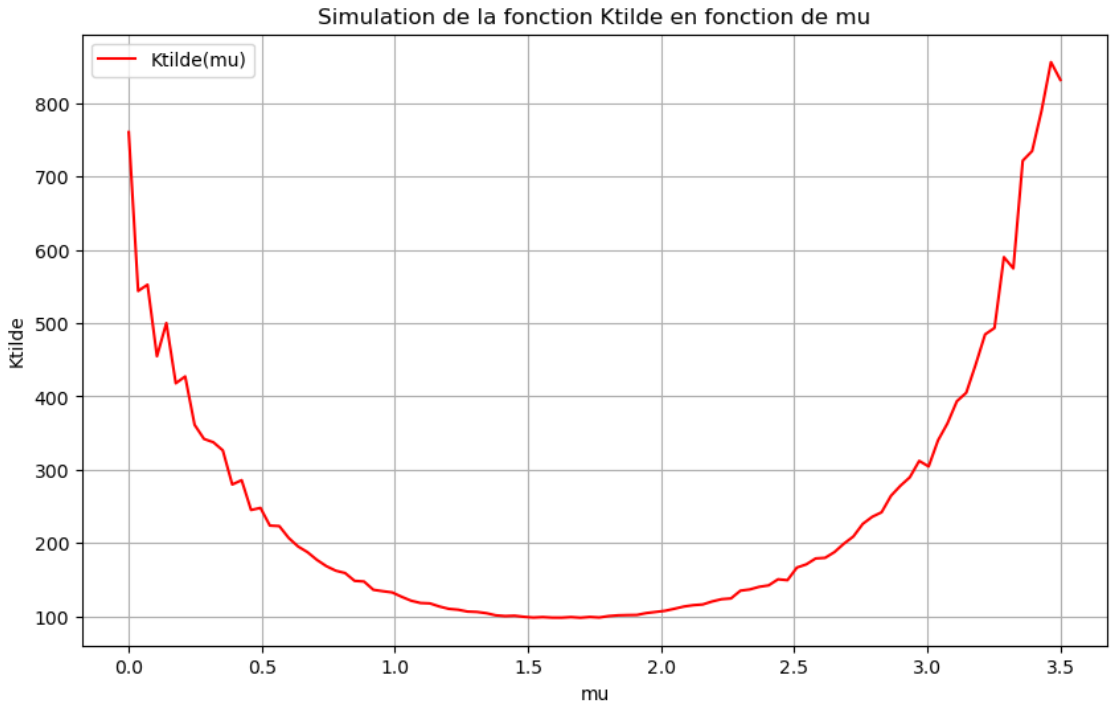


0.2.3 Question 4

(c)

[8]:
```
sigma = 0.9
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()
```



(d)

[9]:
```
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)
```

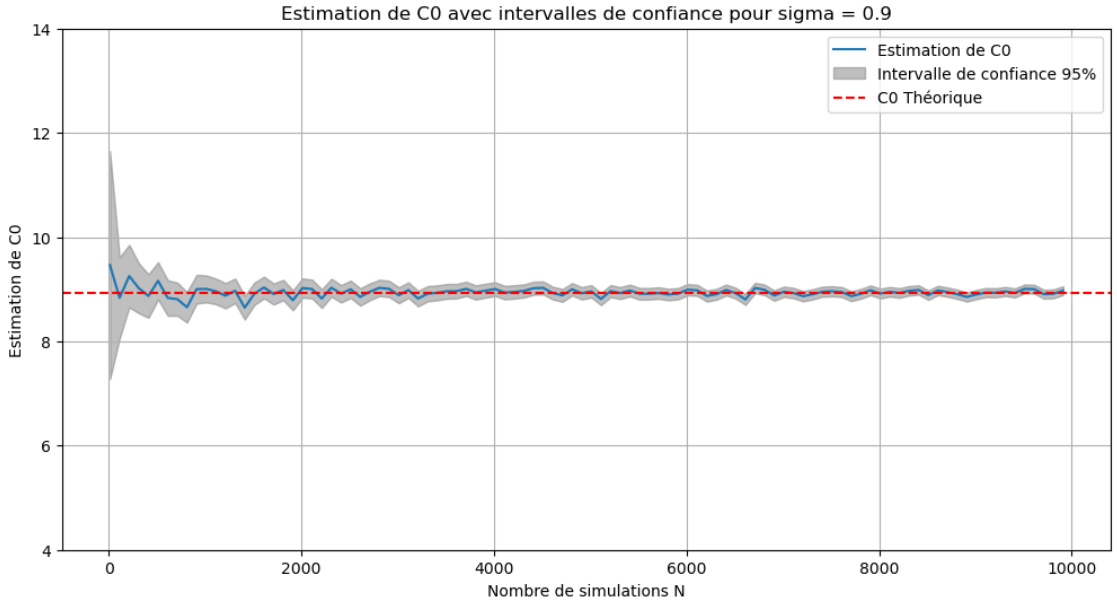
```
    gxip = g_xi_prime(mu, xi, sigma)
    lx = l_xi(mu, xi)
    term1 = gxi**2 * (2 * (xi + mu)**2 - 1)
    term2 = gxip * (gxip + (sigma * np.sqrt(T) - 4 * (xi + mu)) * gxi)
    return 2 * lx * (term1 + term2)

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 = 0.9
mu0 = 2
N = 10000
epsilon = 10**(-6)
n_max = 500

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

