TP 3 - Spread spectrum watermarking

HTI

Digital Content Protection

Marilou Grignoli

Andrea Miens

Thibault Ellong

OBJECTIF :

L'objectif de ce troisième TP du cours de Digital Content protection est d'implémenter une méthode de "Spread Spectrum Watermarking".
D'abord, nous verrons le principe de générateurs de nombre aléatoire uniforme et gaussien. Ensuite nous étudierons le concept de fonctions de corrélation, leur représentation graphique et leurs propriétés. Enfin, nous nous pencherons sur le watermarking CDMA (Code Division Multiple Access).

Pour réaliser ce TP, nous utiliserons les images Lena et Baboon en niveaux de gris.

I/ Générateur de nombres aléatoires

On initialise avec une valeur difficile à prédire, par exemple les millisecondes de l'horloge système. On applique ensuite une formule de génération récursive, par exemple le Linear Congruential Generator (LCG) défini par:

$$x = (a * x + b) \, mod \, c$$

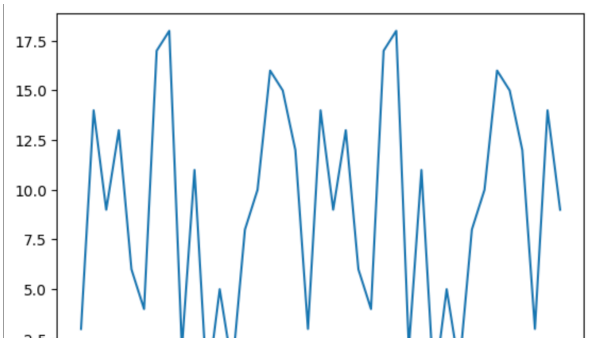
où

- a, b, c sont des entiers positifs
- c peut être un nombre premier
- a < c
- b < c

Voici notre code :

```
def lcg(xinit, a, b, c):
	liste = [xinit]
	i=1
	while(i<=2*c):
		liste.append((a*liste[i-1] + b)%c)
		i+=1
	return liste
```

EXEMPLE 1
Dans cet exemple nous générons une séquence aléatoire avec un LCG en choisissant les paramètres suivants : $c = 19$, $a = 3$, $b = 5$, and $x_0 = 3$
On obtient la séquence suivante :
[3, 14, 9, 13, 6, 4, 17, 18, 2, 11, 0, 5, 1, 8, 10, 16, 15, 12, 3, 14, 9, 13, 6, 4, 17, 18, 2, 11, 0, 5, 1, 8, 10, 16, 15, 12, 3, 14, 9]



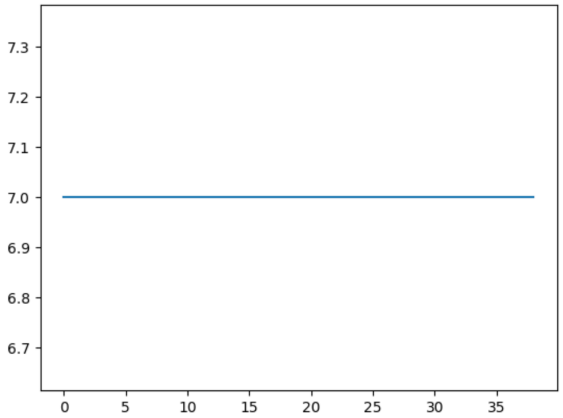
En affichant les résultats de notre fonction LCG, on se rend compte que notre fonction est cyclique.

Malheureusement, cela signifie que nos paramètres ne nous permettent pas d'obtenir une génération de valeurs vraiment aléatoires. La séquence est : 15, 12, 3, 14, 9, 13, 6, 4, 17, 18, 2, 11, 0, 5, 1, 8, 10, 16.

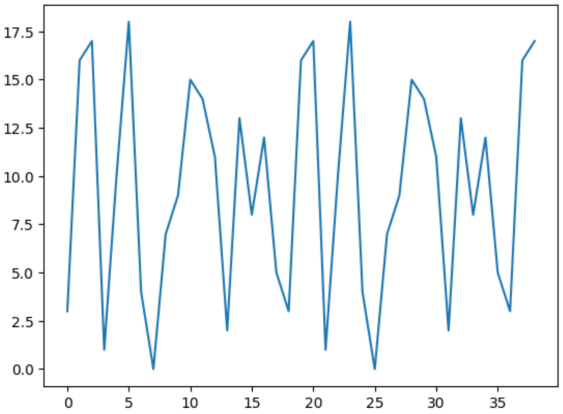
EXEMPLE 2:

Nous changeons les paramètres :

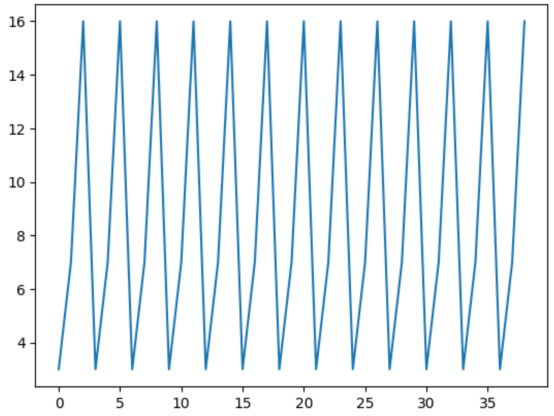
- c = 19, a = 3, b = 5, and x_0 = 7



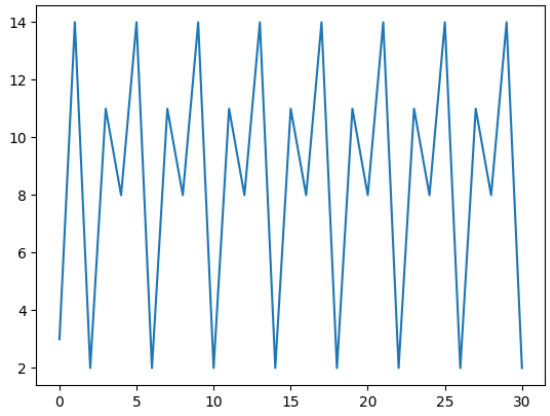
- c = 19, a = 3, b = 7, and x_0 = 3



- c = 19, a = 7, b = 5, and x_0 = 3



- c = 15, a = 3, b = 5, and x_0 = 3



Conclusion:

Ces autres exemples nous montrent que la génération de nombres aléatoires avec la méthode LCG est cyclique. Il faut donc changer régulièrement les paramètres de la fonction après un temps T inférieur à la période si l'on souhaite avoir une séquence aléatoire.

EXEMPLE 3 :

Dans cet exemple nous implémentons la Multiple LGC décrit par:

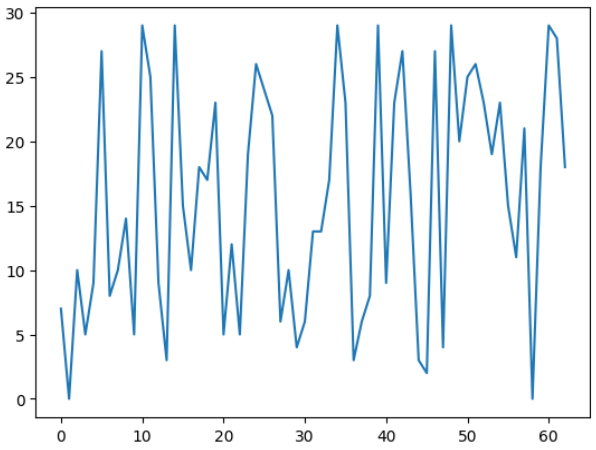
$$xn = (a\,xn - 1 + b\,xn - 2 + c)\,mod\,d$$

où

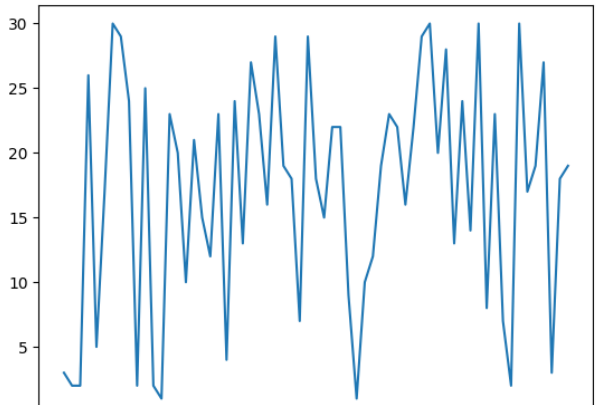
- a, b, c, d sont des entiers positifs
- d peut être un nombre premier
- a < d
- b < d
- c < d

```
def mlcg(xinit, xbis, a, b, c, d):
	liste = [xinit, xbis]
	i=2
	while(i<=2*d):
		liste.append((a*liste[i-1] + b*liste[i-2] + c)%d)
		i+=1
	return liste
```

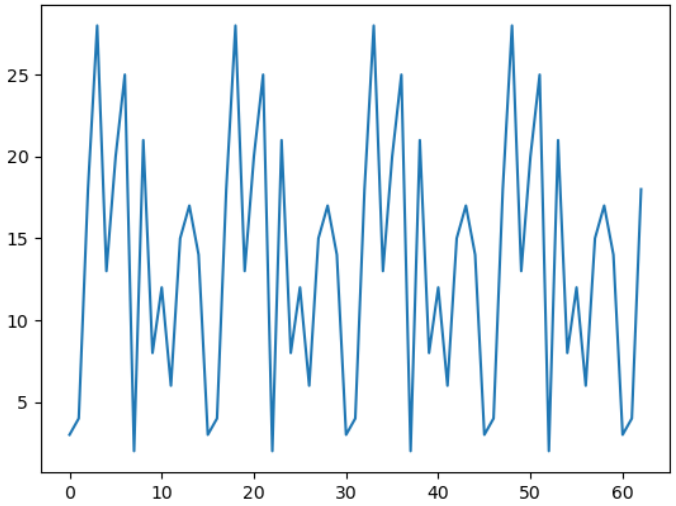
- d = 31, c = 6, a = 3, b = 5, and x_0 = 7, x_1 = 0 :



- d = 31, c = 6, a = 3, b = 7, and x_0 = 3, x_1 = 2 :

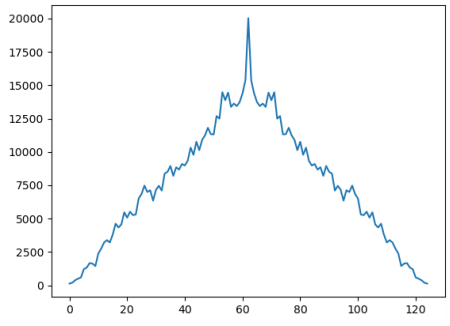


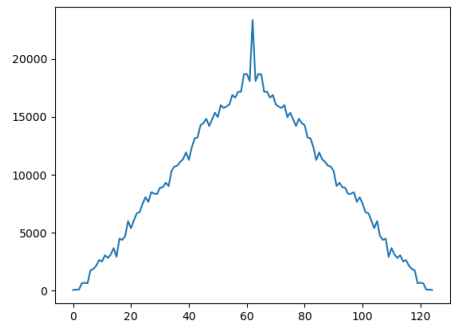
- d = 31, c = 6, a = 7, b = 5, and x_0 = 3, x_1 = 4 :

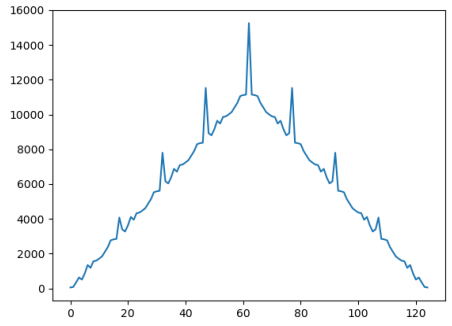


Conclusion :

Cette seconde méthode de MLCG permet de générer des séquences aléatoires plus longues avant qu'elles ne se mettent à boucler comme apparent sur le troisième graphe.







