TP1

25 Février 2024

[1]:
```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from scipy.stats import norm,binom,gaussian_kde
from mpl_toolkits.mplot3d import Axes3D

def rand():
    return random.random()
```

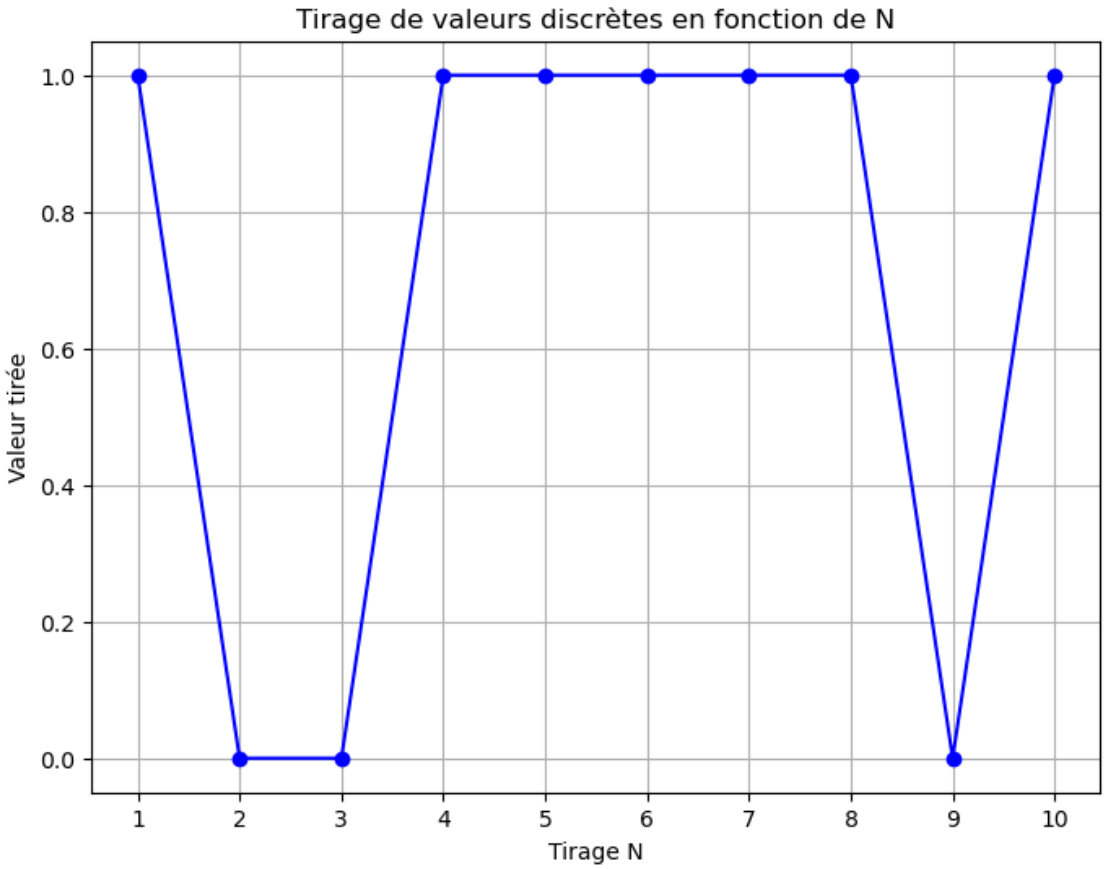
1 Partie 1 : Méthodes d'inversion

1.1 Exercice 1

[ ]:
```
v=[0,1]
pr=[0.2,0.8]
N=10

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

X = tirage_va_Discrete(v,pr,N)

print("Résultats du tirage:")
print(X)
```

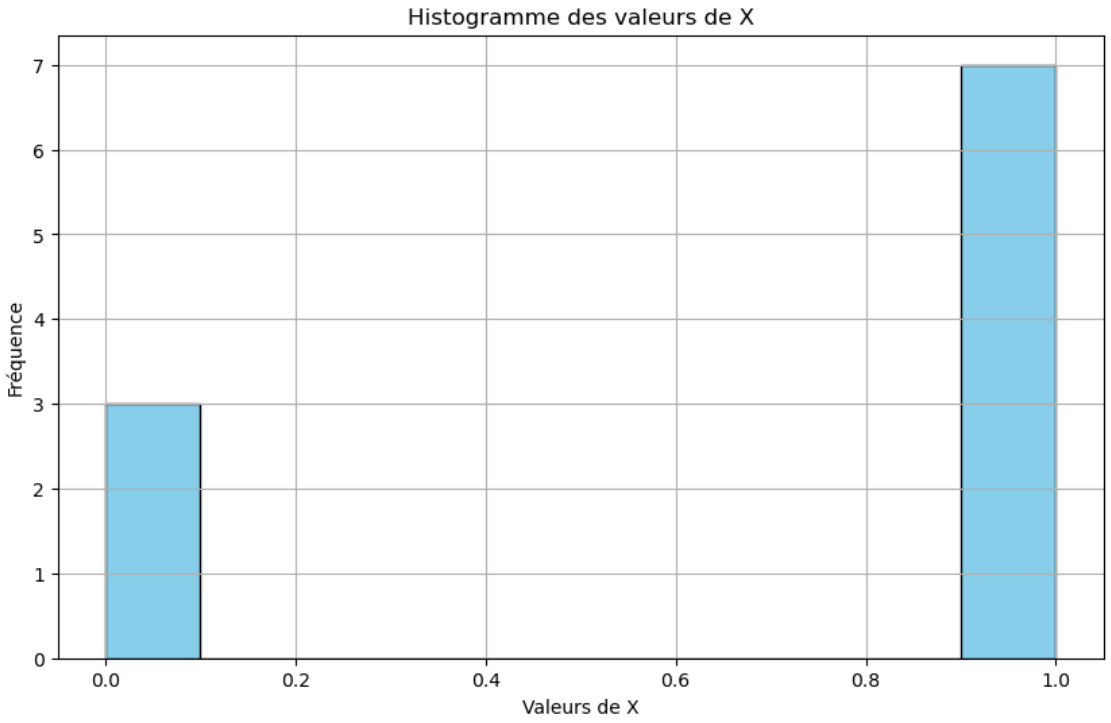
```
Résultats du tirage:
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
```

[3]:
```
X = tirage_va_Discrete(v,pr,N)

# Tracé du graphique
plt.figure(figsize=(8, 6))
plt.plot(range(1, N + 1), X, marker='o', linestyle='-', color='b')
plt.title('Tirage de valeurs discrètes en fonction de N')
plt.xlabel('Tirage N')
plt.ylabel('Valeur tirée')
plt.xticks(range(1, N + 1))
plt.grid(True)
plt.show()
```



[4]:
```
# Affichage de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(X, density=True, color='skyblue', edgecolor='black')
plt.title('Histogramme des valeurs de X')
plt.xlabel('Valeurs de X')
plt.ylabel('Fréquence')
plt.grid(True)
```

```
plt.show()
```



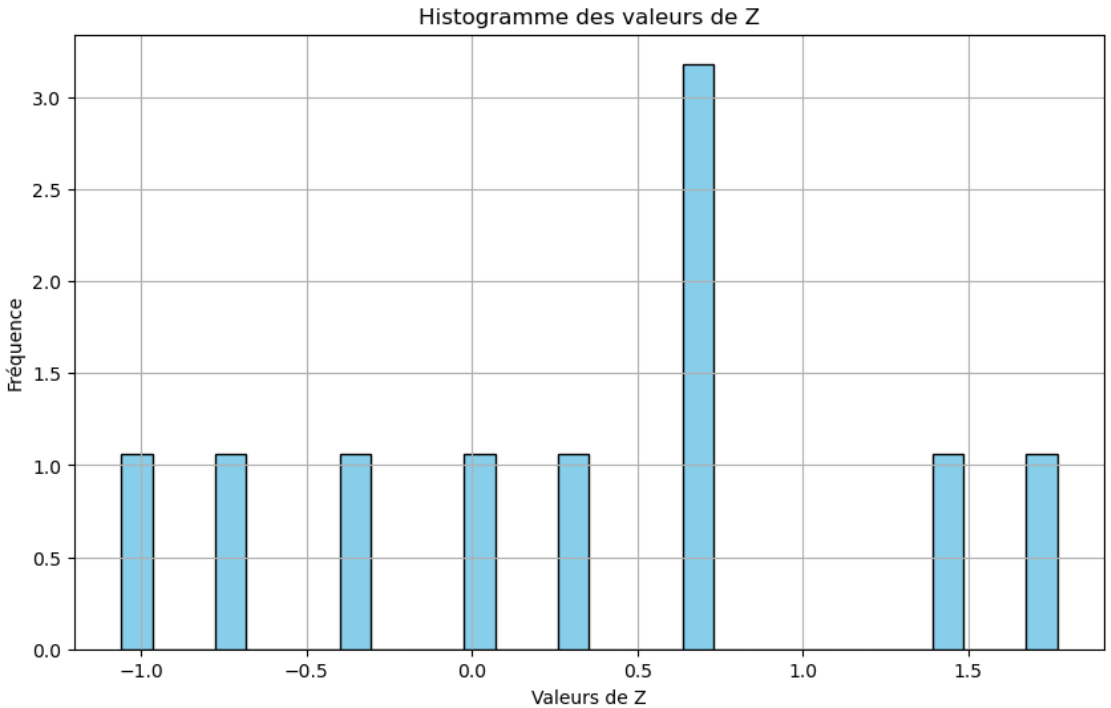
1.1.1 Question 3

[5]:
```
N1=100
N2=1000
N3=10000

X1=tirage_va_Discrete(v,pr,N1)
X2=tirage_va_Discrete(v,pr,N2)
X3=tirage_va_Discrete(v,pr,N3)

p1 = len(X1[X1==1])/N1
p2 = len(X2[X2==1])/N2
p3 = len(X3[X3==1])/N3
print(p1,p2,p3)
```

