Mini Projet : Détection de changements par Entropies et Vraisemblances

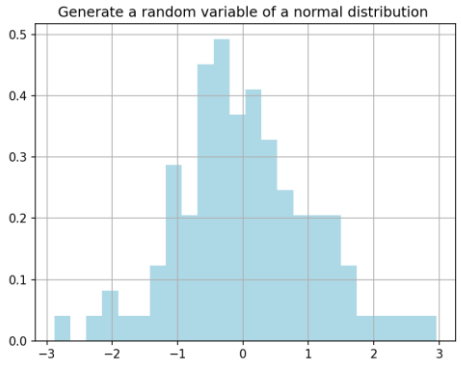
RAFII Ayoub
ELABEDELALOUI Anas
BEKKARI Ahmed

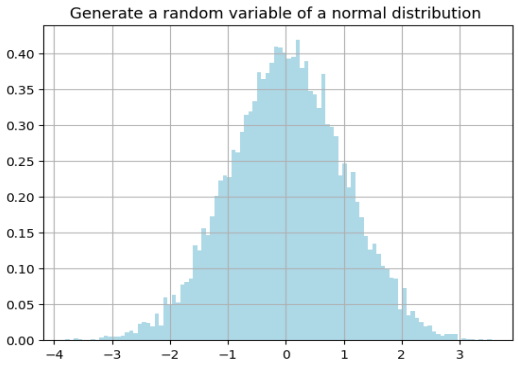
Professeur : Mr A. ATTO
Filière : IDU4

Exercice0 :

Analyse 1 :

- **Génération de N réalisations d'une variable aléatoire suivant une loi normal centré réduite.**





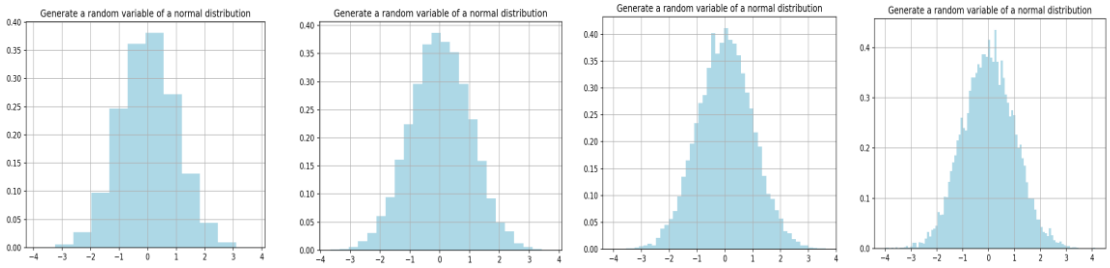
N=100

N=10000

Code source :

```
import numpy as np
import matplotlib.pyplot as plt


mu = 0
sigma = 1.0


data = np.random.normal(0,1,10000)* sigma + mu
count, bins, ignored = plt.hist(data, bins=24, density=True, color="lightblue")


plt.title('Generate a random variable of a normal distribution \n')
plt.grid()
plt.savefig("normal_distribution_10000.png", bbox_inches='tight')
plt.show()
```

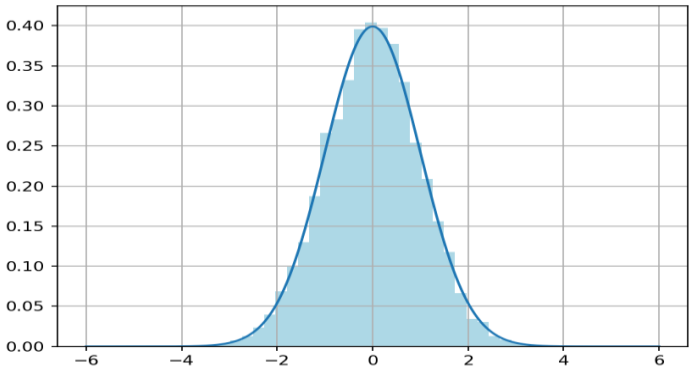
- **Variant maintenant la valeur des bins en prenant b = {12, 24, 50, 100}.**



bins=12 **bins**=24 **bins**=50 **bins**=100

Donc selon ces graphes on déduit qu'en augmentant le nombre de bins et aussi les réalisations la précision augmente et l'histogramme ressemble de plus en plus à une distribution normale.