En regardant les résultats d'auto-corrélation à partir des résultats des fonctions réalisées ci-dessus (l'ordre est le même qu'avant). On se rend compte que la corrélation entre chaque résultat est assez importante et montre que les résultats ne sont pas très aléatoires mais bien liés entre eux.
Pour être plus proche d'une fonction aléatoire, on souhaite que la fonction d'autocorrélation soit proche du dirac (ie. chaque valeur est décorrélée des autres à part avec elle-même), de manière très claire, nous en sommes loin.
De plus, dans le troisième cas, on remarque même les pics de corrélation lorsque la fonction que l'on étudie boucle.

Autres générateurs : LFSR - Registre à décalage à rétroaction linéaire
Génère des séquences idéalement non corrélées (corrélation nulle).
Pour aller vers des générateurs prêts à l'emploi, les symboles générés selon ces principes sont fusionnés ensemble afin d'obtenir des valeurs en virgule flottante.

EXEMPLE 4 : Étude d'un générateur uniforme [0, 1]

Considérons T = 100. Nous allons générer un vecteur x de T composantes, uniformément distribuées entre 0 et 1.
Puis nous allons calculer l'approximation de sa valeur moyenne et de sa variance pour les comparer aux valeurs théoriques.
Pour commencer, nous calculons le vecteur y = a x + b, où a = 1 et b = 6 ; Enfin, nous répéterons les 2 points précédents et expliquerons les résultats.
Nous réitérerons cet exemple en changeant les valeurs de a,b et T.

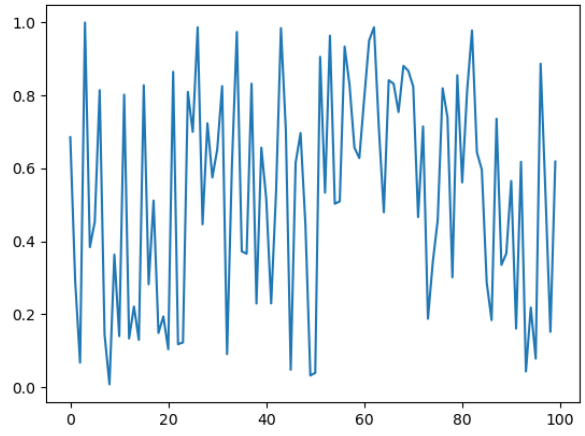
La fonction qui génère le vecteur x avec T composantes est :

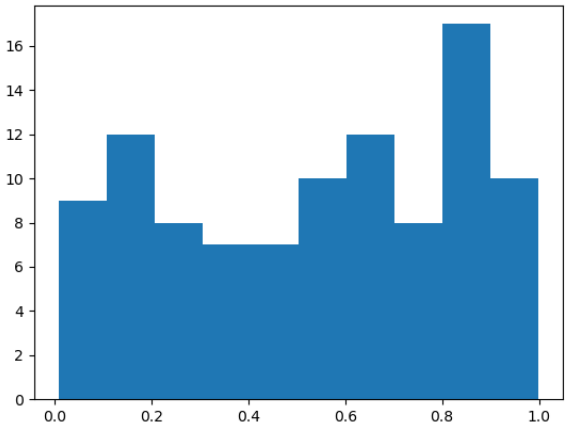
```
def unigenerator(T):
    x = np.random.uniform(0, 1, T)
    return x
```

De plus, pour répondre à la suite des questions, nous avons besoin d'une fonction moyenne, variance et ainsi que la fonction d'affichage des graphes nécessaires:

```
def moyenne(x):
    moy_cal = 0
    for i in x:
        moy_cal+=i
    moy_cal = moy_cal/len(x)
    print(moy_cal)
    return moy_cal

def variance(x, moy):
    var_cal = 0
    for i in range(0, len(x)):
        var_cal+=(x[i]-moy)**2
    var_cal+=var_cal/len(x)
    return var_cal

def plot_result_uni(T):
    x = unigenerator(T)
    moy = moyenne(x)
    var =variance(x, moy)
    print( 'Moyenne :'+str(moy)+' et la Variance : '+str(var) )
    plt.plot(x)
    plt.figure()
    plt.hist(x, 10)
    plt.show()

plot_result_uni(100)
```

Ainsi, on obtient le graphique et l'histogramme suivant :





La moyenne et la variance sont assez proches des résultats théoriques respectivement de 0,5 et 1/12~0.083, les résultats expérimentaux sont :

```
Moyenne :0.49934817782510016 et la Variance : 0.07619981019346495
```

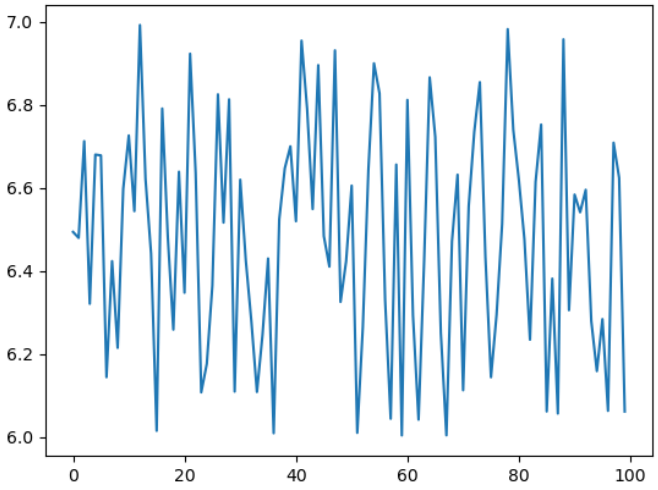
On remarque la très forte proximité entre nos résultats et la théorie.

Génération du vecteur y, des graphes et valeurs associées :

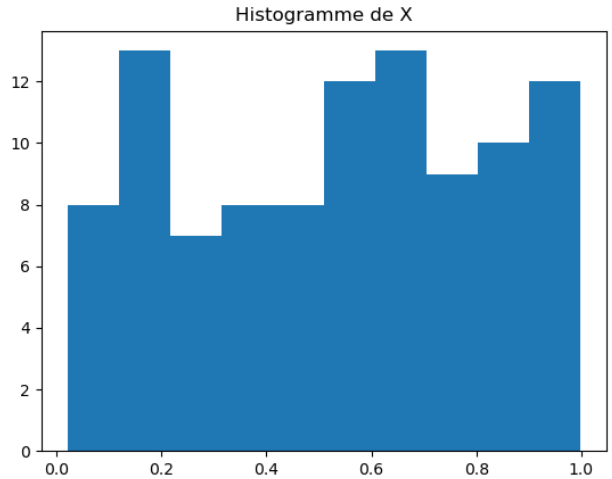
Code associé à l'exercice pour a = 1 et b = 6 :

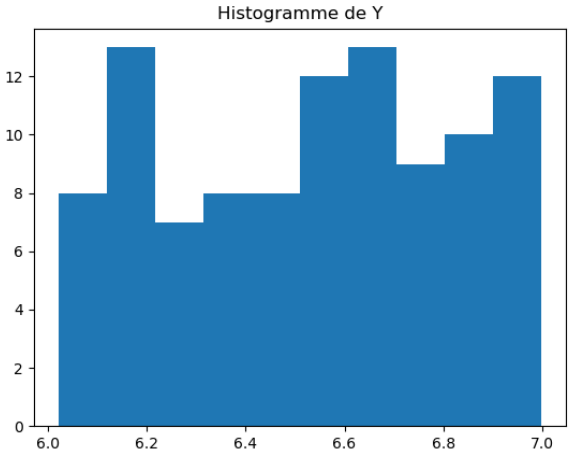
```python
def plot_result_uni(T):
    x = unigenerator(T)
    moy = moyenne(x)
    var =variance(x, moy)
    print( 'Moyenne de x :'+str(moy)+' et la Variance de x : '+str(var))
    y = 1*x + 6
    moy_y = moyenne(y)
    var_y = variance(y, moy_y)
    print( 'Moyenne de y :'+str(moy_y)+' et la Variance de y : '+str(var_y))
    plt.plot(x)
    plt.figure()
    plt.plot(y)
    plt.figure()
    plt.hist(x, 10)
    plt.figure()
    plt.hist(y, 10)
    plt.show()
```

Graphique de y :



Histogramme de X et Y:





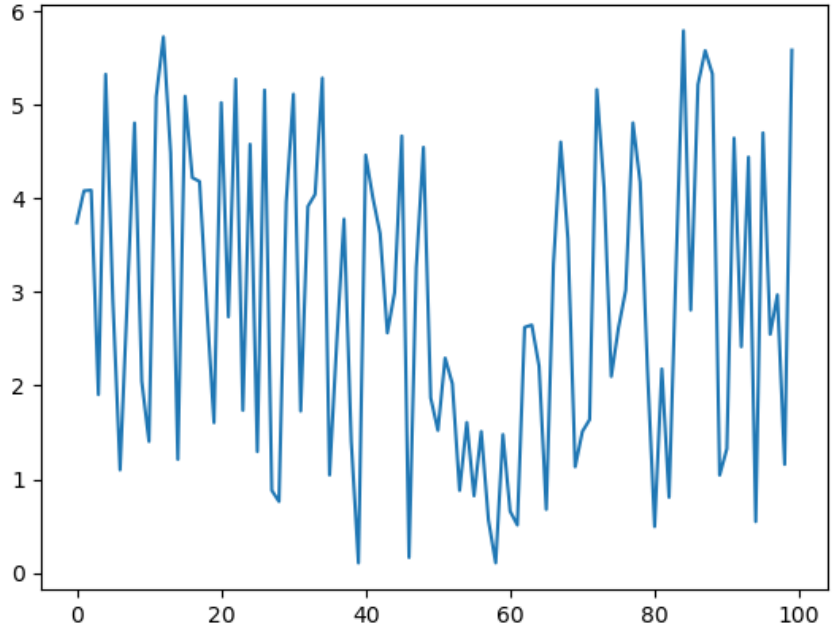
Les résultats de x et y, pour rappel les résultats théoriques pour y sont 6,5 pour la moyenne (moyenne théorique de x + b, dans notre cas 6) et la variance de y doit être la même que celle de x car la variance ne change pas par rapport à une translation :

```
Moyenne de x :0.5347647447162514 et la Variance de x : 0.08160579054652312
Moyenne de y :6.534764744716252 et la Variance de y : 0.08160579054652306
```

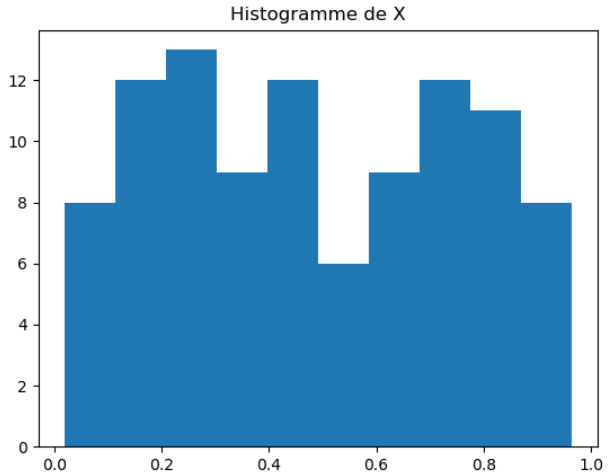
On observe bien les résultats escomptés.