```
mu optimal = 1.6387634729673144
```

(e)

[10]:
```
sigma = 0.9
N_max = 10000
step = 100
alpha = 0.05
mu_opti = 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)
```

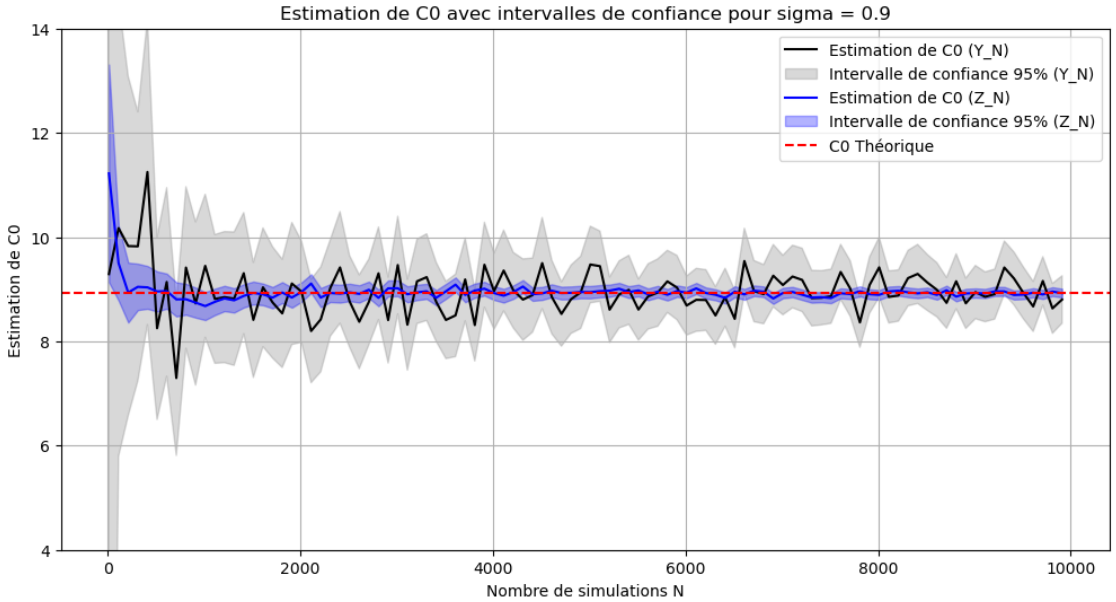
```
    return C0 - marge_erreur, C0 + marge_erreur

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()
```



Les deux simulations ensemble

[11]:
```
sigma = 0.9
N_max = 10000
step = 100  # Le pas de N
alpha = 0.05
mu_opti = 1.65

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('Estimation de C0 avec intervalles de confiance pour sigma = 0.9')
plt.ylim(4, 14)
plt.legend()
plt.grid(True)
plt.show()
```



0.2.4 Question 5

[12]:
```
N = 1000000
alpha = 0.05