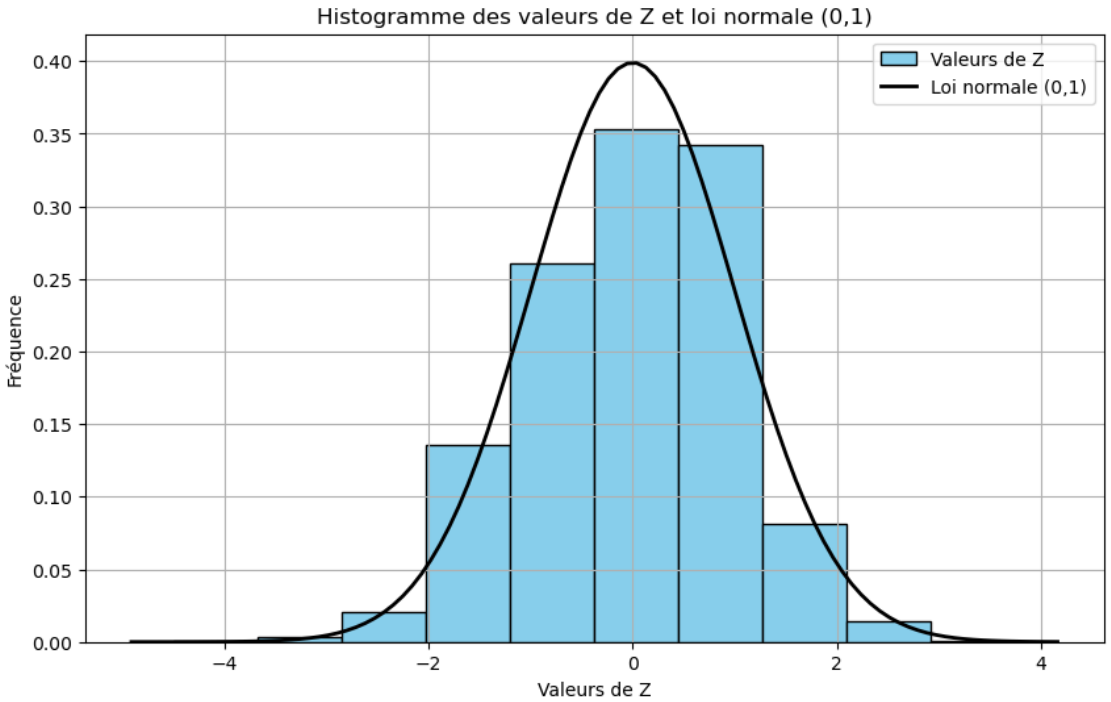
```
0.82 0.809 0.7992
```

1.1.2 Question 4

[6]:
```
n=50 # Au choix entre 10,30,50
N=10

p=0.8

def Z_bernoulli(n,N):
    Z=np.zeros(N)
    for i in range(N):
        X = tirage_va_Discrete(v,pr,n)
        Xnbarre = np.mean(X)
        Z[i] = np.sqrt(n/(p*(1-p)))*(Xnbarre - p)
    return Z

Z = Z_bernoulli(n,N)

# Affichage de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(Z, bins=30, density=True, color='skyblue', edgecolor='black')
plt.title('Histogramme des valeurs de Z')
plt.xlabel('Valeurs de Z')
plt.ylabel('Fréquence')
plt.grid(True)
plt.show()
```



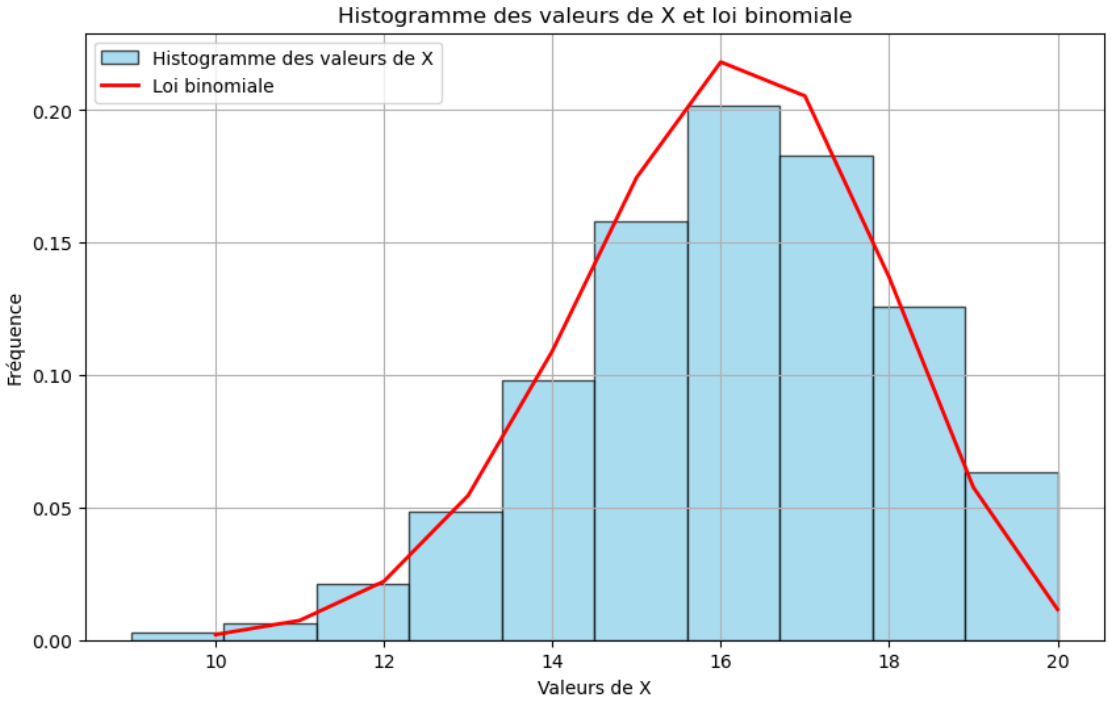
[7]:
```
n=100
N=100000

Z=Z_bernoulli(n,N)

# Vos données Z et l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(Z, bins=10, density=True, color='skyblue', edgecolor='black', alpha=1,␣
↪label='Valeurs de Z')

# Tracer de la loi normale
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
pp = norm.pdf(x, 0, 1)  # Loi normale de moyenne 0 et d'écart-type 1
plt.plot(x, pp, 'k', linewidth=2, label='Loi normale (0,1)')

# Paramètres du graphique
plt.title('Histogramme des valeurs de Z et loi normale (0,1)')
plt.xlabel('Valeurs de Z')
plt.ylabel('Fréquence')
plt.grid(True)
plt.legend()
plt.show()
```



1.2 Exercice 2

1.2.1 Question 2

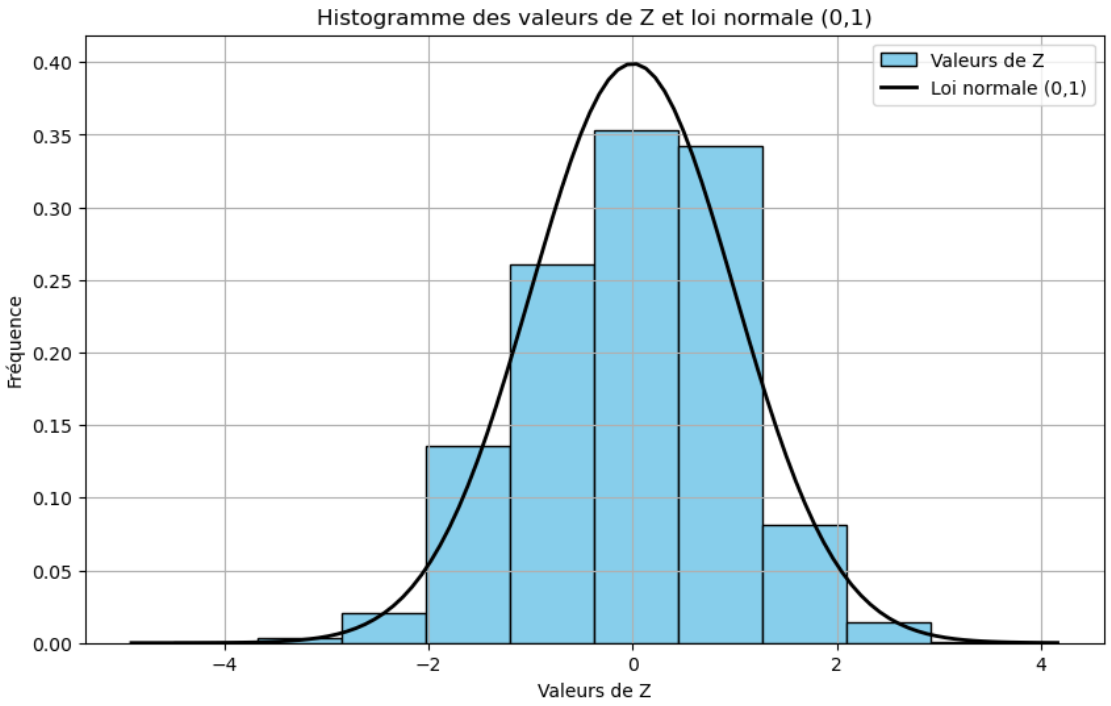
[8]:
```
n_bin=20
N=10000

def tirage_Binomiale(n_bin):
    return np.sum(tirage_va_Discrete(v,pr,n_bin))

def echantillon_Binomiale(n_bin,N):
    return np.array([tirage_Binomiale(n_bin) for _ in range(N)])

X = echantillon_Binomiale(n_bin,N)

# Affichage de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(X, density=True, color='skyblue', edgecolor='black', alpha=0.7,␣
 ↪label='Histogramme des valeurs de X')

# Tracé de la loi binomiale
x = np.arange(binom.ppf(0.001, n_bin, p), binom.ppf(0.999, n_bin, p)+1)
plt.plot(x, binom.pmf(x, n_bin, p), 'r-', linewidth=2, label='Loi binomiale')

# Paramètres du graphique
plt.title('Histogramme des valeurs de X et loi binomiale')
plt.xlabel('Valeurs de X')
plt.ylabel('Fréquence')
plt.grid(True)
plt.legend()
plt.show()
```