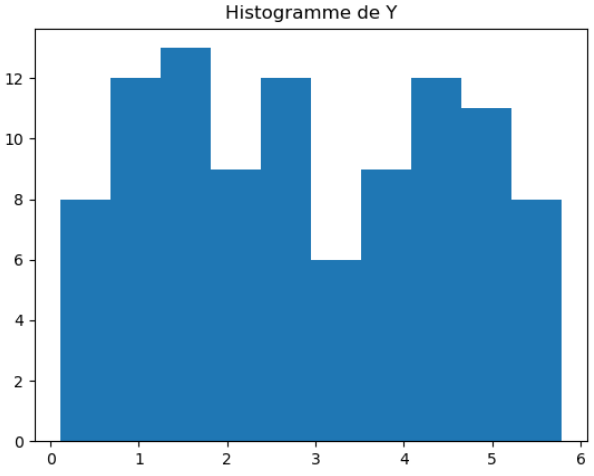
Résultats associé à l'exercice pour a = 6 et b = 0 :

Graphique de y :



Histogramme de X et Y:





Les histogrammes ont la même forme mais sont sur un intervalle plus large pour y dû à la multiplication par 6.
Résultats expérimentaux pour la moyenne et la variance de Y:

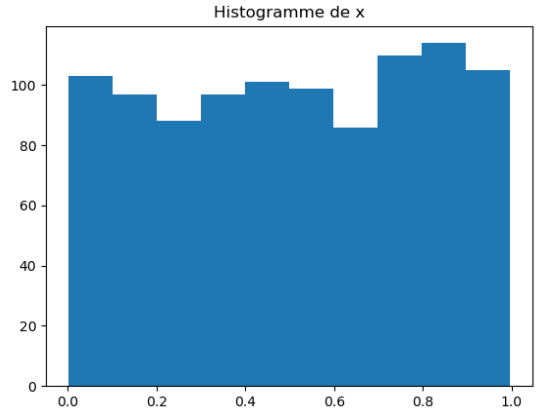
```
Moyenne de x :0.48379260219481274 et la Variance de x : 0.07379216605050297
Moyenne de y :2.902755613168876 et la Variance de y : 2.656517977818106
```

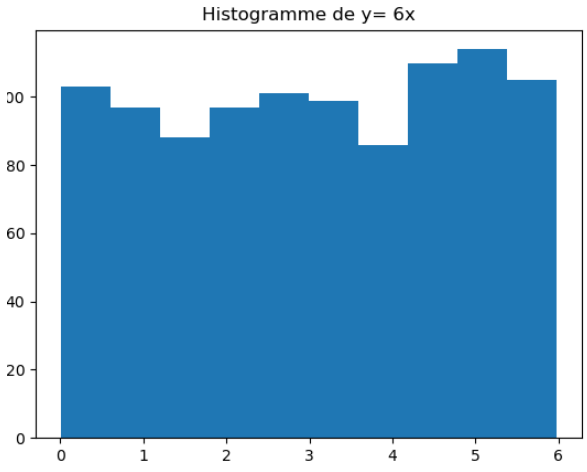
Là encore, les résultats sont cohérents car la moyenne théorique est le produit de a et la moyenne théorique de x c'est-à-dire 3. De même pour la variance, on sait que la

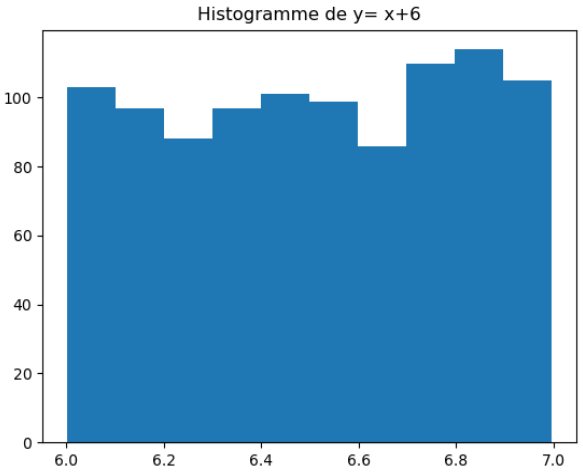
variance théorique de y est le produit de la variance théorique de x et a au carré dans notre cas 36/12=3.

<u>Résultats pour T=1000 et T= 10000 :</u>

Pour les histogrammes de x, y avec a=1 et b=6, y avec a=6 et b=0 et T= 1000:

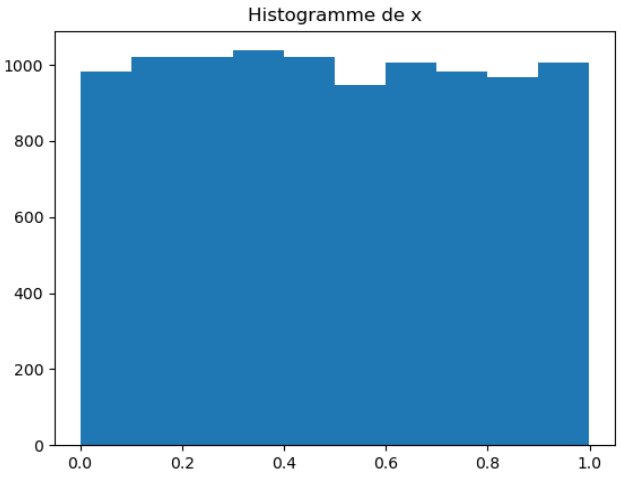


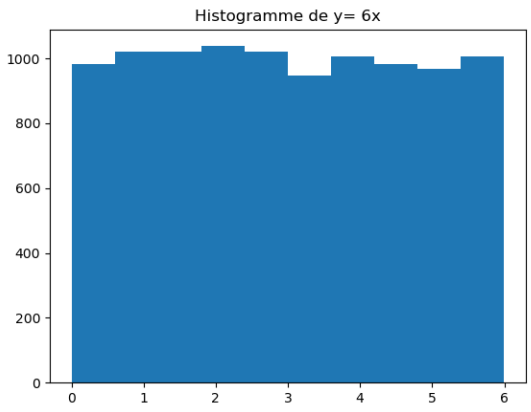


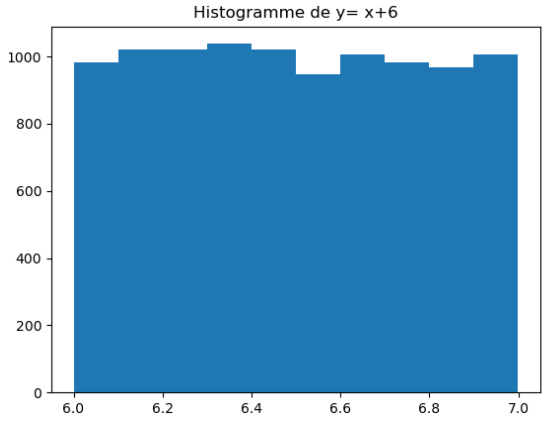



```
Moyenne de x :0.5080945751607491 et la Variance de x : 0.08640556803700102
Moyenne de y avec (6, 0) :3.04856745096449 et la Variance de y avec (6, 0) : 3.110600449332032
Moyenne de y avec (1, 6) :6.508094575160748 et la Variance de y avec (1, 6) : 0.08640556803700092
```

Pour les histogrammes de x, y avec a=1 et b=6, y avec a=6 et b=0 et T= 10000 :








```
Moyenne de x :0.49699443968243856 et la Variance de x : 0.08280097330679986
Moyenne de y avec (6, 0) :2.9819666380946326 et la Variance de y avec (6, 0) : 2.980835039044802
Moyenne de y avec (1, 6) :6.496994439682448 et la Variance de y avec (1, 6) : 0.08280097330679963
```

Conclusion :

En augmentant la valeur de T, on remarque qu'on se rapproche beaucoup plus des résultats théoriques que l'on est censé observer. Ces résultats sont cohérents d'après la loi des grands nombres.

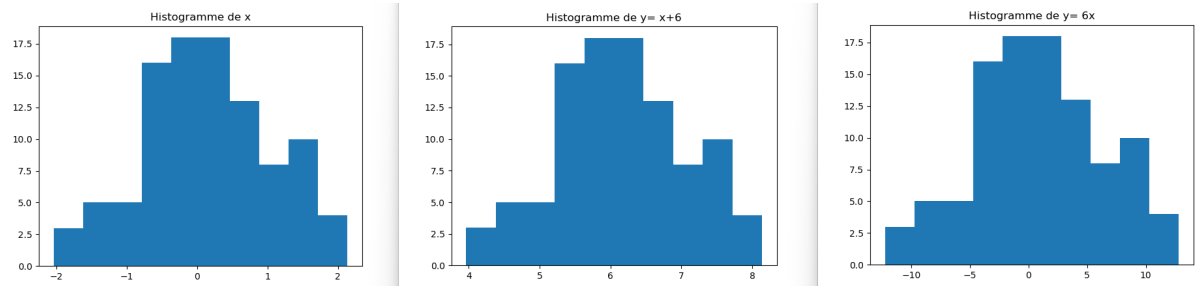
EXEMPLE 5 : Étude d'un générateur gaussien standard

Reprenons l'exemple précédent pour le générateur gaussien.