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])
Var_emp_Y = variance_empirique(C0_Y, N)
s_N_Y = math.sqrt(Var_emp_Y)
print("Var(g(X)) =", Var_emp_Y)

# Simulation pour Z_N
Z = mu_opti + X
C0_Z = np.array([psi(z, mu_opti, sigma) for z in Z])
Var_emp_Z = variance_empirique(C0_Z, N)
s_N_Z = math.sqrt(Var_emp_Z)
print("Var(psi(Z)) =", Var_emp_Z)

def trouver_N0(s):
    marge_erreur = 1
    N = 1000
    while marge_erreur > 10**(-2):
        t_quantile = t.ppf(1 - alpha/2, df=N-1)
        marge_erreur = t_quantile * s / np.sqrt(N)
        N+=1000
```

```
    return N

N0_Y = trouver_N0(s_N_Y)
print("Pour mu = 0, on a : N_0 =", N0_Y)
N0_Z = trouver_N0(s_N_Z)
print("Pour mu = mu*, on a : N_0 =", N0_Z)
```

```
Var(g(X)) = 581.0780882060931
Var(psi(Z)) = 18.32515475372756
Pour mu = 0, on a : N_0 = 22323000
Pour mu = mu*, on a : N_0 = 705000
```

0.3 Exercice 5

0.3.1 Question 1

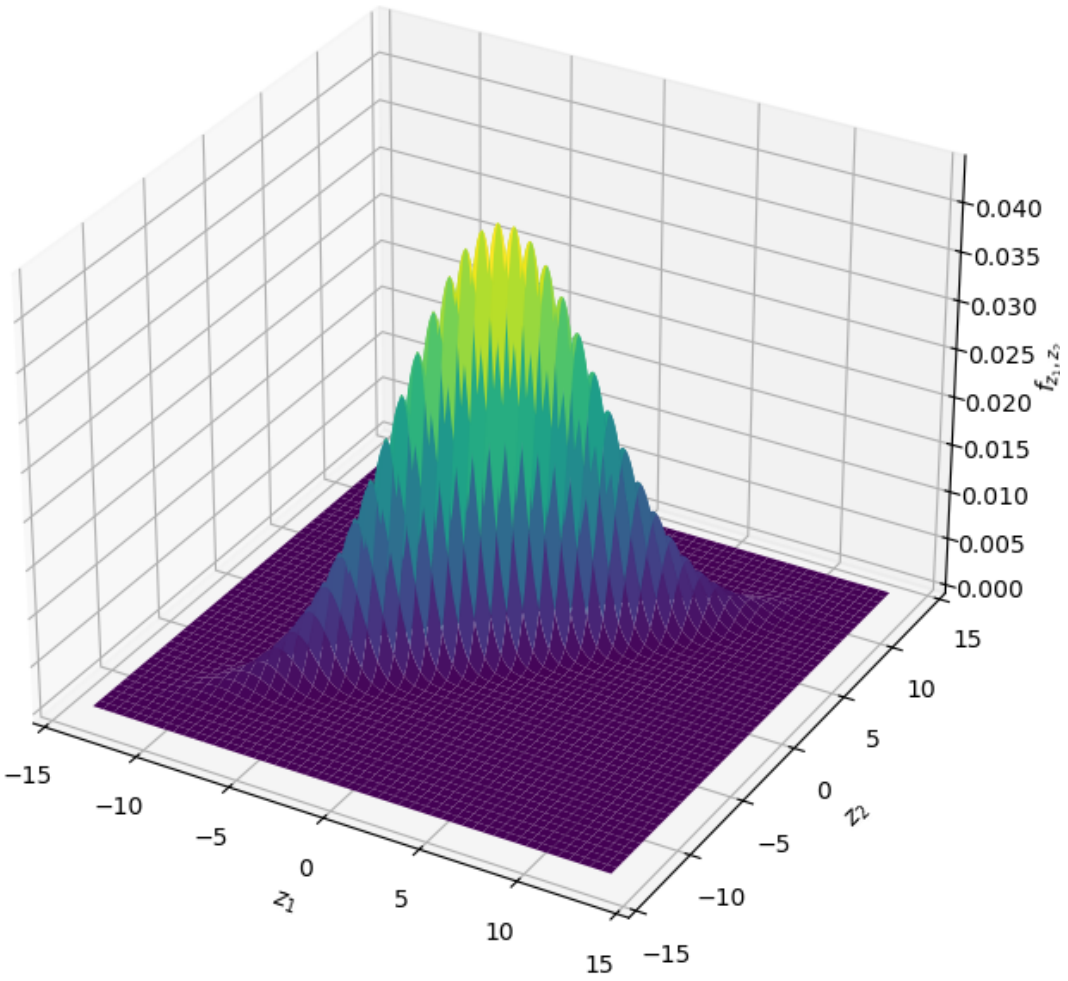
[20]:
```
def choix_des_theta(rho,theta2):
    theta1=(1+math.sqrt(1-rho**2))*theta2/rho
    return theta1,theta2