- **Après normalisation voici le tracé de la courbe de la ddp d'une loi normale.**

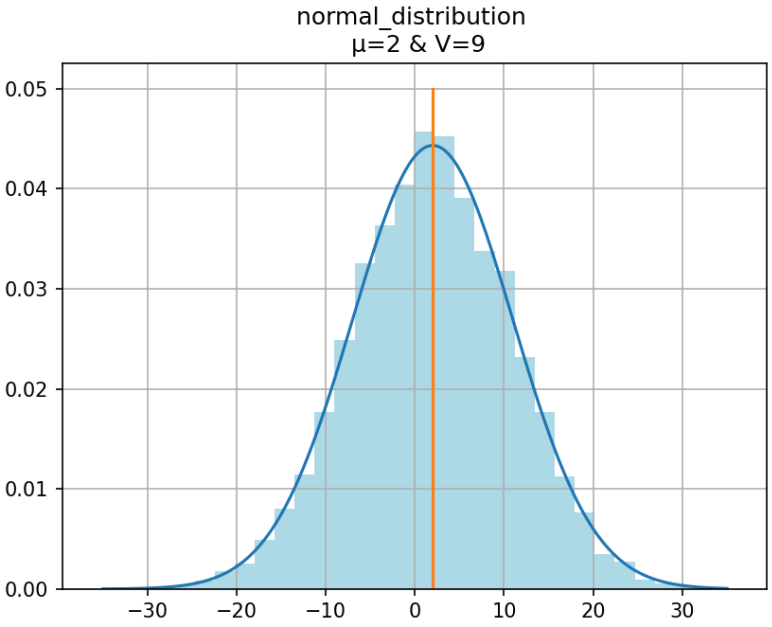


Code source:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats

x = np.linspace(-6.0, 6.0, 1000)
plt.plot(x, scipy.stats.norm.pdf(x,0,1))

plt.grid()
plt.savefig("ddp_normal_distribution.png")
plt.show()
```

Analyse 2 :

Il suffit de changer les valeurs de mu et sigma qui sont surlignés en dessus. On obtient le graphe suivant :

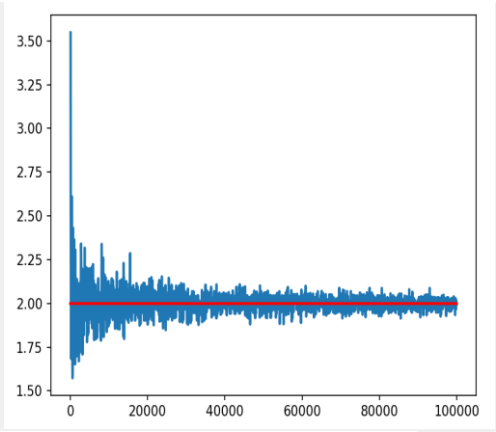


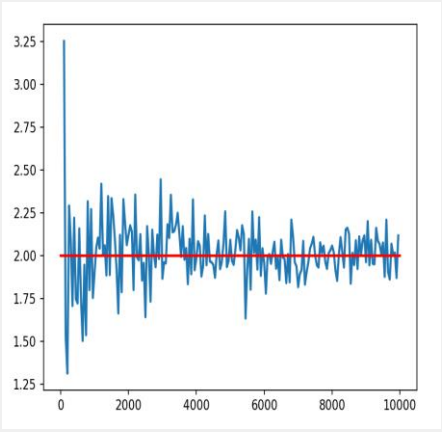
Analyse 3 :

Pour étudier les fluctuations des estimateurs empiriques on a créé deux fonctions qui stockent les variances, les moyennes et traces aussi les courbes associées :

Pour la moyenne :