Voici le code pour générer un vecteur x de longueur T rempli d'éléments générer avec une gaussienne:

```
def normalgenerator(T):
    x =np.random.normal(0, 1, T)
    return x
```

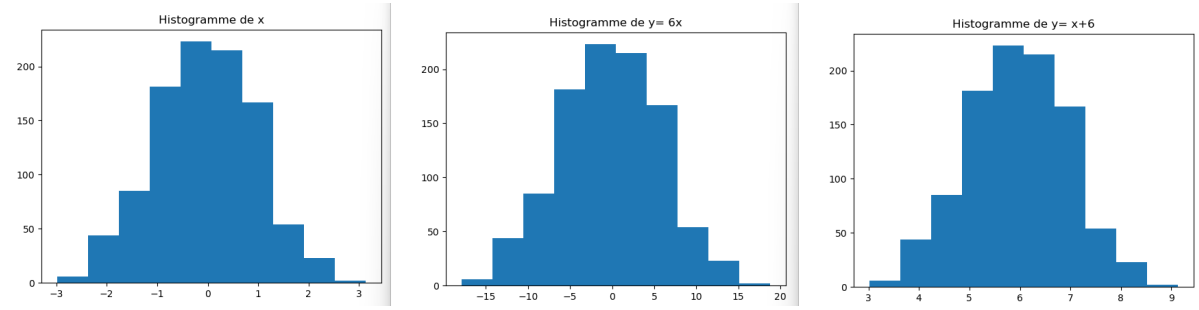
Résultats pour T=100:



Les résultats attendus sont cohérents car ils correspondent au même cas théorique que vu dans l'exemple 4.

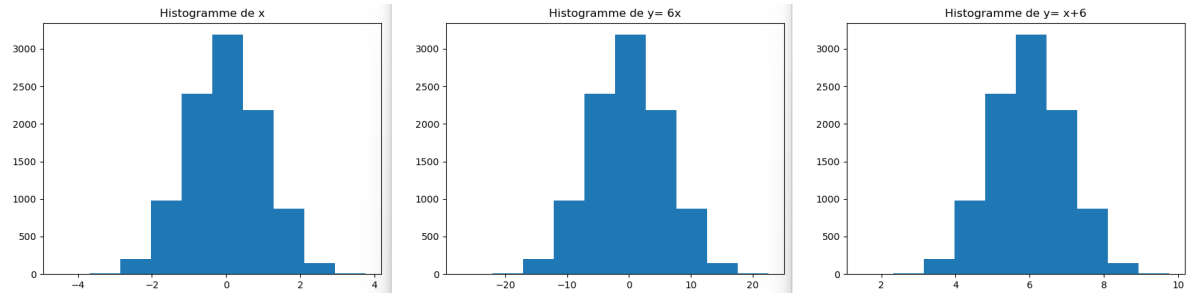
```
Moyenne de x :0.13363704472505017 et la Variance de x : 0.8270728631320377
Moyenne de y avec (6, 0) :0.8018222683503013 et la Variance de y avec (6, 0) : 29.77462307275336
Moyenne de y avec (1, 6) :6.133637044725052 et la Variance de y avec (1, 6) : 0.8270728631320377
```

Pour T = 1000:




```
Moyenne de x :-0.04109138954601498 et la Variance de x : 0.9910380351561939
Moyenne de y avec (6, 0) :-0.24654833727609016 et la Variance de y avec (6, 0) : 35.67736926562301
Moyenne de y avec (1, 6) :5.95890861045399 et la Variance de y avec (1, 6) : 0.991038035156194
```

Pour T=10000:




```
Moyenne de x :-0.008822851428769388 et la Variance de x : 1.0040021377905468
Moyenne de y avec (6, 0) :-0.05293710857261589 et la Variance de y avec (6, 0) : 36.1440769604596
Moyenne de y avec (1, 6) :5.991177148571208 et la Variance de y avec (1, 6) : 1.0040021377905441
```

Conclusion :

Tout d’abord, on remarque que plus la valeur de T augmente, plus les histogrammes s'approchent de la courbe théorique d’une gaussienne.

Pour rappel, pour la gaussienne de x, la moyenne théorique est 0 et la variance est 1 car ce sont les paramètres que nous avons fixés.

Cela implique que pour y=6x, la variance théorique est 36 et sa moyenne théorique est 0.
De même pour y=x+6, sa moyenne théorique est 6 et sa variance théorique est 1.

En augmentant, la valeur d’échantillon pour x, on a encore une fois un diminution de l’écart entre la valeur effective et la valeur théorique, ce qui est cohérent.

II/ Fonctions de Corrélation

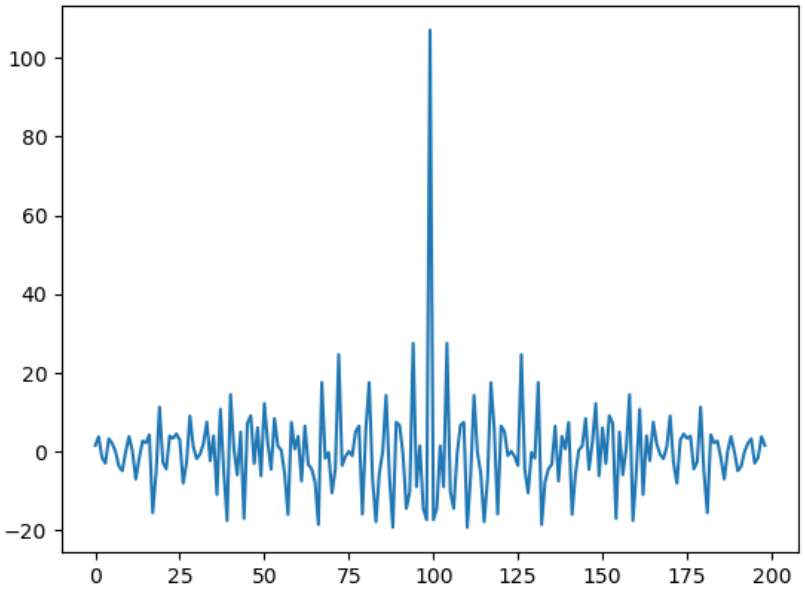
EXEMPLE 6: Fonction d'Autocorrélation

Considérons T = 100. Dans cette partie nous allons générer un vecteur x de T composantes, suivant une loi gaussienne standard. Nous allons calculer et représenter sa fonction d'autocorrélation puis comparer les résultats aux valeurs théoriques attendues. Nous ferons également cet exemple pour T = 1000 et T = 10000.

Voici le code pour obtenir le résultat de l'autocorrélation de la fonction gaussienne pour T=100.

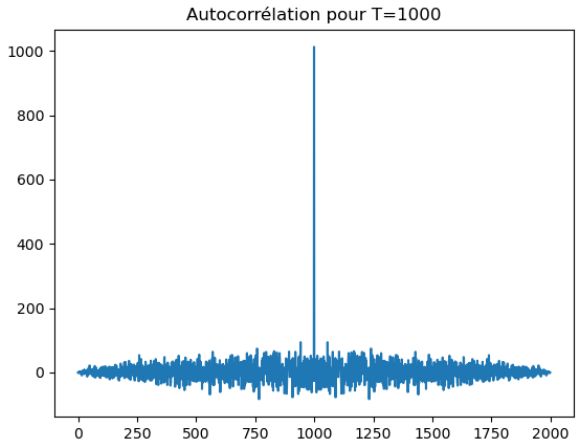
```
def auto_correlate_norm():
    x = normalgenerator(100)
    correlx = np.correlate(x, x, mode='full')
    plt.plot(correlx)
    plt.show()
```

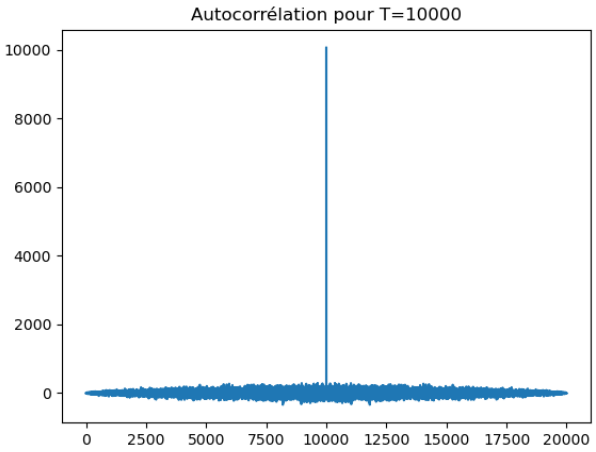
Résultat de l'autocorrélation :



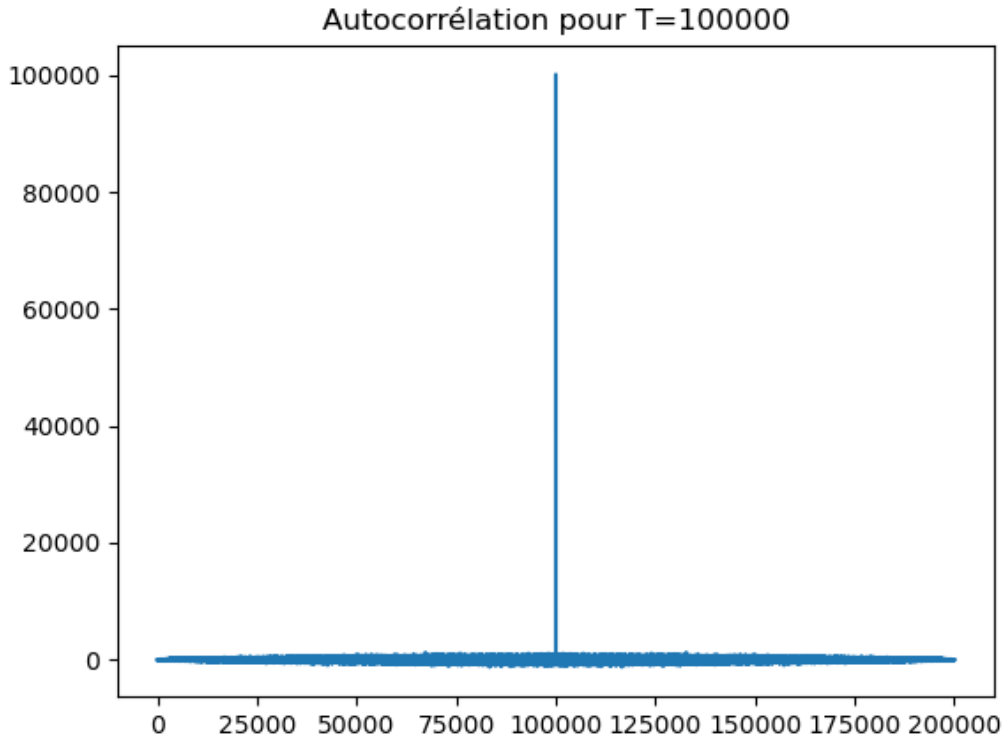
On remarque tout d'abord que l'on observe bien le pic central qui correspond à la corrélation du point avec lui-même.
Pour une fonction aléatoire, on se rend compte qu'un nombre important de points ont l'air d'être corrélés ce qui n'est pas tout à fait ce que l'on souhaiterait car on aimerait des valeurs aléatoires.
Encore une fois, le résultat théorique serait l'obtention d'un dirac, or on en est encore loin.

Pour T= 1000 et 10000:





En augmentant la valeur de T, on obtient des résultats qui s'approchent beaucoup plus de la théorie ce qui devient nettement plus satisfaisant. On en déduit que la génération de nombre aléatoire à partir de la gaussienne est acceptable.