1.2.2 Question 3/4

[9]:
```
n=10
N=100000

def Z_binomiale(n_bin,n,N):
    Z=np.zeros(N)
    for i in range(N):
        X = echantillon_Binomiale(n_bin,n)
        Xnbarre = np.mean(X)
        Z[i] = np.sqrt(n/(n*p*(1-p)))*(Xnbarre - n*p)
    return Z

# Vos données Z et l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(Z, bins=10, density=True, color='skyblue', edgecolor='black', alpha=1,␣
 ↪label='Valeurs de Z')

# Tracer de la loi normale
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
pp = norm.pdf(x, 0, 1)  # Loi normale de moyenne 0 et d'écart-type 1
plt.plot(x, pp, 'k', linewidth=2, label='Loi normale (0,1)')
```

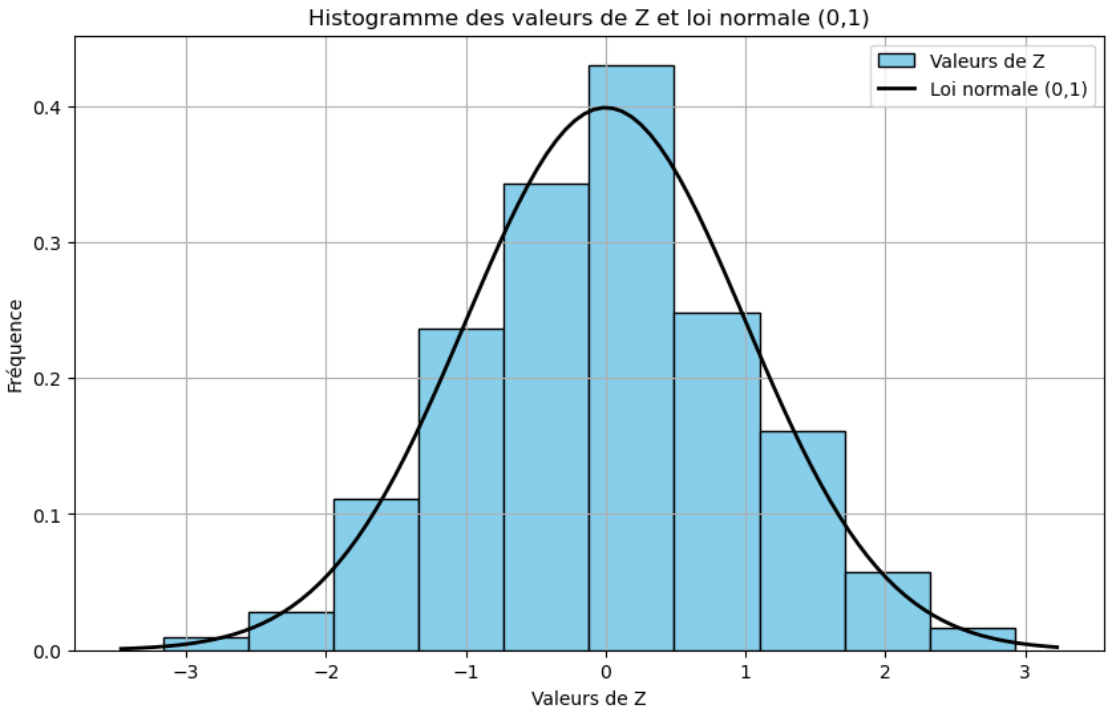
```
# Paramètres du graphique
plt.title('Histogramme des valeurs de Z et loi normale (0,1)')
plt.xlabel('Valeurs de Z')
plt.ylabel('Fréquence')
plt.grid(True)
plt.legend()
plt.show()
```



1.2.3 Question 5

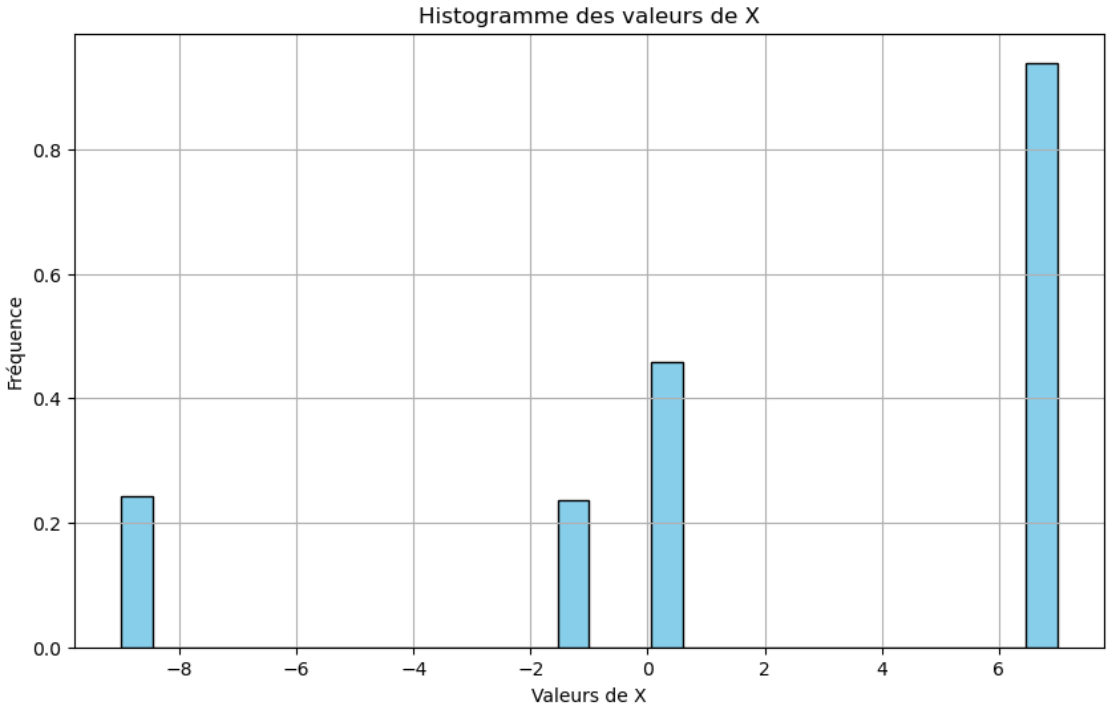
[10]:
```
n_bin=20
n=50
N=1000

def Z_binomiale(n_bin,n,N):
    Z=np.zeros(N)
    for i in range(N):
        X = echantillon_Binomiale(n_bin,n)
        Xnbarre = np.mean(X)
        Z[i] = np.sqrt(n/(n_bin*p*(1-p)))*(Xnbarre - n_bin*p)
    return Z

Z = Z_binomiale(n_bin,n,N)
```

```
# Vos données Z et l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(Z, bins=10, density=True, color='skyblue', edgecolor='black', alpha=1,␣
 ↪label='Valeurs de Z')

# Tracer de la loi normale
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
pp = norm.pdf(x, 0, 1)  # Loi normale de moyenne 0 et d'écart-type 1
plt.plot(x, pp, 'k', linewidth=2, label='Loi normale (0,1)')

# Paramètres du graphique
plt.title('Histogramme des valeurs de Z et loi normale (0,1)')
plt.xlabel('Valeurs de Z')
plt.ylabel('Fréquence')
plt.grid(True)
plt.legend()
plt.show()
```



1.3 Exercice 3

[11]:
```
v = [0.5, -9, -1.5, 7]
pr = [1/4, 1/8, 1/8, 1/2]
N = 10000

X = tirage_va_Discrete(v, pr, N)

# Affichage de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(X, bins=30, density=True, color='skyblue', edgecolor='black')
plt.title('Histogramme des valeurs de X')
plt.xlabel('Valeurs de X')
plt.ylabel('Fréquence')
plt.grid(True)
plt.show()
```



[12]:
```
valeurs_uniques, f = np.unique(X, return_counts=True)

frequences_emp = np.array([f[2],f[0],f[1],f[3]])/N

print("Proba : ", pr)
print("Fréquences empiriques : ", frequences_emp)
```

```
Proba :  [0.25, 0.125, 0.125, 0.5]
```

```
Fréquences empiriques :  [0.2441 0.1291 0.1261 0.5007]
```