theta1, theta2 = choix_des_theta(-0.95, 1)
print(theta1,theta2)

theta1, theta2 = choix_des_theta(0.95, 1)
print(theta1,theta2)
```

```
-1.3813156841262315 1
1.3813156841262315 1
```

0.3.2 Question 2

[42]:
```
rho = 0.95
theta2 = 2
nb_points = 1000

s1=choix_des_theta(rho,theta2)
sigma11=s1[0]
sigma22=s1[0]
sigma12=s1[1]
sigma21=s1[1]

Sig1=math.sqrt(sigma11**2+sigma12**2)
Sig2=math.sqrt(sigma21**2+sigma22**2)

#rho=(sigma11*sigma21+sigma12*sigma22)/math.sqrt(Sig1*Sig2)

# sous fonction pour la  densité
def f_Z1Z2(x,y):
```

```
    C=1.0/(2*np.pi*Sig1*Sig1*np.sqrt(1.0-rho**2))
    D=-1.0/(2*(1-rho**2))
    A1=(x/Sig1)**2
    A2=(y/Sig2)**2
    A3=2*rho*x*y/(Sig1*Sig2)
    return C*np.exp(D*(A1-A3+A2))

z1_intervalle = np.linspace(-4*Sig1, 4*Sig1, nb_points)
z2_intervalle = np.linspace(-4*Sig2, 4*Sig2, nb_points)

X, Y = np.meshgrid(z1_intervalle, z2_intervalle)
Z = f_Z1Z2(X, Y)


plt.figure(figsize=(11,8))
ax = plt.axes(projection='3d')

ax.plot_surface(X, Y, Z, cmap='viridis') #cmap=cm.coolwarm

ax.set_title("Densité pour $\\rho=%s$ \n $\\sigma_{11}=%.2f , \\sigma_{12}=%.2f,␣
 ↪\\sigma_{21}=%.2f, \\sigma_{22}=%.2f$"%(rho,sigma11,sigma12,sigma21,sigma22),
            fontsize=13)

ax.set_xlabel(r"$z_{1}$")
ax.set_ylabel(r"$z_{2}$")
ax.set_zlabel(r"$f_{z_{1},z_{2}}$")

plt.show()
```

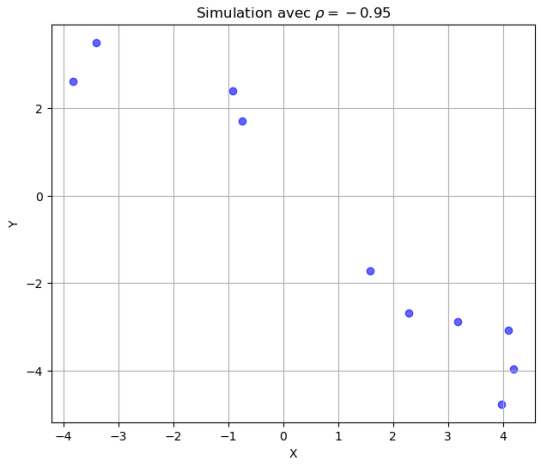


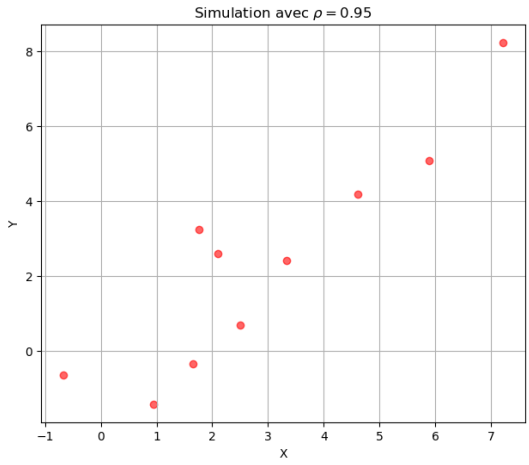
0.3.3 Question 3

[114]:
```
N = 100
theta2 = 2