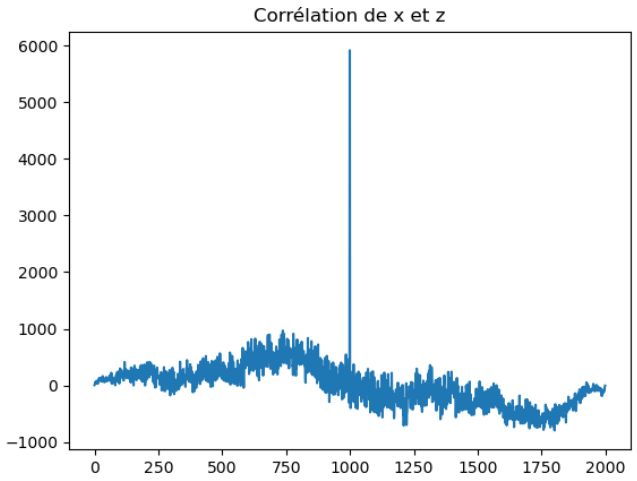


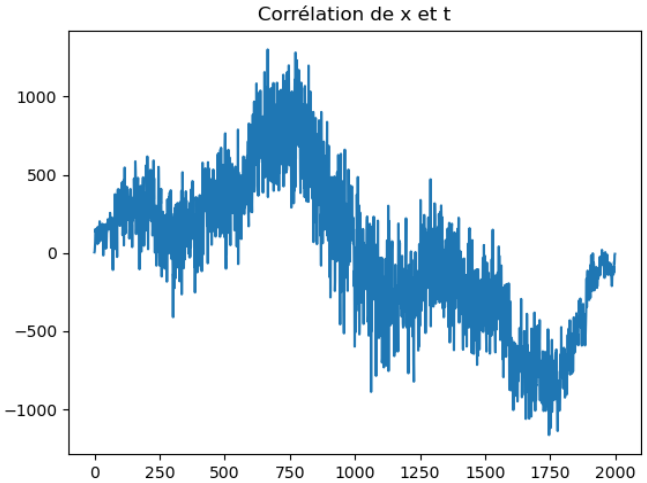
EXEMPLE 7 : Fonction de cross-corrélation

Considérons T = 1000. Nous allons considérer z = 6x + 16 et t = 6x^2 + 16. L'objectif est de calculer et représenter les fonctions de cross-corrélation entre x et z ainsi qu'entre x et t.

Code pour réaliser cet exercice:

```
def cross_correlation(T):
    x = normalgenerator(T)
    z = 6*x+16
    xsquare = np.array(len(x))
    for i in range(0, len(x)):
        xsquare[i] = x[i]**2
    t = 6*xsquare + 16
    plt.plot(np.correlate(x, z, mode='full'))
    plt.title('Corrélation de x et z')
    plt.figure()
    plt.title('Corrélation de x et t')
    plt.plot(np.correlate(x, t, mode='full'))
    plt.show()

cross_correlation(1000)
```





Dans le premier cas, on conserve une corrélation importante au centre, ce qui paraît cohérent car il s'agit d'une transformation linéaire de notre vecteur initial. De plus, la corrélation est plus faible dans le cas de x et t car on perd la linéarité entre les valeurs qu'on a généré et les nouvelles valeurs.

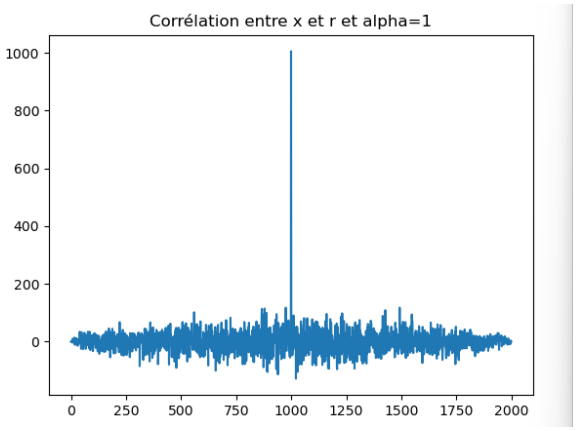
EXEMPLE 8 : Étude de la robustesse face à l'ajout de bruit

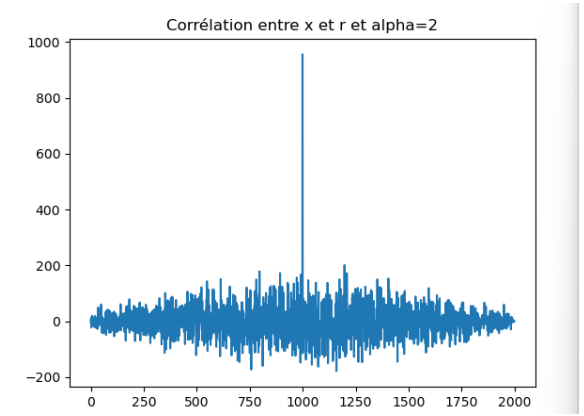
Considérons T = 1000, alpha = 1. Nous allons générer deux vecteurs x et n de T composantes, suivant une loi gaussienne standard ; En considérant r = x + alpha * n. nous allons calculer et représenter la fonction de cross-corrélation entre x et r.
Nous prendrons ensuite alpha = 2, alpha = 8, alpha = 64 ; puis T = 10000 et T = 100000.

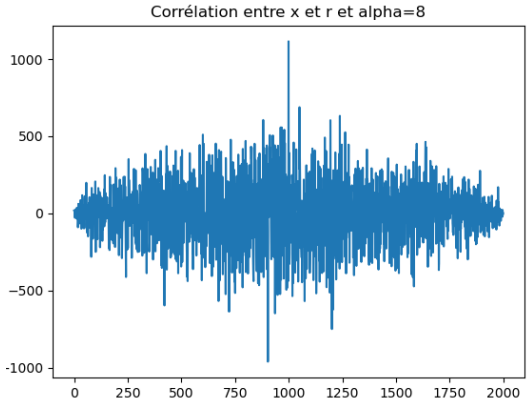
Code pour résoudre cet exemple et le suivant :

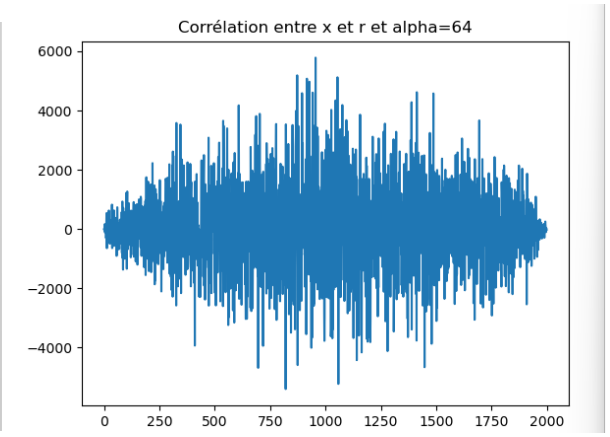
```
def cdma(T, alpha):
    x = normalgenerator(T)
    y = normalgenerator(T)
    n = normalgenerator(T)
    r = x + alpha*n
    r2 = x + y + alpha*n
    return x, y, r, r2


def plot_correlatecdma(T, alpha):
    x, y, r, r2=cdma(T, alpha)
    correlx=np.correlate(x, r, mode='full')
    correly=np.correlate(y, r, mode='full')
    correlrx=np.correlate(x, r2, mode='full')
    correlry=np.correlate(y, r2, mode='full')
    plt.plot(correlx)
    plt.figure()
    plt.plot(correly)
    plt.figure()
    plt.plot(correlrx)
    plt.figure()
    plt.plot(correlry)
    plt.show()

for T in [ 1000, 10000, 100000]:
    for alpha in [2, 8, 64]:
        plot_correlatecdma(T, alpha)
```

T=1000 :

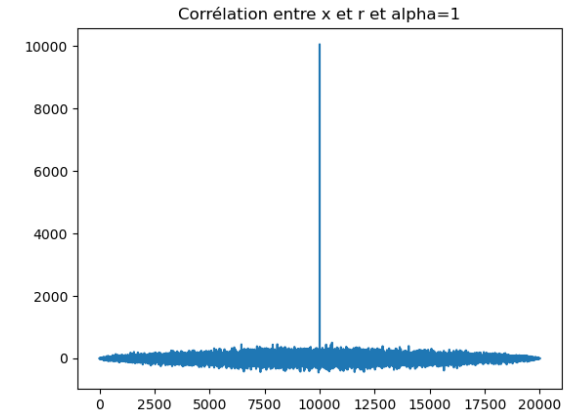


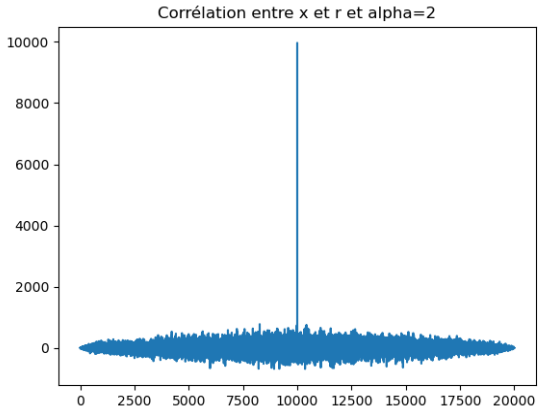




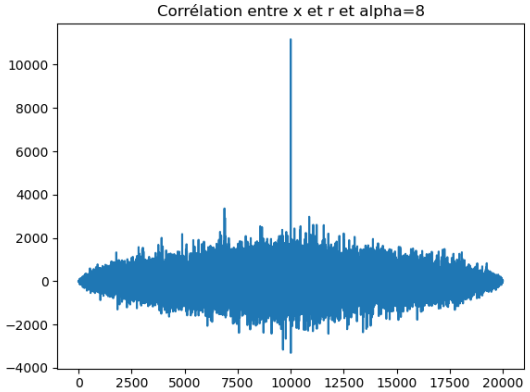


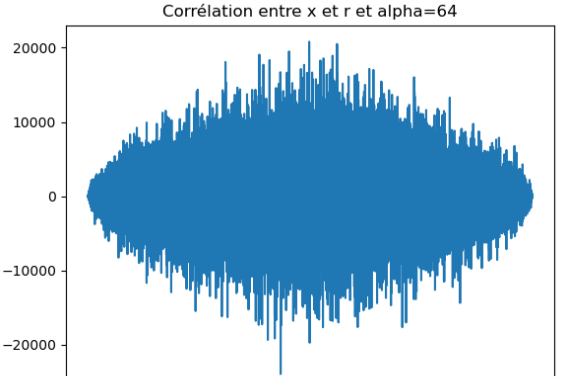
T=10000 :