1.4 Exercice 4

[13]:
```python
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 constituer_groupes(n,p):
    E = [i+1 for i in range(n)]
    groupes = [i for i in range(p)]
    q = n//p
    r = n%p
    res = [[] for _ in range(p)]
    while len(E) > r:
        i_g = tirage_va_Discrete_Uniforme(groupes)
        x = tirage_va_Discrete_Uniforme(E)
        res[i_g].append(x)
        E.remove(x)
        if len(res[i_g]) == q:
            groupes.remove(i_g)

    res[0] += E
    return res
```

[14]:
```python
p=3
n=11

constituer_groupes(n,p)
```

[14]:
```
[[6, 9, 10, 2, 7], [3, 8, 4], [1, 11, 5]]
```

1.5 Exercice 5

1.5.1 Méthode 1 : Inversion analytique

[15]:
```
def simuler_geometrique1(p):
    u = rand()
    k = math.ceil(math.log(1 - u) / math.log(1 - p))
    return k

def simuler_n_geometrique1(n,p):
    return [simuler_geometrique1(p) for _ in range(n)]

n = 10
p = 0.1
simuler_n_geometrique1(n,p)
```

[15]:
```
[28, 5, 2, 20, 14, 23, 3, 3, 9, 3]
```

1.5.2 Méthode 2 : Spécifique à la loi Géométrique

[16]:
```
def simuler_geometrique2(p):
    n = 1
    while n<100000: # arbitraire
        u = rand()
        if u < p:
            return n
        n += 1

def simuler_n_geometrique2(n,p):
    return [simuler_geometrique2(p) for _ in range(n)]

n = 10
p = 0.1
simuler_n_geometrique2(n,p)
```

[16]:
```
[6, 4, 4, 10, 6, 6, 3, 13, 2, 12]
```

1.5.3 Méthode 3 - Méthode générale du prof

[17]:
```
alpha = 0.999

def simuler_geometrique3(p):
    u = rand()
    N = math.ceil(math.log(1 - alpha) / math.log(1 - p))
    C = np.zeros(N+1) # N+1 car C_0 = 0
    P = np.zeros(N+2)
    C[0],P[1] = 0,p
    for k in range(1,N+1):
```

```
        C[k] = C[k-1] + P[k]
        if (C[k-1]<u and u<=C[k]):  # C[N] = alpha ~ 1 à la fin de la boucle
            return k
        P[k+1] = (1-p)*P[k]
    return N

def simuler_n_geometrique3(n,p):
    return [simuler_geometrique3(p) for _ in range(n)]

n = 10
p = 0.1
simuler_n_geometrique3(n,p)
```

[17]: [5, 7, 3, 9, 2, 16, 9, 5, 2, 1]

1.5.4 Méthode 4 - Méthode de la loi de Poisson (pas du tout naturelle pour la loi Géométrique)

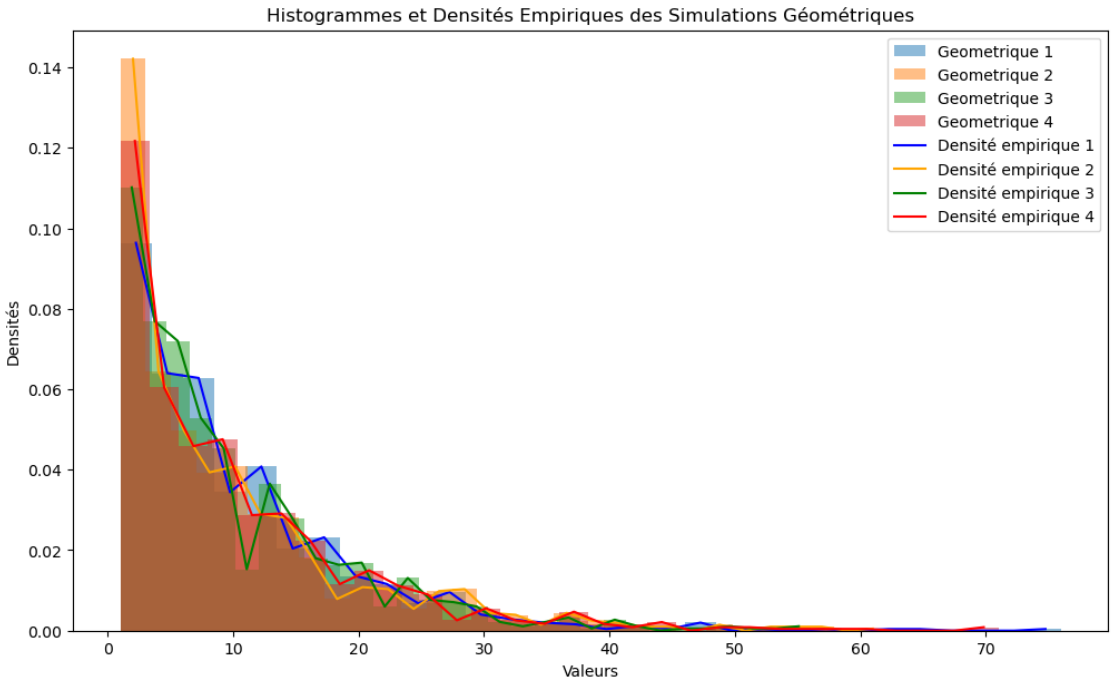
[18]:
```
def simuler_geometrique4(p):
    F = 0
    k = 0
    u = rand()
    while F < u:
        k += 1
        F += (1-p)**(k-1) * p   # C'est la que c'est pas naturelle + pas optimisé
    return k

def simuler_n_geometrique4(n,p):
    return [simuler_geometrique4(p) for _ in range(n)]

n = 10
p = 0.1
simuler_n_geometrique4(n,p)
```

[18]: [4, 13, 2, 4, 19, 6, 4, 7, 21, 8]

1.5.5 Histogramme

[19]:
```
# Paramètres
n = 1000
p = 0.1

# Simulation
data1 = simuler_n_geometrique1(n, p)
data2 = simuler_n_geometrique2(n, p)
data3 = simuler_n_geometrique3(n, p)
```

```
data4 = simuler_n_geometrique4(n, p)

# Tracé des histogrammes
plt.figure(figsize=(12, 7))

plt.hist(data1, bins=30, alpha=0.5, label='Geometrique 1', density=True)
plt.hist(data2, bins=30, alpha=0.5, label='Geometrique 2', density=True)
plt.hist(data3, bins=30, alpha=0.5, label='Geometrique 3', density=True)
plt.hist(data4, bins=30, alpha=0.5, label='Geometrique 4', density=True)

# Ajout des densités empiriques
densities = [data1, data2, data3, data4]
colors = ['blue', 'orange', 'green', 'red']
labels = ['Densité empirique 1', 'Densité empirique 2', 'Densité empirique 3',␣
↪'Densité empirique 4']

for density, color, label in zip(densities, colors, labels):
    count, bins = np.histogram(density, bins=30, density=True)
    center = (bins[:-1] + bins[1:]) / 2
    plt.plot(center, count, color=color, label=label)

plt.legend()
plt.title('Histogrammes et Densités Empiriques des Simulations Géométriques')
plt.xlabel('Valeurs')
plt.ylabel('Densités')
plt.show()
```



2 Partie 2 : Méthodes de rejet et de transformation

2.1 Exercice 6

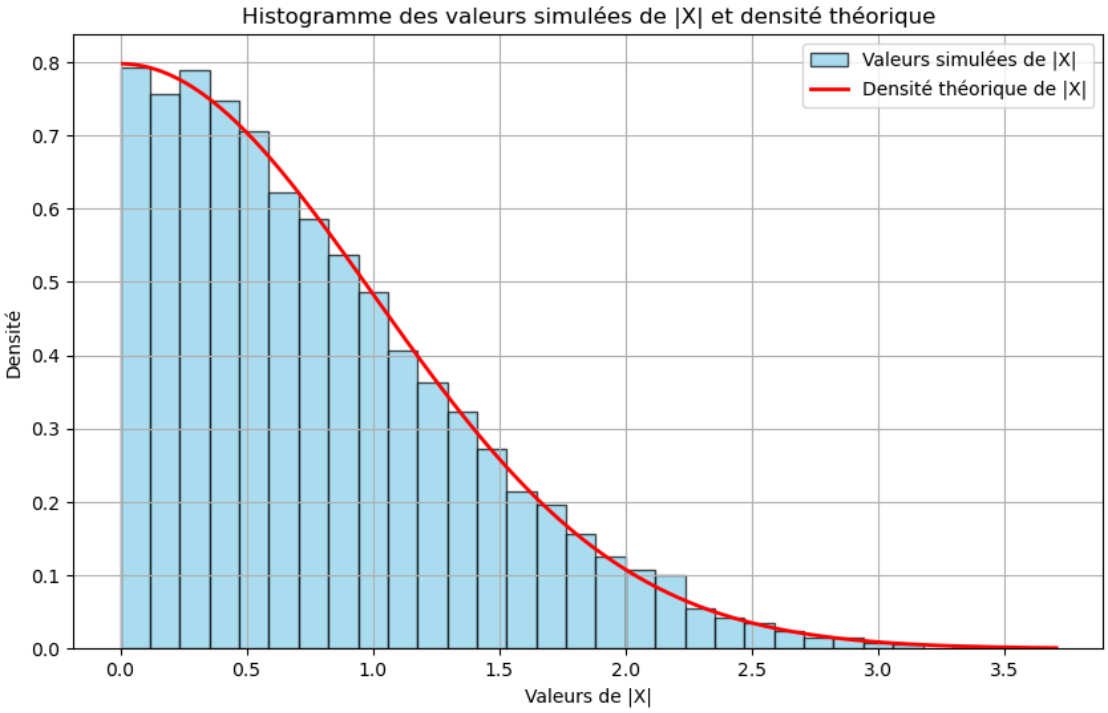
2.1.1 Question 3

[20]:
```
def simulerAbsX():
    c = math.sqrt(2*math.e/math.pi)
    def f(x):
        if x <= 0 : return 0
        else : return (2/math.sqrt(2*math.pi))*np.exp((-1/2)*x**2)
    def g(x):
        if x <= 0 : return 0
        else : return math.exp(-x)

    u0 = rand()
    x = math.log(1/(1-u0))
    u = rand()
    test = c*u*g(x)
    while test > f(x):
        u0 = rand()
        x = math.log(1/(1-u0))
        u = rand()
        test = c*u*g(x)
    return x

def simuler_n_AbsX(n):
    return [simulerAbsX() for _ in range(n)]
```