def simuler_Z(rho, theta2):
    X1,X2 = simuler_gaussienne_bidimensionnelle()
    s1=choix_des_theta(rho,theta2)
    sigma11=s1[0]
    sigma22=s1[0]
    sigma12=s1[1]
```

```
    sigma21=s1[1]
    Z1 = sigma11*X1 + sigma12*X2
    Z2 = sigma21*X1 + sigma22*X2
    return [Z1,Z2]

def simuler_N_Z(rho, theta2, n):
    return [simuler_Z(rho, theta2) for _ in range(n)]

def affichage_simulation_Z(rho, theta2, N):
    # Simulation pour rho = -0.95 et rho = 0.95
    Z_neg = simuler_N_Z(-rho, theta2, N)
    Z_pos = simuler_N_Z(rho, theta2, N)

    # Extraction des coordonnées
    z1_neg, z2_neg = zip(*Z_neg)
    z1_pos, z2_pos = zip(*Z_pos)

    plt.figure(figsize=(16, 6))

    # Pour rho = -0.95
    plt.subplot(1, 2, 1)
    plt.scatter(z1_neg, z2_neg, alpha=0.6, color='blue')
    plt.title('Simulation avec $\\rho = -%s$'%rho)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)

    # Pour rho = 0.95
    plt.subplot(1, 2, 2)
    plt.scatter(z1_pos, z2_pos, alpha=0.6, color='red')
    plt.title('Simulation avec $\\rho = %s$'%rho)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
```

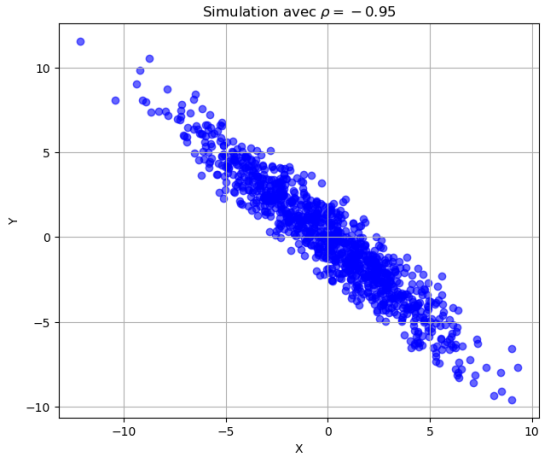
[115]:
```
affichage_simulation_Z(0.95, 2, 10)
```

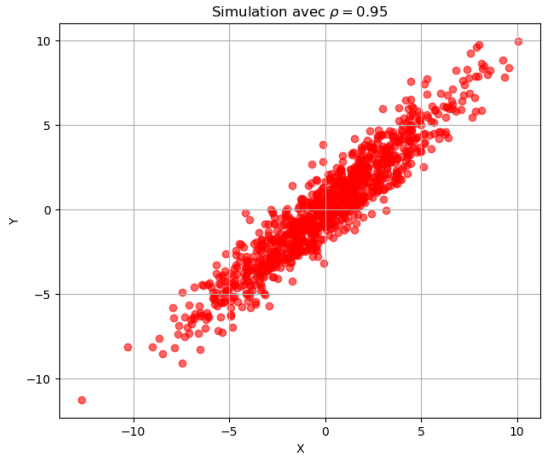




0.3.4 Question 4

[72]:
```
affichage_simulation_Z(0.95, 2, 1000)
```





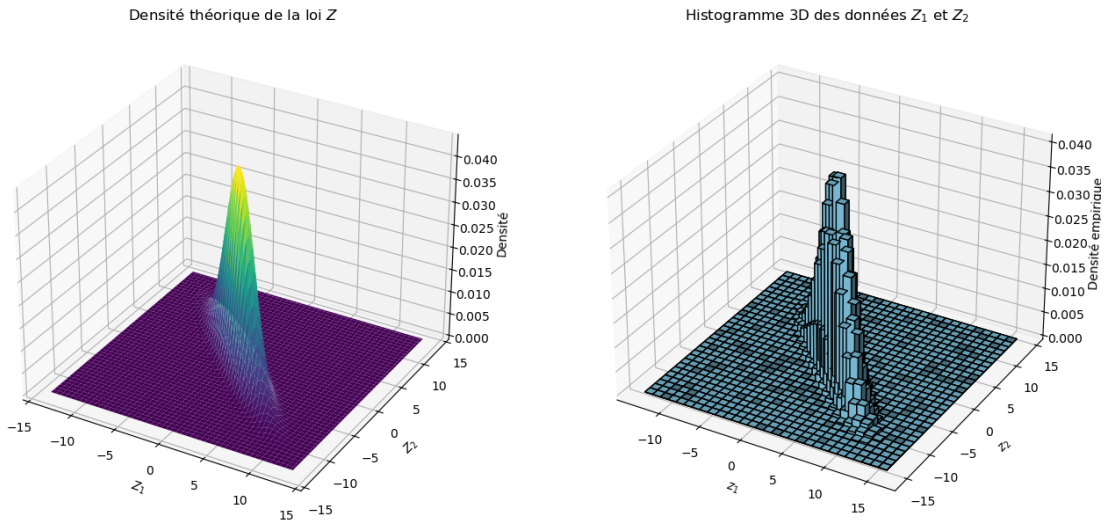
0.3.5 Question 5

En 3D :

[116]:
```
rho = -0.95
theta2 = 2
N = 100000