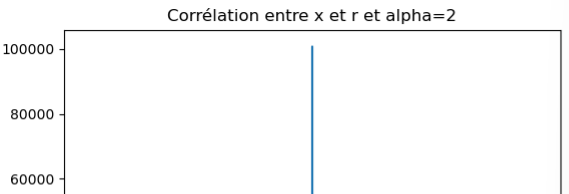


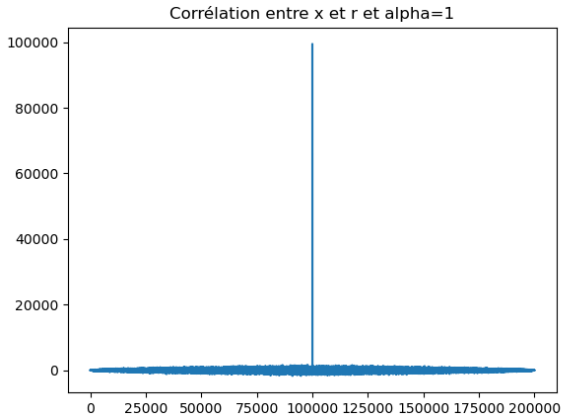
-

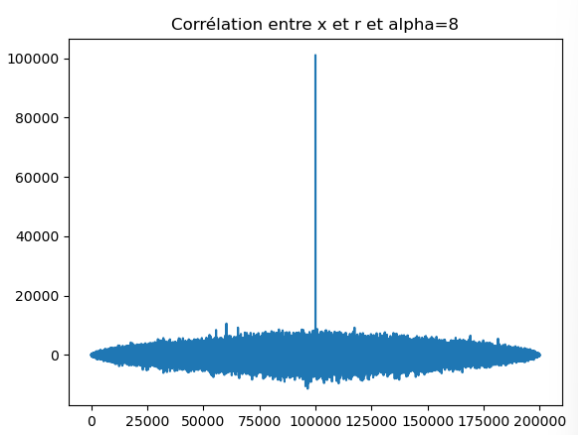


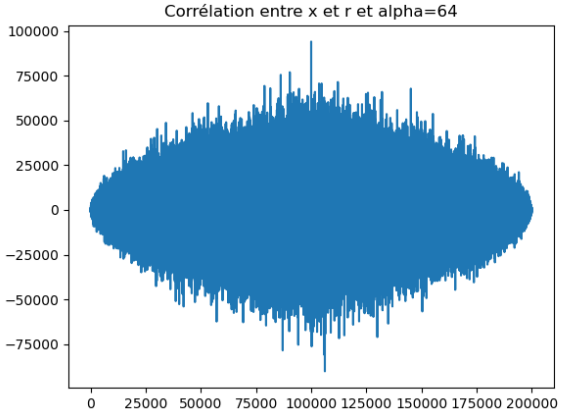


T=









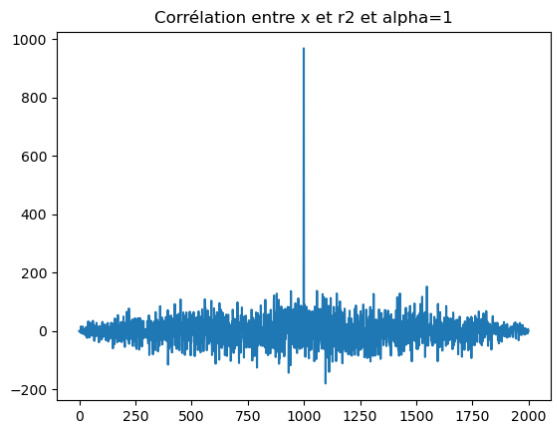
Conclusion de l'exemple:

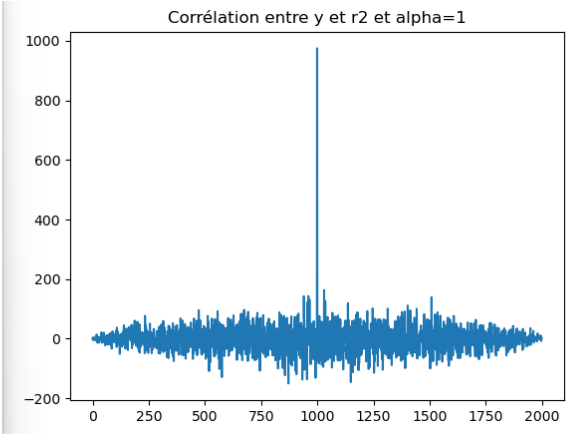
Lorsque l'on s'intéresse à la corrélation pour alpha égal à 1, T égal à 1000 entre x et r, on remarque que l'on a un résultat très intéressant car on a sommé deux vecteurs composés de valeurs aléatoires gaussiennes.
En les additionnant, on a un résultat qui nous montre que x et r ne sont plutôt pas corrélés ce qui peut paraître étonnant.
En augmentant les valeurs d'alpha, on augmente corrélation entre les deux signaux. Pour un alpha constant et augmentant la valeur de T on remarque que l'on s'approche de signaux décorrélés entre x et r. Ces résultats sont cohérents vis-à-vis des propriétés de la fonction de corrélation.

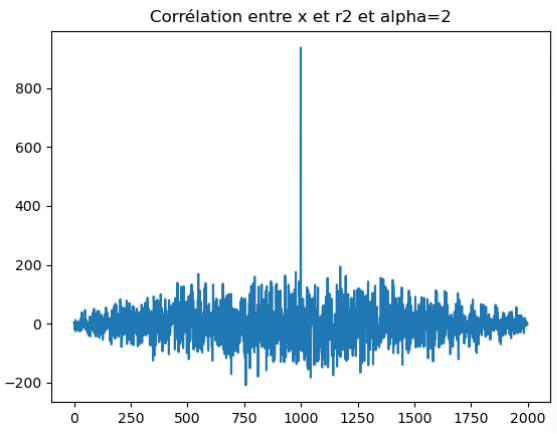
EXEMPLE 9 : Étude de la robustesse face à l'ajout de bruit et à l'interférence entre signaux

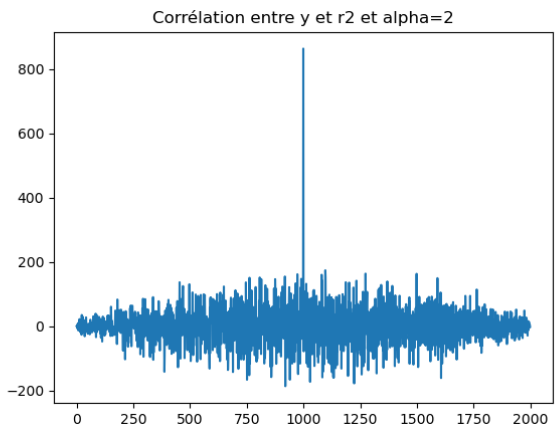
Considérons T = 1000, alpha = 1. Nous générons trois vecteurs x, y et n de T composantes, suivant une loi gaussienne standard. En considérant r = x + y + alpha*n nous allons calculer et représenter les fonctions de cross-corrélation suivantes : entre x et r, et entre y et r.
Nous prendrons ensuite alpha = 2, alpha = 8, alpha = 64 puis T = 10000 et T = 100000.

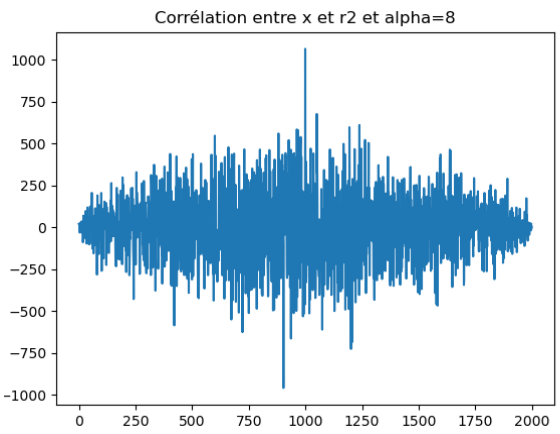
T= 1000:

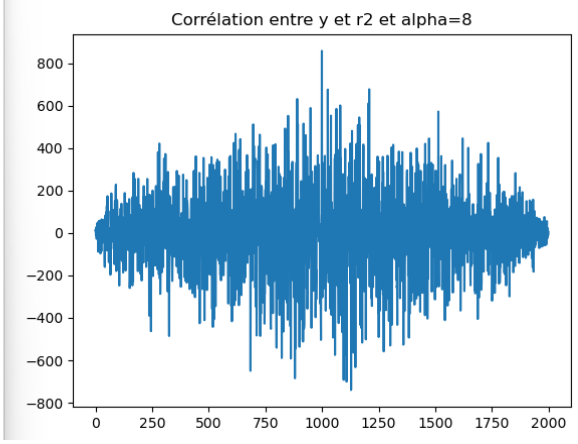


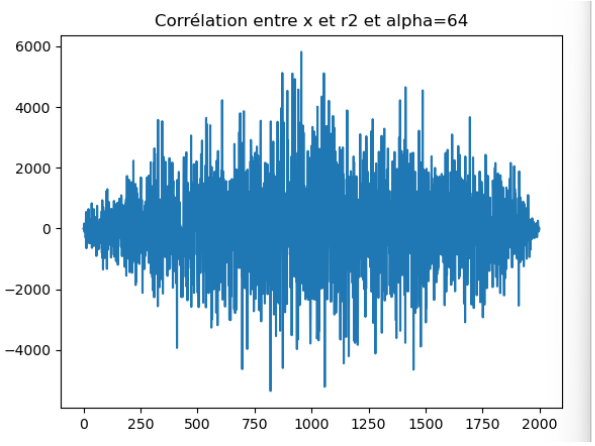


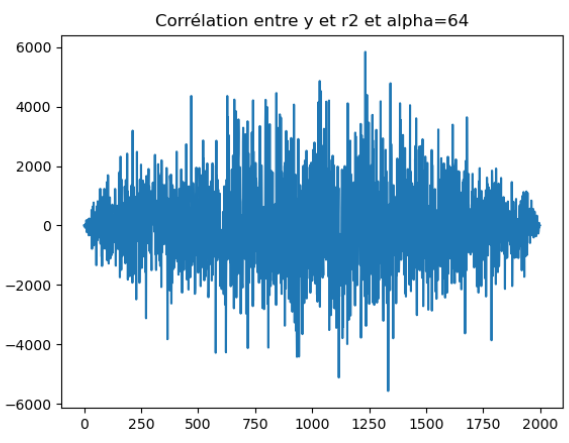




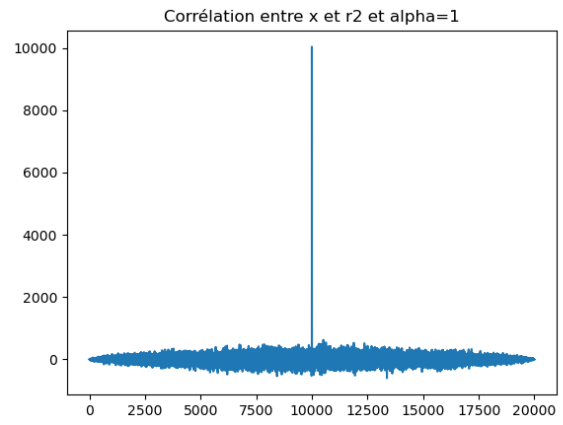


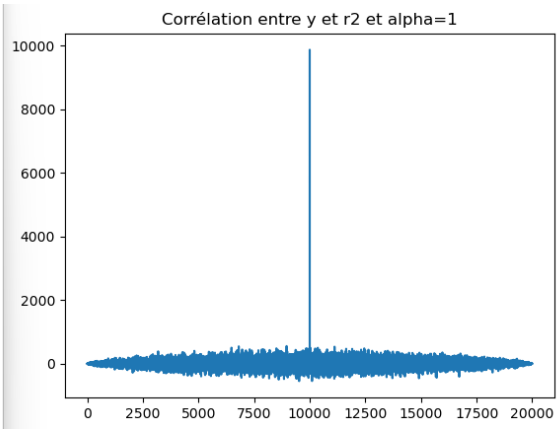


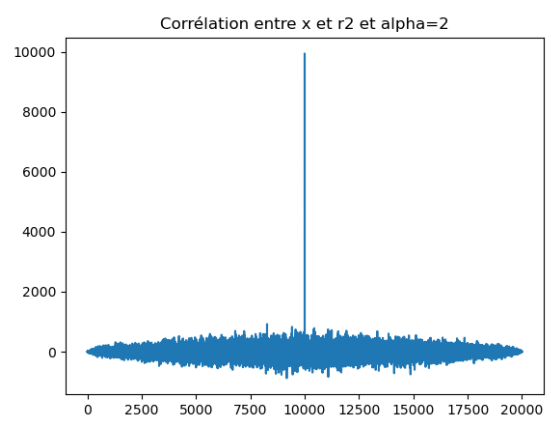

Corrélation entre x et r2 et alpha=64
6000
4000
2000
0
−2000
−4000
0
250
500
750
1000
1250
1500
1750
2000

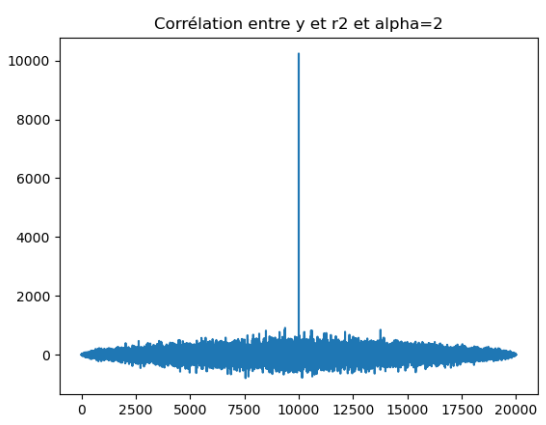

Corrélation entre y et r2 et alpha=64
6000
4000
2000
0
−2000
−4000
−6000
0
250
500
750
1000
1250
1500
1750
2000

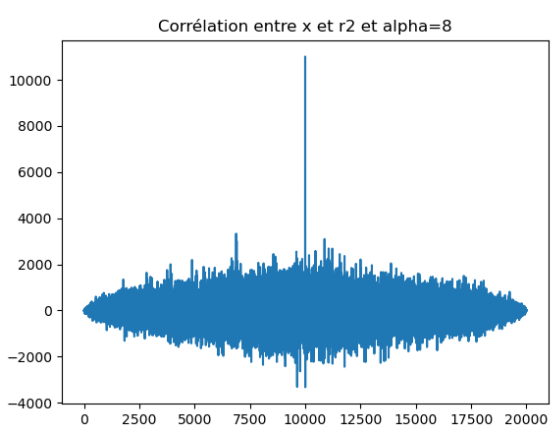
T = 10000

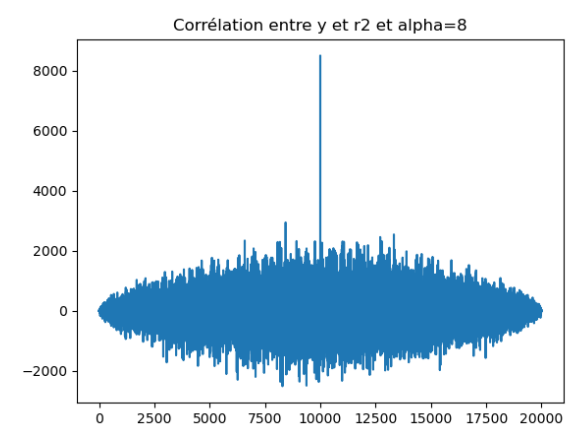

Corrélation entre x et r2 et alpha=1
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
20000

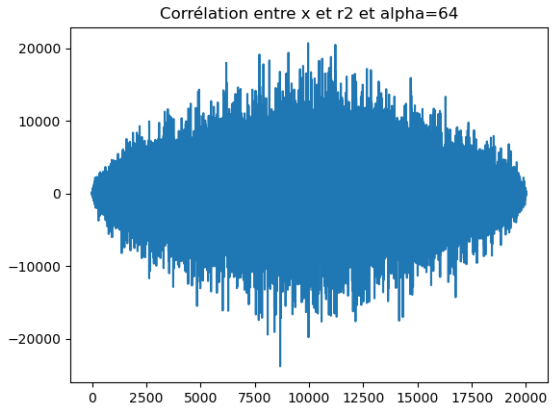

Corrélation entre y et r2 et alpha=1
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
20000

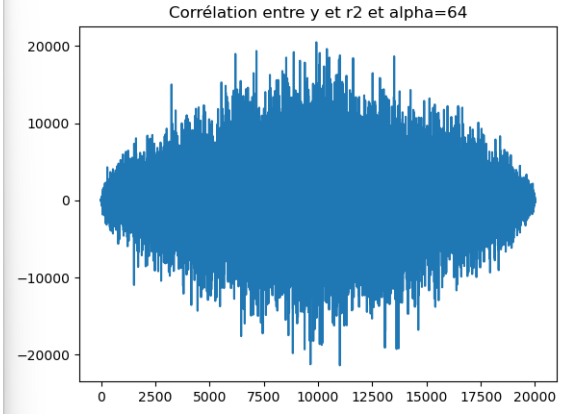

Corrélation entre x et r2 et alpha=2
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
20000


Corrélation entre y et r2 et alpha=2
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
20000

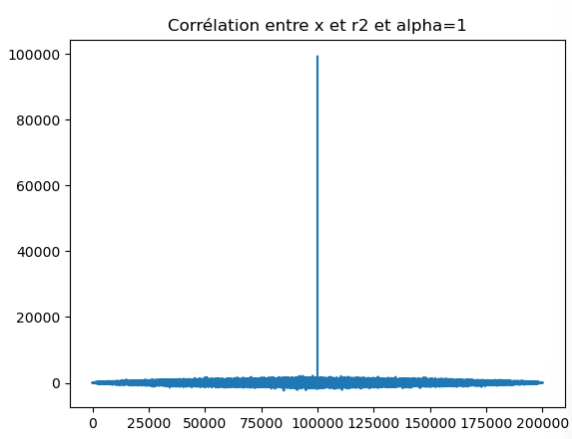

Corrélation entre x et r2 et alpha=8
10000
8000
6000
4000
2000
0
−2000
−4000
0
2500
5000
7500
10000
12500
15000
17500
20000

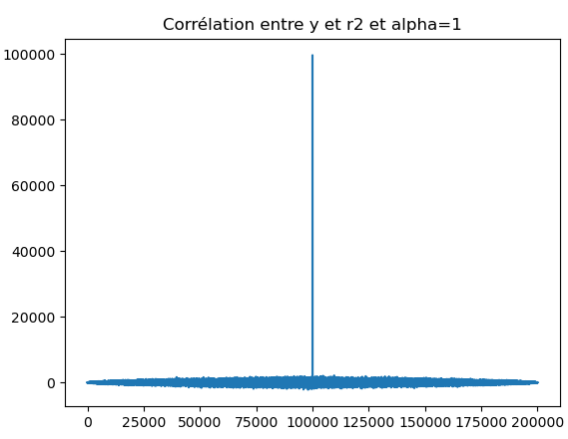

Corrélation entre y et r2 et alpha=8
8000
6000
4000
2000
0
−2000
0
2500
5000
7500
10000
12500
15000
17500
20000

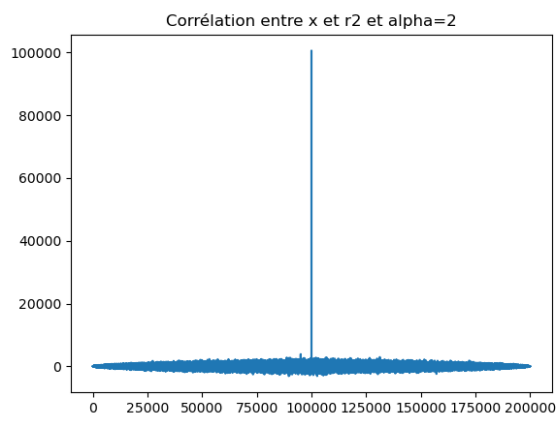

Corrélation entre x et r2 et alpha=64
20000
10000
0
−10000
−20000
0
2500
5000
7500
10000
12500
15000
17500
20000

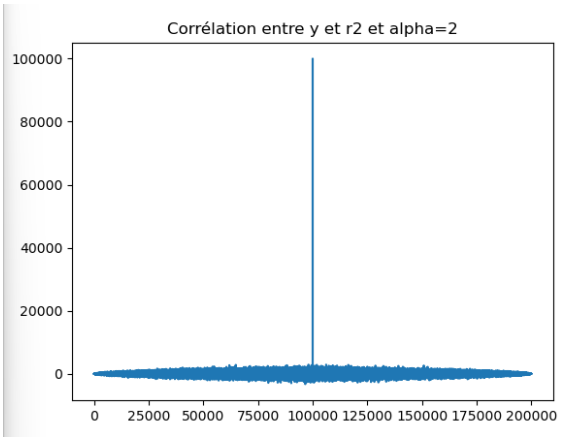

Corrélation entre y et r2 et alpha=64
20000
10000
0
−10000
−20000
0
2500
5000
7500
10000
12500
15000
17500
20000