2.1.2 Question 4

[21]:
```
N = 10000
X = simuler_n_AbsX(N)

# Création de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(X, bins=30, density=True, color='skyblue', edgecolor='black', alpha=0.
 ↪7, label='Valeurs simulées de |X|')

# Tracé direct de la densité f pour |X| en utilisant l'expression mathématique
xmin, xmax = plt.xlim()
x = np.linspace(0.01, xmax, 1000)  # Début à 0.01
pp = (2/math.sqrt(2*math.pi)) * np.exp(-0.5 * x**2)  # Utilisation de␣
 ↪l'expression mathématique de f pour |X|
```

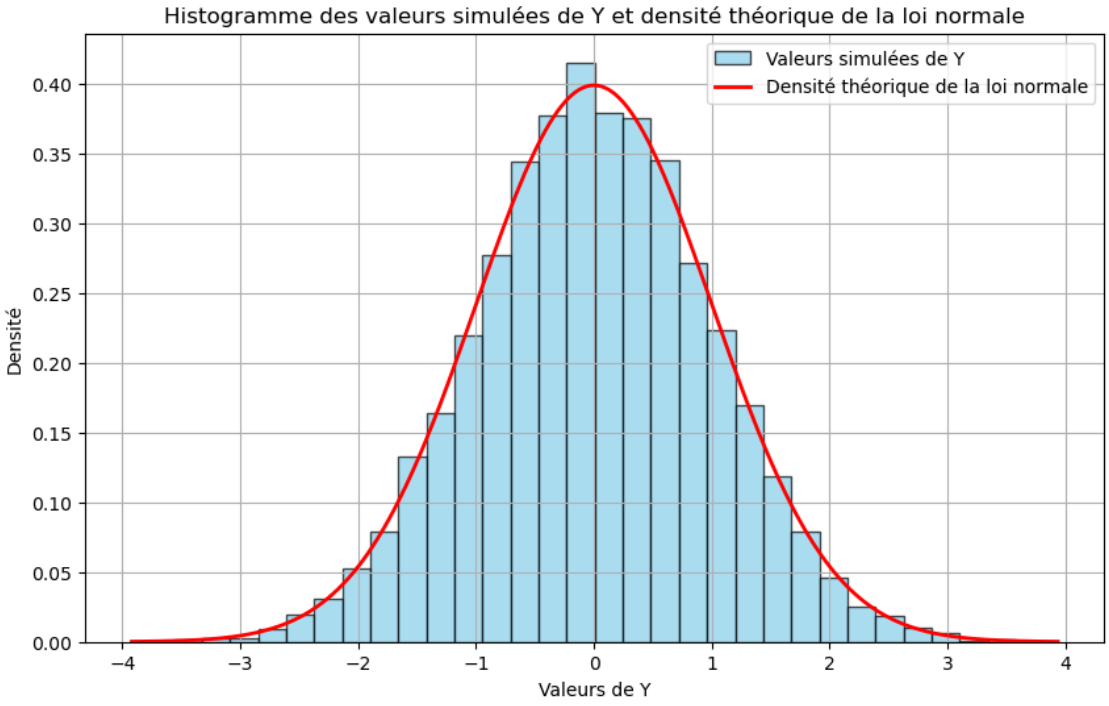
```
plt.plot(x, pp, 'r', linewidth=2, label='Densité théorique de |X|')

# Paramètres du graphique
plt.title('Histogramme des valeurs simulées de |X| et densité théorique')
plt.xlabel('Valeurs de |X|')
plt.ylabel('Densité')
plt.grid(True)
plt.legend()
plt.show()
```



2.1.3 Question 6

[22]:
```
def simuler_loi_normale():
    theta = tirage_va_Discrete_Uniforme([-1,1])
    return theta*simulerAbsX()

def simuler_n_loi_normale(n):
    return [simuler_loi_normale() for _ in range(n)]

N = 10000
X = simuler_n_loi_normale(N)

# Création de l'histogramme
```

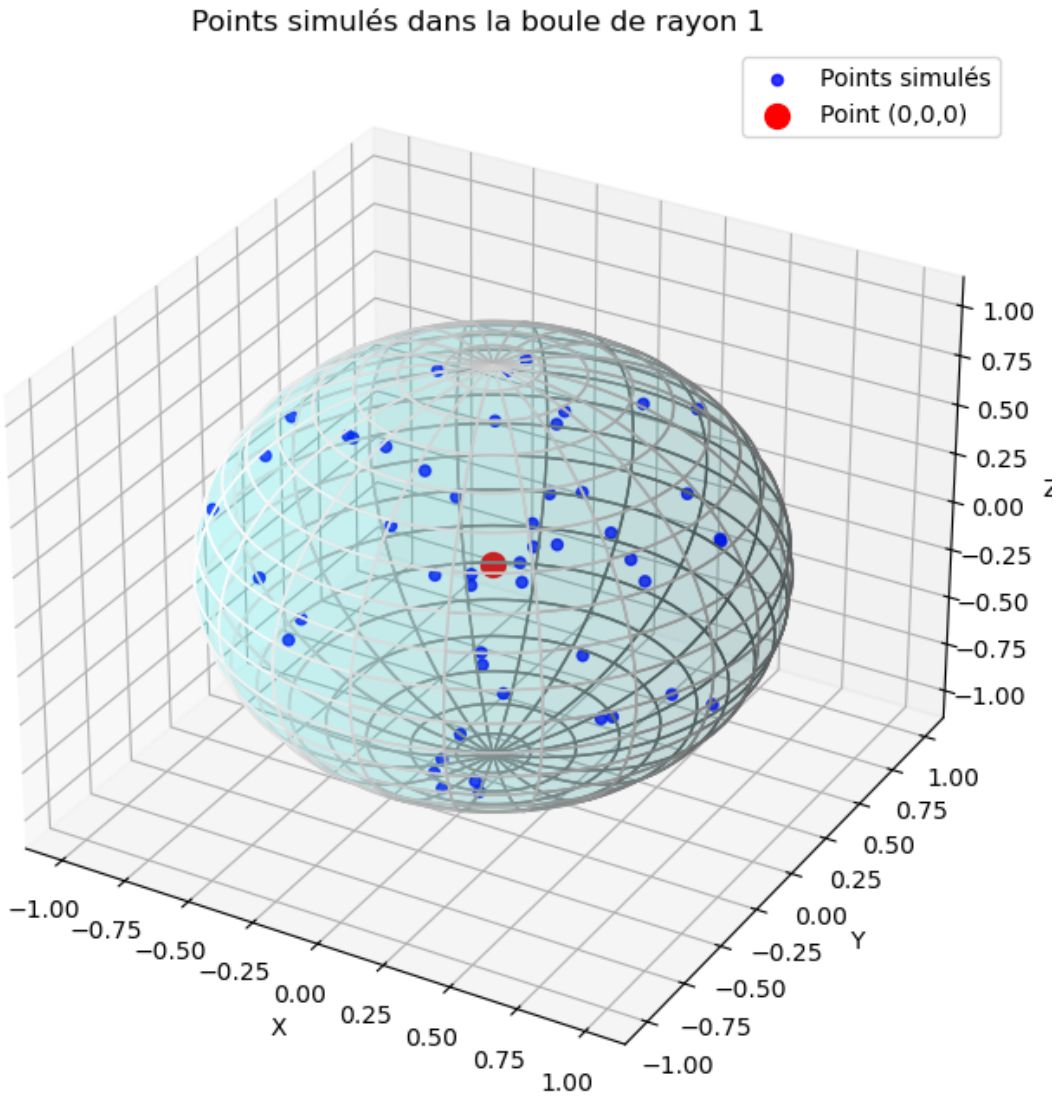
```
plt.figure(figsize=(10, 6))
plt.hist(X, bins=30, density=True, color='skyblue', edgecolor='black', alpha=0.
 ↪7, label='Valeurs simulées de Y')

# Tracé direct de la loi normale centrée réduite
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 1000)
pp = norm.pdf(x, 0, 1)  # Utilise la moyenne 0 et l'écart-type 1
plt.plot(x, pp, 'r', linewidth=2, label='Densité théorique de la loi normale')

# Paramètres du graphique
plt.title('Histogramme des valeurs simulées de Y et densité théorique de la loi␣
 ↪normale')   # Y = \Theta * |X|
plt.xlabel('Valeurs de Y')
plt.ylabel('Densité')
plt.grid(True)
plt.legend()
plt.show()
```