s1=choix_des_theta(rho,theta2)
sigma11=s1[0]
sigma22=s1[0]
```

```
sigma12=s1[1]
sigma21=s1[1]

Sig1=math.sqrt(sigma11**2+sigma12**2)
Sig2=math.sqrt(sigma21**2+sigma22**2)


# Densité théorique de la loi Z
x_theory = np.linspace(-4*Sig1, 4*Sig1, 100)
y_theory = np.linspace(-4*Sig2, 4*Sig2, 100)
x_theory, y_theory = np.meshgrid(x_theory, y_theory)
z_theory = f_Z1Z2(x_theory,y_theory)

data = simuler_N_Z(rho, theta2, N)
data_np = np.array(data)

# Calcul de l'histogramme 2D avec numpy pour obtenir les hauteurs et les bords␣
 ↪des bacs
hist, xedges, yedges = np.histogram2d(data_np[:,0], data_np[:,1], bins=30,␣
 ↪density=True)

# Préparation des positions X, Y pour les barres
xpos, ypos = np.meshgrid(xedges[:-1] + np.diff(xedges)/2, yedges[:-1] + np.
 ↪diff(yedges)/2)
xpos = xpos.ravel()
ypos = ypos.ravel()
zpos = 0

# Dimensions des barres
dx = dy = np.ones_like(zpos) * (xedges[1] - xedges[0])
dz = hist.ravel()

# Configuration de la figure
fig = plt.figure(figsize=(16, 8))

# Affichage de la densité théorique
ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(x_theory, y_theory, z_theory, cmap='viridis')
ax1.set_title('Densité théorique de la loi $Z$')
ax1.set_xlabel(r"$Z_{1}$")
ax1.set_ylabel(r"$Z_{2}$")
ax1.set_zlabel('Densité')