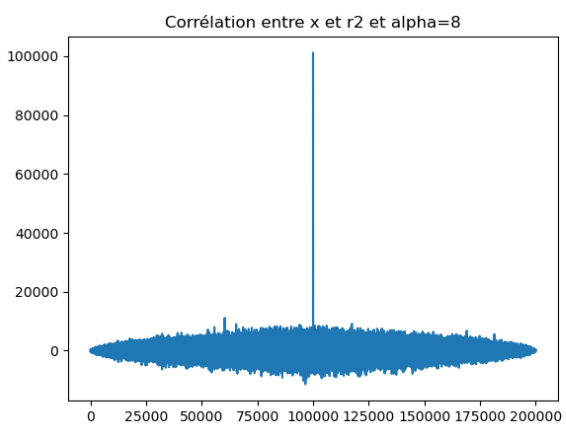
T=100000

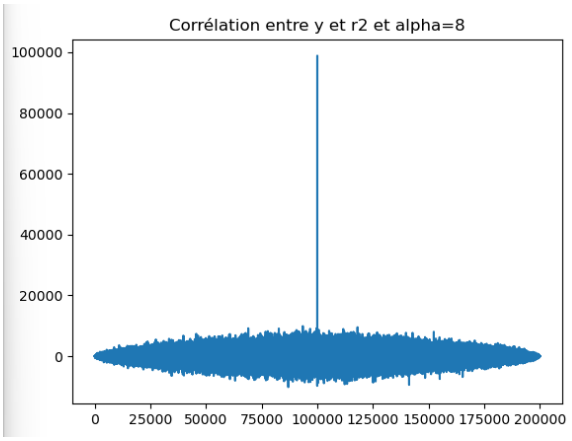

Corrélation entre x et r2 et alpha=1
100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
200000

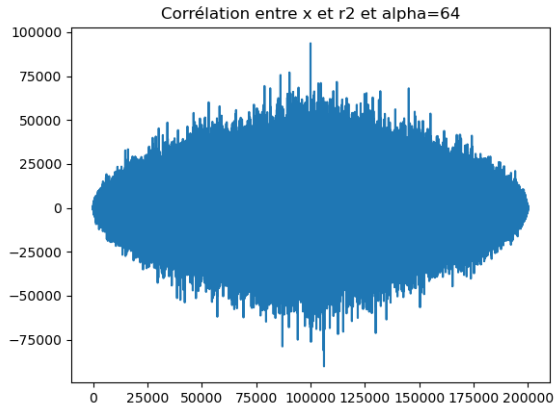

Corrélation entre y et r2 et alpha=1
100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
200000

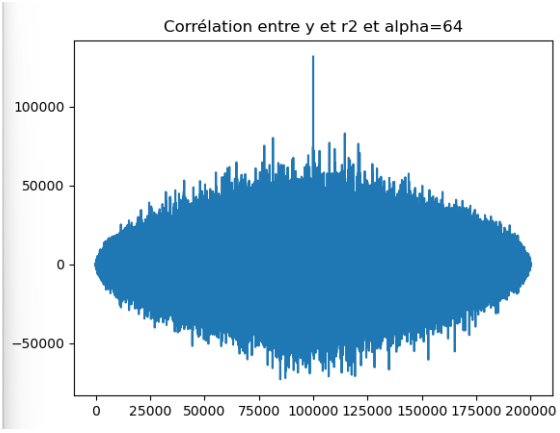

Corrélation entre x et r2 et alpha=2
100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
200000


Corrélation entre y et r2 et alpha=2
100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
200000









Conclusion :

Dans le cas d'étude de alpha = 1, T=1000; on observe un très faible taux de corrélation entre nos signaux que ce soit entre x et r2 et y et r2.

On retrouve un cas similaire à l'expérience précédente: lorsqu'on augmente la valeur de alpha, la corrélation entre les éléments est de plus en plus importante ce qui paraît cohérent car les variations relatives entre les points sera de plus en plus petite.

De même que pour l'exemple précédent, en augmentant T (i.e. le nombre d'échantillon), nous faisons la moyenne de nos cas d'étude.

Pour un grand alpha, on augmente la corrélation entre les points et pour un alpha plus petit, la situation d'étude tend à être décorrélée.

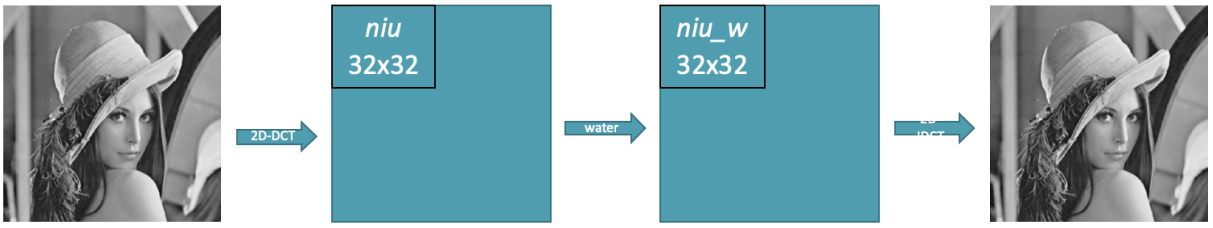
III/ CDMA-based watermarking

EXEMPLE 10.1 : Implémentation d'une méthode basique de CDMA watermarking - Étape d'insertion

Pour commencer, nous appliquons la transformation 2D-DCT (TP2) sur l'image de Lena. La matrice de coefficients DCT de 32x32 en haut à gauche est notée niu.

Nous générons un watermark : une séquence de bruit gaussien standard de 32x32 valeurs ; m(1,1) = 0.

Nous ajoutons la marque selon la formule : niu_water = niu * (1 + alpha * m) où alpha = 1 est un scalaire.
Ensuite nous calculons l'IDCT pour obtenir lena_water ; l'objectif étant de trouver alpha de manière à ce que PSNR (lena, lena_mark) > 30 dB.



lena *lena_dct* *lena_water_dct* *lena_water*

```
def dct2d(image):
    image1= np.float32(image)
    img = cv.dct(image1)
    return(img)

def idct2d(image):
    image1= np.float32(image)
    img = cv.idct(image1)
    return(img)


def noiseadding(im, T, alpha):
    img_mod = im
    mark = normalgenerator(T*T)
    for x in range(0, T):
        for y in range(0, T):
            if (x!=0)&(y!=0):
                img_mod[x][y] = im[x][y]+ alpha*mark[x*T+y]
    return img_mod


def cdmawatermak(im, T, alpha):
    dct_img = dct2d(im)
    dct_mod = noiseadding(dct_img, T, alpha)
    new_img = idct2d(dct_mod)
    return new_img
```

```
def test_alpha():
    for alpha in [0.00, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1]:
        img_test=cdmawatermak(img, 32, alpha)
        psnr = skimage.metrics.peak_signal_noise_ratio(image_true=img, image_test=img_test, data_range=None)
        print(psnr)
        plt.imshow(img_test, cmap='gray')
        plt.show()
```

On obtient un alpha = 128.5. C'est la première valeur telle que PSNR<30.

Exemple 10.2 : Implémentation d'une méthode de tatouage CDMA de base - Étape de détection

Nous disposons maintenant de lena, lena_water et de la matrice m.
Avec la formule de détection nous allons pouvoir étudier la robustesse de la méthode face à la compression jpeg avec des facteurs de qualité Q = 99, Q = 75 et Q = 50.

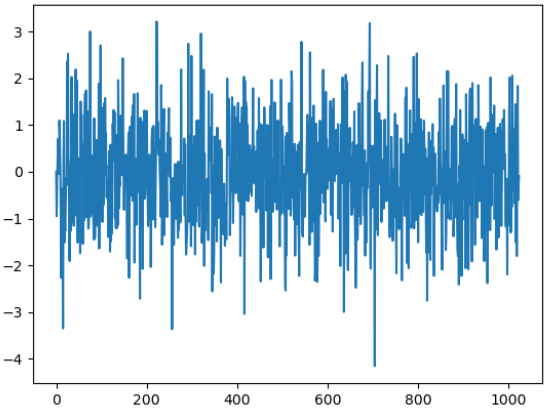
Code pour réaliser l'exercice :

```python
def noiseextract(lena, lena_watermarked, T):
    watermmark_recup = normalgenerator(T*T)
    watermmark_recup[0] = 0
    for x in range(0, T):
        for y in range(0, T):
            if (x!=0)&(y!=0):
                watermmark_recup[x*T+y]=lena_watermarked[x][y]- lena[x][y]
    return watermmark_recup

def detect_sig(im, lena_watermarked, T):
    dct_img = dct2d(im)
    dct_water = dct2d(lena_watermarked)
    dct_mod = noiseextract(dct_img, dct_water, T=T)
    return dct_mod

def enregistrement(im, Q):
    cv.imwrite('C:/Users/andre/OneDrive/Documents/HTI/DCP/lena_'+str(Q)+'.jpg', im, [cv.IMWRITE_JPEG_QUALITY, Q])

def test():
    for Q in [99, 75, 50]:
        res = [0]
        im, mark = cdmawatermak(lena, 32, 1)
        print(mark)
        enregistrement(im, Q)
        img2 = cv.imread('C:/Users/andre/OneDrive/Documents/HTI/DCP/lena_'+str(Q)+'.jpg', cv.IMREAD_GRAYSCALE)
        lena_water = np.array(img2)
        mark_recup = detect_sig(lena, lena_water, 32)
        print(mark_recup)
        for x in range(1, 32*32):
                res.append(mark_recup[x]-mark[x])
        print(res)
        plt.plot(res)
        plt.show()

test()
```

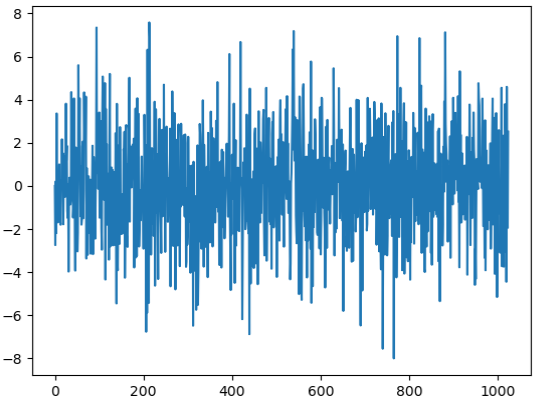
Lena_50.jpg **Lena_75.jpg** **Lena_99.jpg**

Différence entre le watermark initial et récupéré après compression:

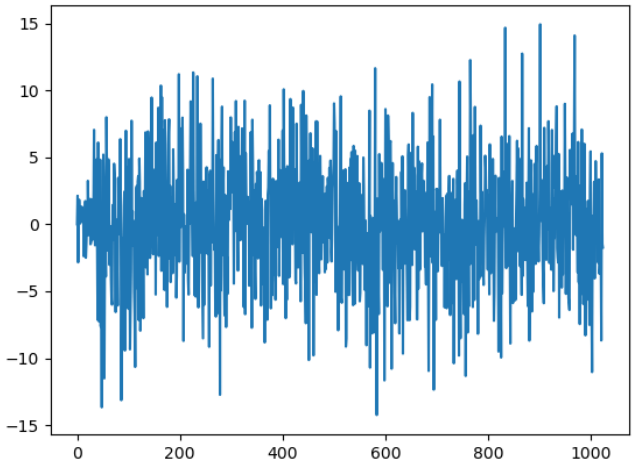
Léna 99 :



Léna 75:



Léna 50:



Conclusion :

Visuellement, notre watermark reste transparent peu importe le niveau de compression qu'on a appliqué à l'image.
On remarque que plus l'image subit un niveau de compression élevé, plus notre watermark est dégradé et les valeurs sont très éloignées de celles qu'il nous faut. Dans le premier cas, notre watermark a des écarts déjà assez importants pour des éléments générés à partir d'une loi gaussienne.