2.2 Exercice 7

[23]:
```
def simuler_point_sphere():
    u1, u2, u3 = 2*rand()-1, 2*rand()-1, 2*rand()-1
    while (u1*u1 + u2*u2 + u3*u3 > 1):
        u1, u2, u3 = 2*rand()-1, 2*rand()-1, 2*rand()-1
    return [u1, u2, u3]

def simuler_n_points_sphere(n):
    return [simuler_point_sphere() for _ in range(n)]

N = 50
X = simuler_n_points_sphere(N)

# Préparation des données pour l'affichage
X_data = np.array(X)
x, y, z = X_data[:,0], X_data[:,1], X_data[:,2]

# Création du graphique 3D pour afficher les points sur la sphère avec la sphère␣
↪elle-même
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Affichage des points simulés
ax.scatter(x, y, z, color='blue', alpha=0.8, label='Points simulés')

# Génération de la sphère en transparent
phi = np.linspace(0, np.pi, 100)
theta = np.linspace(0, 2 * np.pi, 100)
phi, theta = np.meshgrid(phi, theta)

# Coordonnées sphériques (r = 1 pour la sphère unitaire)
x_sphere = np.sin(phi) * np.cos(theta)
y_sphere = np.sin(phi) * np.sin(theta)
z_sphere = np.cos(phi)

# Affichage de la sphère
ax.plot_surface(x_sphere, y_sphere, z_sphere, rstride=5, cstride=5,␣
↪color='cyan', edgecolors='w', alpha=0.1)

# Ajout du point (0,0,0) en rouge
ax.scatter([0], [0], [0], color='red', s=100, label='Point (0,0,0)')

ax.set_title('Points simulés dans la boule de rayon 1')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
```

```
ax.legend()

plt.show()
```

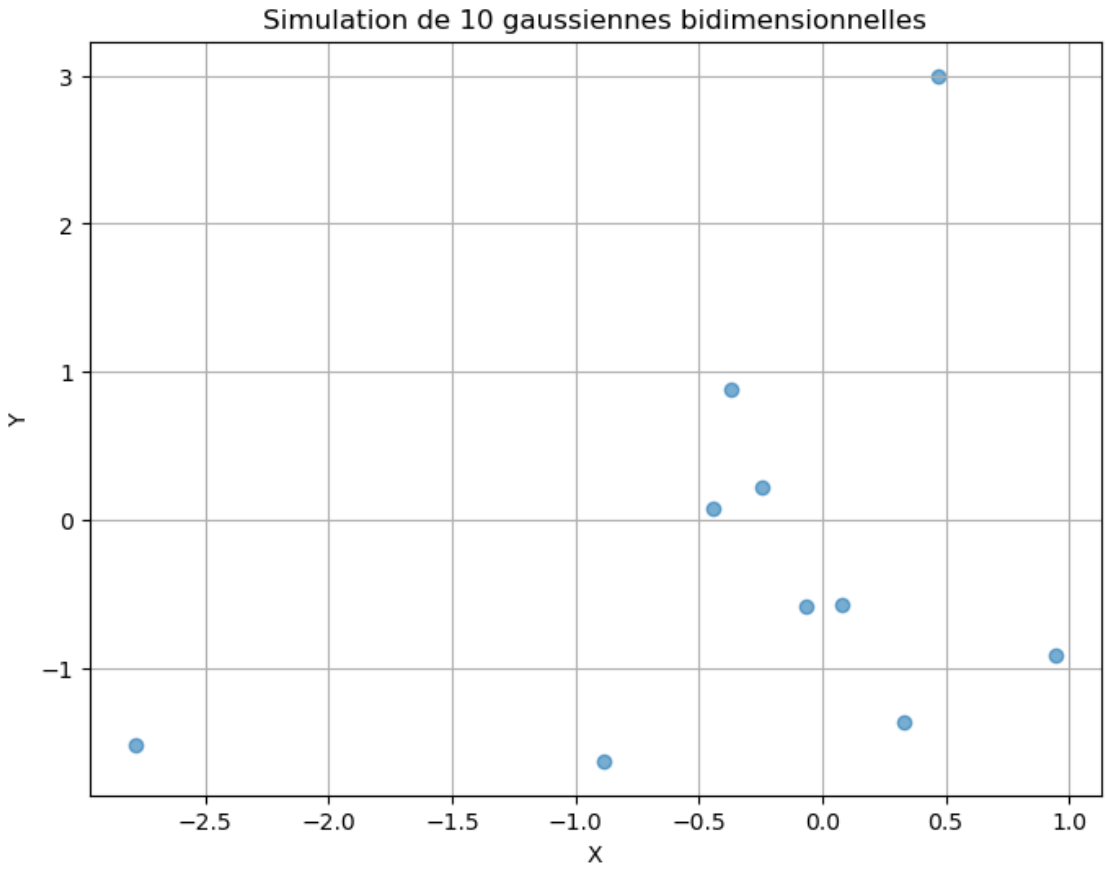


2.3 Exercice 8

2.3.1 Question 1

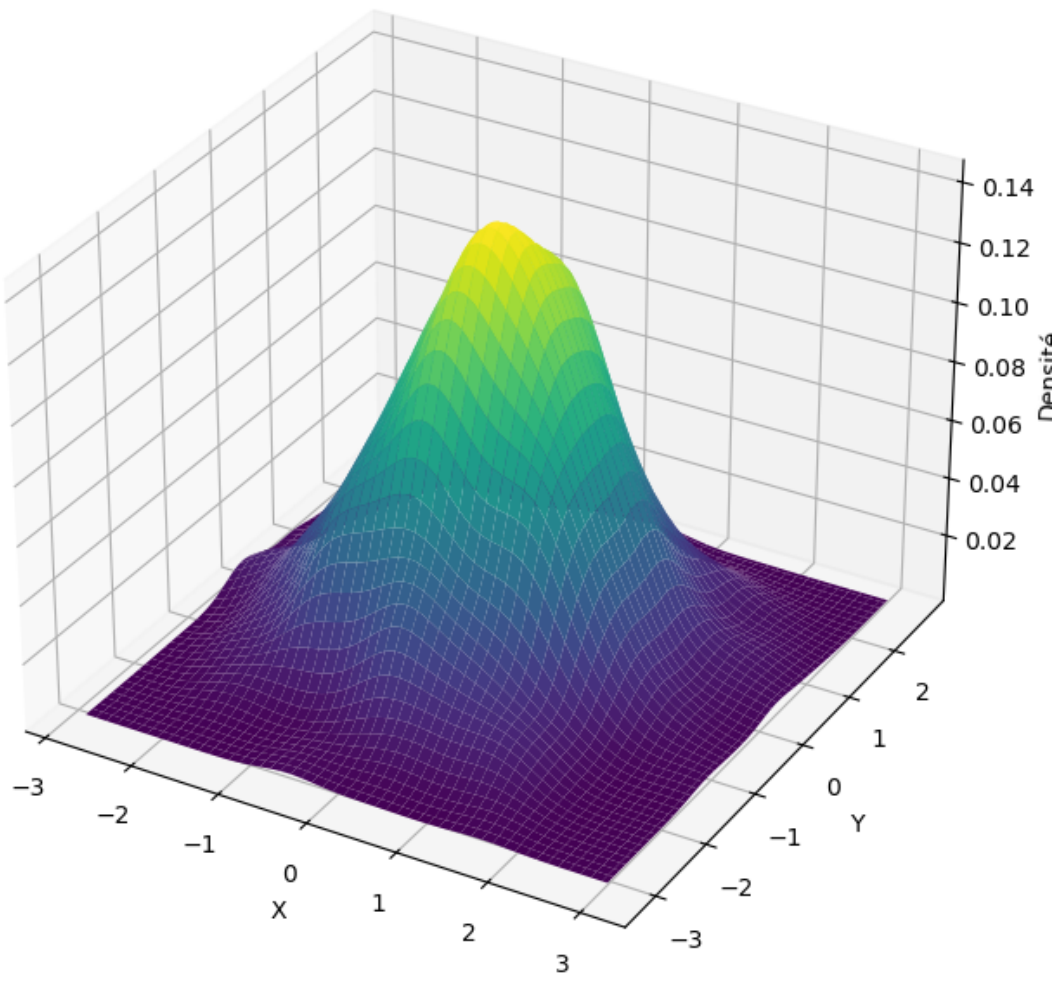
[24]:
```
def simuler_gaussienne_bidimensionnelle():
    u1 = rand()
    u2 = rand()
    R = -2*math.log(u1)
    Theta = u2*2*math.pi
```

```
    return (math.sqrt(R)*math.cos(Theta), math.sqrt(R)*math.sin(Theta))

def simuler_n_gaussienne_bidimensionnelle(n):
    return [simuler_gaussienne_bidimensionnelle() for _ in range(n)]

N = 10

X = simuler_n_gaussienne_bidimensionnelle(N)

# Extraction des coordonnées
x, y = zip(*X)

# Affichage des points
plt.figure(figsize=(8, 6))
plt.scatter(x, y, alpha=0.6)
plt.title('Simulation de 10 gaussiennes bidimensionnelles')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.show()
```