# Histogramme 3D des données X et Y
ax2 = fig.add_subplot(122, projection='3d')
ax2.bar3d(xpos, ypos, zpos, dx, dy, dz, color='skyblue', edgecolor='black',␣
 ↪zsort='average')
```

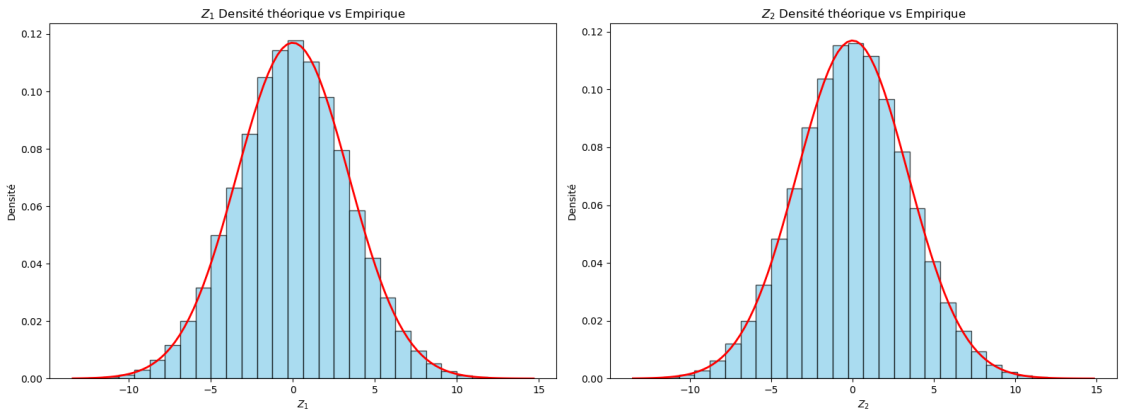
```
ax2.set_title('Histogramme 3D des données $Z_{1}$ et $Z_{2}$')
ax2.set_xlabel(r"$z_{1}$")
ax2.set_ylabel(r"$z_{2}$")
ax2.set_zlabel('Densité empirique')


plt.show()
```



Et en 2D :

[117]:
```
rho = 0.95
theta2 = 2
N = 100000

s1=choix_des_theta(rho,theta2)
sigma11=s1[0]
sigma22=s1[0]
sigma12=s1[1]
sigma21=s1[1]

Sig1=math.sqrt(sigma11**2+sigma12**2)
Sig2=math.sqrt(sigma21**2+sigma22**2)


# Simulation des données
data = simuler_N_Z(rho, theta2, N)
data_np = np.array(data)

# Extraction de Z1 et Z2
```

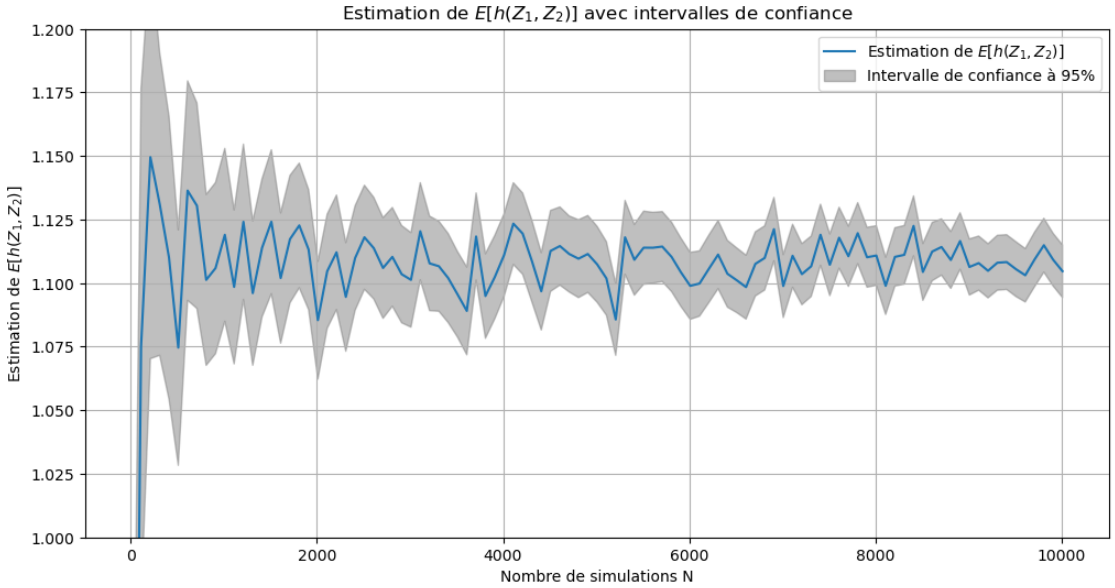
```
z1_data = data_np[:, 0]
z2_data = data_np[:, 1]