2.3.2 Du ChatGPT de fou

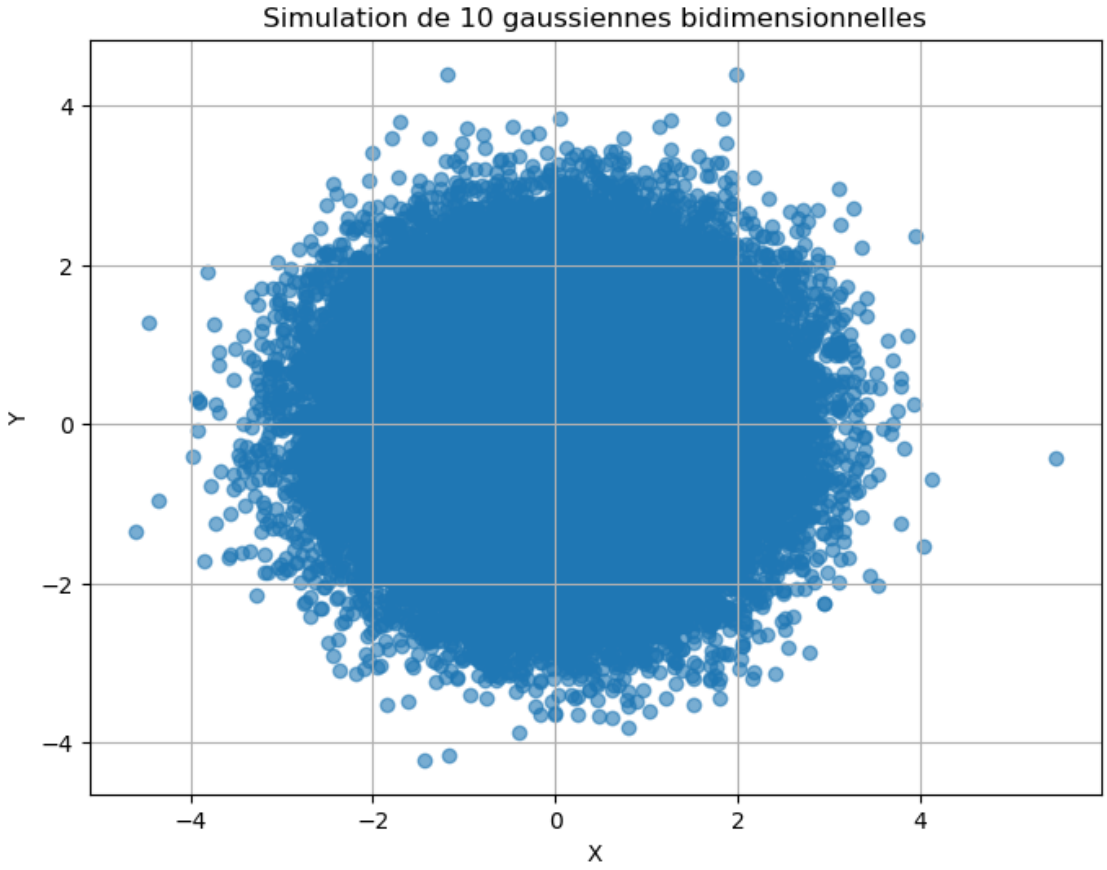
[25]:
```
# Utilisation des points déjà générés pour estimer la densité
data = np.array(simuler_n_gaussienne_bidimensionnelle(1000))  # Génération de␣
↪plus de points pour une meilleure estimation
kde = gaussian_kde(data.T)  # Transposition pour ajuster à l'input attendu par␣
↪gaussian_kde

# Création d'un maillage pour l'évaluation de la densité
xmin, xmax = min(data[:,0]), max(data[:,0])
ymin, ymax = min(data[:,1]), max(data[:,1])
x, y = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]  # j indique un nombre complexe␣
↪pour définir le nombre de pas
positions = np.vstack([x.ravel(), y.ravel()])
z = np.reshape(kde(positions).T, x.shape)

# Affichage 3D de la densité empirique
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='viridis')

ax.set_title('Densité empirique de la gaussienne bidimensionnelle')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Densité')
plt.show()
```



2.3.3 Question 2

[26]:
```
N = 100000

X = simuler_n_gaussienne_bidimensionnelle(N)

# Extraction des coordonnées
x, y = zip(*X)

# Affichage des points
plt.figure(figsize=(8, 6))
plt.scatter(x, y, alpha=0.6)
```

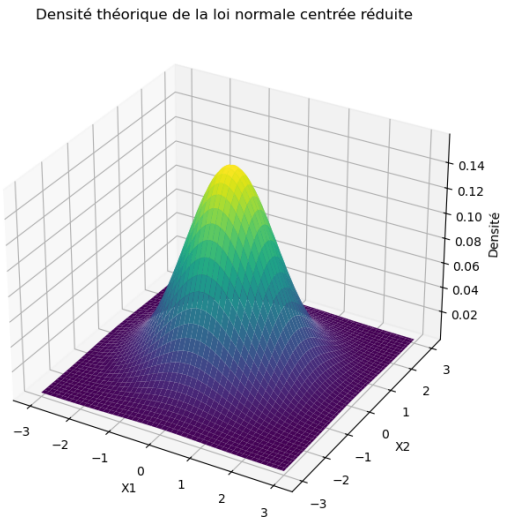
```
plt.title('Simulation de 10 gaussiennes bidimensionnelles')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.show()
```

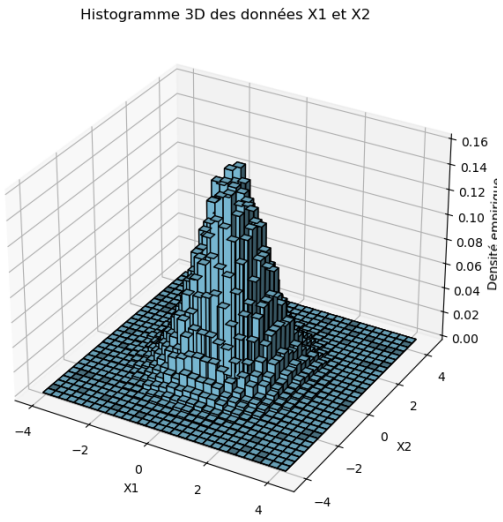


2.3.4 Question 3

[31]:
```
# Densité théorique de la loi normale centrée réduite bidimensionnelle
x_theory = np.linspace(-3, 3, 100)
y_theory = np.linspace(-3, 3, 100)
x_theory, y_theory = np.meshgrid(x_theory, y_theory)
z_theory = np.exp(-(x_theory**2 + y_theory**2) / 2) / (2 * np.pi)

# Histogramme en 3D de X
N = 100000
data = simuler_n_gaussienne_bidimensionnelle(N)
data_np = np.array(data)
```

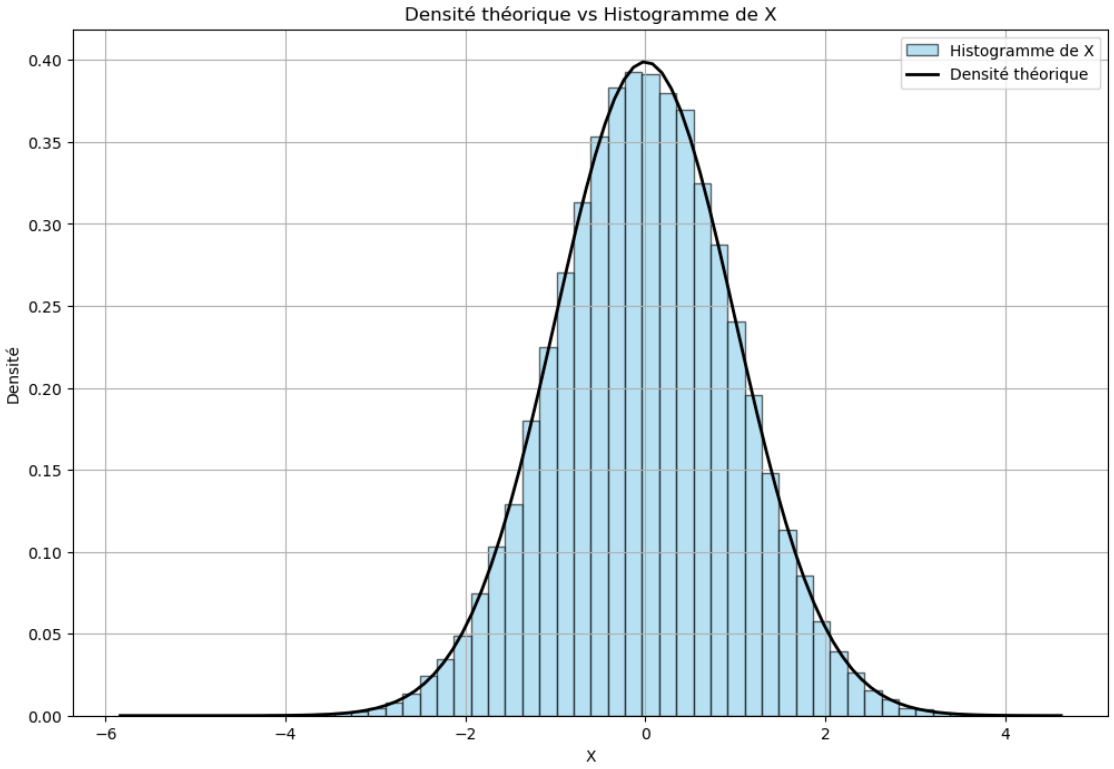
```
# Calcul de l'histogramme 2D avec numpy pour obtenir les hauteurs et les bords␣
 ↪des bacs
hist, xedges, yedges = np.histogram2d(data_np[:,0], data_np[:,1], bins=30,␣
 ↪density=True)

# 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 normale centrée réduite')
ax1.set_xlabel('X1')
ax1.set_ylabel('X2')
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 X1 et X2')
ax2.set_xlabel('X1')
ax2.set_ylabel('X2')
ax2.set_zlabel('Densité empirique')

plt.show()
```





2.3.5 Question 4

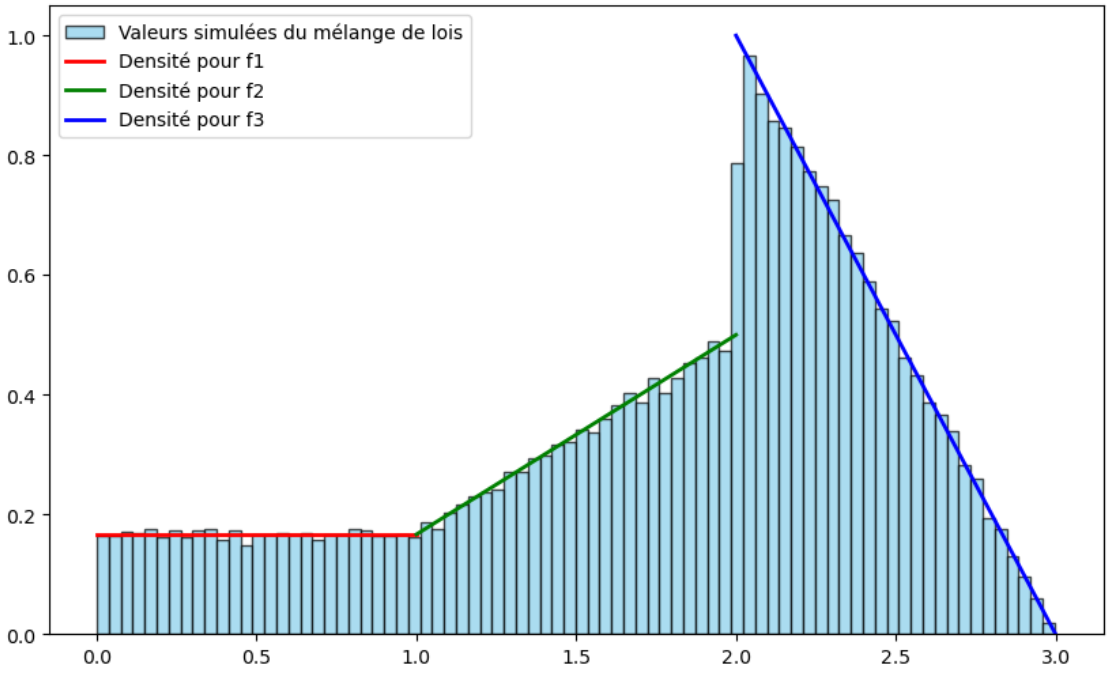
[28]:
```
N = 100000
X = np.array(simuler_n_gaussienne_bidimensionnelle(N))[:, 0]  # Sélection de la␣
↪première dimension

# Création de l'histogramme de X
plt.figure(figsize=(12, 8))
plt.hist(X, bins=50, density=True, alpha=0.6, color='skyblue',␣
↪edgecolor='black', label='Histogramme de X')

# Superposition de la densité théorique de la loi normale centrée réduite
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, 0, 1)
plt.plot(x, p, 'k', linewidth=2, label='Densité théorique')

plt.title('Densité théorique vs Histogramme de X')
plt.xlabel('X')
plt.ylabel('Densité')
plt.legend()
plt.grid(True)
plt.show()
```



2.4 Exercice 9

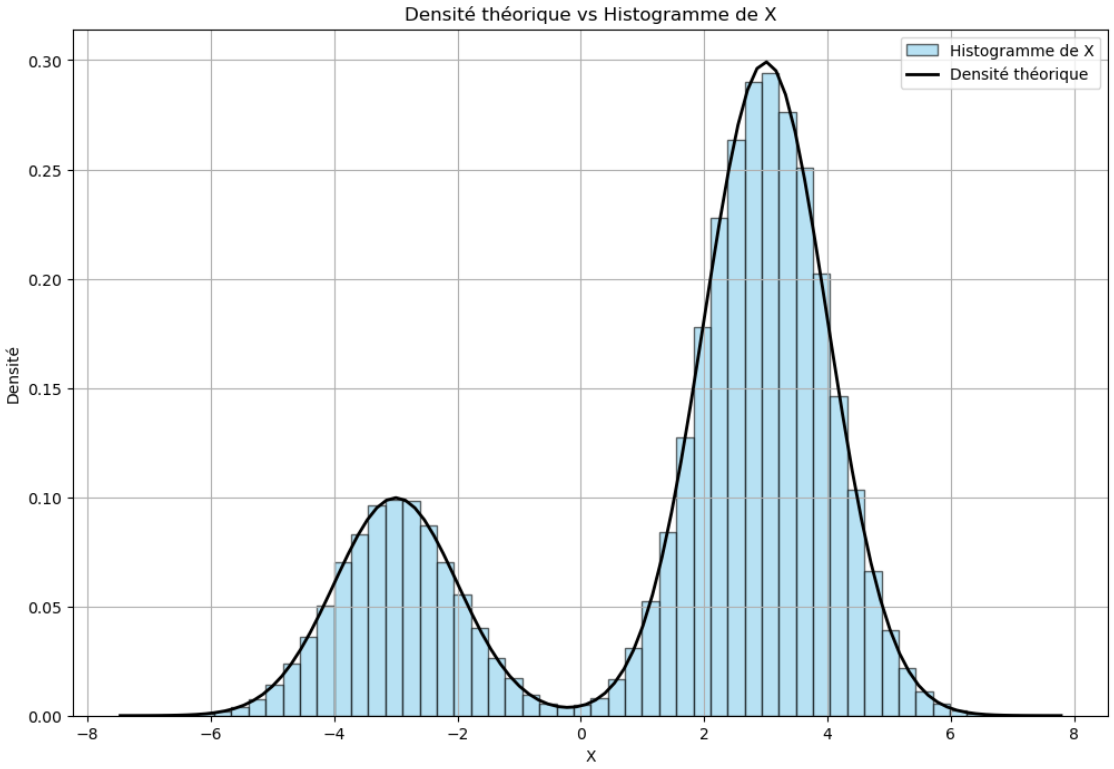
[29]:
```
p = [1/6, 1/3, 1/2]

def simuler_melange_lois():
    valeurs = [1,2,3]
    v = tirage_va_Discrete(valeurs, p)
    if v == 1:
        return rand()
    elif v == 2:
        u = rand()
        return ((1+math.sqrt(1+8*u))/2)
    else:
        u = rand()
        return 3 - math.sqrt(1-u)

def simuler_n_melange_lois(n):
    return [simuler_melange_lois() for _ in range(n)]

N = 100000
X = simuler_n_melange_lois(N)
```

```python
# Création de l'histogramme
plt.figure(figsize=(10, 6))
plt.hist(X, bins=80, density=True, color='skyblue', edgecolor='black', alpha=0.
 ↪7, label='Valeurs simulées du mélange de lois')

x1 = np.linspace(0, 1, 100)
f1 = np.ones_like(x1) * p[0]

x2 = np.linspace(1, 2, 100)
f2 = (x2 - 1/2) * p[1]

x3 = np.linspace(2, 3, 100)
f3 = (6 - 2*x3) * p[2]

plt.plot(x1, f1, 'r-', linewidth=2, label='Densité pour f1')
plt.plot(x2, f2, 'g-', linewidth=2, label='Densité pour f2')
plt.plot(x3, f3, 'b-', linewidth=2, label='Densité pour f3')

plt.legend()
plt.show()
```



2.5 Exercice 10

[30]:
```
def simuler_loi_normale():
    return simuler_gaussienne_bidimensionnelle()[0];


N = 100000
p = [1/4, 3/4]

def simuler_melange_gaussiennes():
    valeurs = [1,2]
    v = tirage_va_Discrete(valeurs, p)
    if v == 1: return (simuler_loi_normale() - 3)
    else: return (simuler_loi_normale() + 3)

def simuler_n_melange_gaussiennes(n):
    return [simuler_melange_gaussiennes() for _ in range(n)]


X = simuler_n_melange_gaussiennes(N)

# Création de l'histogramme de X
plt.figure(figsize=(12, 8))
plt.hist(X, bins=50, density=True, alpha=0.6, color='skyblue',␣
 ↪edgecolor='black', label='Histogramme de X')

# Superposition de la densité théorique de la loi normale centrée réduite
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
pp = p[0] * norm.pdf(x, -3, 1) + p[1] * norm.pdf(x, 3, 1)
plt.plot(x, pp, 'k', linewidth=2, label='Densité théorique')

plt.title('Densité théorique vs Histogramme de X')
plt.xlabel('X')
plt.ylabel('Densité')
plt.legend()
plt.grid(True)
plt.show()
```


Densité théorique vs Histogramme de X
Histogramme de X
Densité théorique
Densité
X
0.00
0.05
0.10
0.15
0.20
0.25
0.30
−8
−6
−4
−2
0
2
4
6
8