# Préparation des plages pour les courbes théoriques basées sur les données␣
 ↪simulées
x_theory_z1 = np.linspace(z1_data.min(), z1_data.max(), 100)
y_theory_z1 = norm.pdf(x_theory_z1, 0, Sig1)

x_theory_z2 = np.linspace(z2_data.min(), z2_data.max(), 100)
y_theory_z2 = norm.pdf(x_theory_z2, 0, Sig2)

# Préparation des figures pour Z1 et Z2
fig = plt.figure(figsize=(16, 6))

# Histogramme et densité théorique pour Z1
ax1 = fig.add_subplot(121)
ax1.hist(z1_data, bins=30, density=True, color='skyblue', edgecolor='black',␣
 ↪alpha=0.7)
ax1.plot(x_theory_z1, y_theory_z1, color='red', linewidth=2)
ax1.set_title('$Z_1$ Densité théorique vs Empirique')
ax1.set_xlabel('$Z_1$')
ax1.set_ylabel('Densité')

# Histogramme et densité théorique pour Z2
ax2 = fig.add_subplot(122)
ax2.hist(z2_data, bins=30, density=True, color='skyblue', edgecolor='black',␣
 ↪alpha=0.7)
ax2.plot(x_theory_z2, y_theory_z2, color='red', linewidth=2)
ax2.set_title('$Z_2$ Densité théorique vs Empirique')
ax2.set_xlabel('$Z_2$')
ax2.set_ylabel('Densité')

plt.tight_layout()
plt.show()
```



0.3.6 Question 6

[133]:
```
rho = 0.95
theta2 = 0.135
N_max = 10000 + 100
step = 100  # le pas de N
alpha = 0.05

def h(z1, z2):
    return np.exp(z1 + z2)

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)

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

# Initialisation des listes pour les résultats
Y_N = []
intervalle_inf = []
intervalle_sup = []

# Simulation et calcul pour différents N
for N in range(10, N_max + 1, step):
    Z = simuler_N_Z(rho, theta2, N)
    Z_np = np.array(Z)
    h_values = h(Z_np[:, 0], Z_np[:, 1])

    M = np.mean(h_values)
    Y_N.append(M)

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

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

# Affichage des résultats
plt.figure(figsize=(12, 6))
```

```
plt.plot(range(10, N_max + 1, step), Y_N, label='Estimation de $E[h(Z_1,Z_2)]$')
plt.fill_between(range(10, N_max + 1, step), intervalle_inf, intervalle_sup,␣
↪color='grey', alpha=0.5,
                 label='Intervalle de confiance à 95%')
plt.xlabel('Nombre de simulations N')
plt.ylabel('Estimation de $E[h(Z_1,Z_2)]$')
plt.title('Estimation de $E[h(Z_1,Z_2)]$ avec intervalles de confiance')
plt.legend()
plt.grid(True)
plt.ylim(1.0,1.2)
plt.show()
```



[134]:
```
N = 10000
n = 100
theta2 = 0.135

Vec_M = []

for _ in range(n):
    Z = simuler_N_Z(rho, theta2, N)
    Z_np = np.array(Z)
    h_values = h(Z_np[:, 0], Z_np[:, 1])
    M = np.mean(h_values)
    Vec_M.append(M)

M_final = np.mean(Vec_M)
```

```
Var_emp = variance_empirique(Vec_M, n)
s_N = math.sqrt(Var_emp)

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

Dic_Result={}

Dic_Result["rho"]=rho
Dic_Result["taille_echantillon"]=int(N)
Dic_Result["nombre_echantillons"]=int(n)
Dic_Result["valeur_moyenne"]=np.around(M_final, decimals=4)
Dic_Result["Borne_inf"]=np.around(inf, decimals=4)
Dic_Result["Borne_sup"]=np.around(sup, decimals=4)

print(Dic_Result)
```

```
{'rho': 0.95, 'taille_echantillon': 10000, 'nombre_echantillons': 100,
'valeur_moyenne': 1.1086, 'Borne_inf': 1.1085, 'Borne_sup': 1.1087}
```