```
def emperique_mean(min,max,pas):
     X=[]
     Y=[]
     V=[]
     for k in range(min,max,pas):
         m = ( np.random.normal(0, 1, k) *9 + 2).mean()
         X.append(m)
         Y.append(k)
     V.append(X)
     V.append(Y)
     fig, ax = plt.subplots()
     ax.plot(V[1], V[0], linewidth=2.0)
     x = [0, max]
     y = [2, 2]
     ax.plot(x, y, linewidth=2.0, color="r")
     plt.show()
```





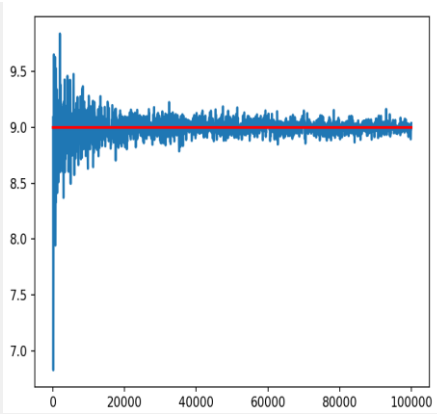
N = 100 000

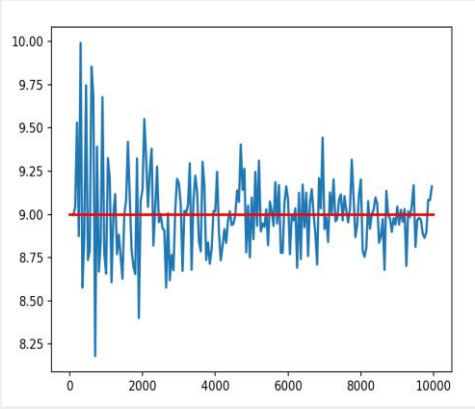
N = 10 000

Donc selon ces graphes on déduit qu'en augmentant le nombre des réalisations et la moyenne tends vers sa valeur réelle.

Pour la variance :

```
def emperique_var(min,max,pas):
    X=[]
    Y=[]
    V=[]
    for k in range(min,max,pas):
        m = (np.random.normal(0, 1, k)*3 + 2).var()
        X.append(m)
        Y.append(k)
    V.append(X)
    V.append(Y)
    fig, ax = plt.subplots()
    ax.plot(V[1], V[0], linewidth=2.0)
    x = [0, max]
    y = [0, 0]
    ax.plot(x, y, linewidth=2.0, color="r")
    plt.show()
```





N = 100

000 N = 10 000

Donc selon ces graphes on déduit qu'en augmentant le nombre des réalisations et la moyenne tends vers sa valeur réelle.

Exercice 1 :

- **Matrice de covariance :**

Y1 = (X1, X2) Y2 = (X2, X3) Y3 = (X3, X1)

```
[[1.05945356 0.03792308]
 [0.03792308 1.04421192]]
```

```
[[1.04421192 0.03195589]
 [0.03195589 1.07320364]]
```

```
[[1.07320364 0.0375571 ]
 [0.0375571  1.05945356]]
```

On observe que c1 égale un peu près c2 et c3. Donc ils on peut dire qu'ils sont corrélés.

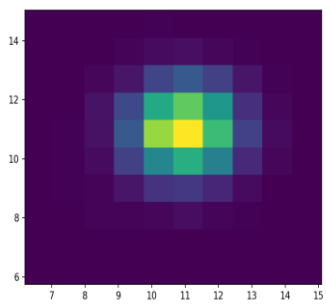
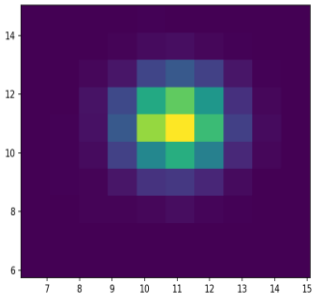
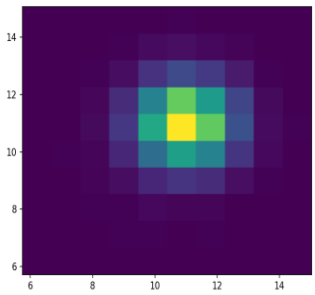
Code Source :

```
mat = scipy.io.loadmat('X_pluv.mat')
#print(mat)
matrix = mat["X_pluv"]


c1=np.cov(matrix[0],matrix[1])
c2=np.cov(matrix[1],matrix[2])
c3=np.cov(matrix[2],matrix[0])
```

- **Histogramme bivarié :**

Nous avons choisi de faire un diagramme en 2D pour que les observations soit plus simple.

Y1 = (X1, X2) Y2 = (X2, X3) Y3 = (X3, X1)

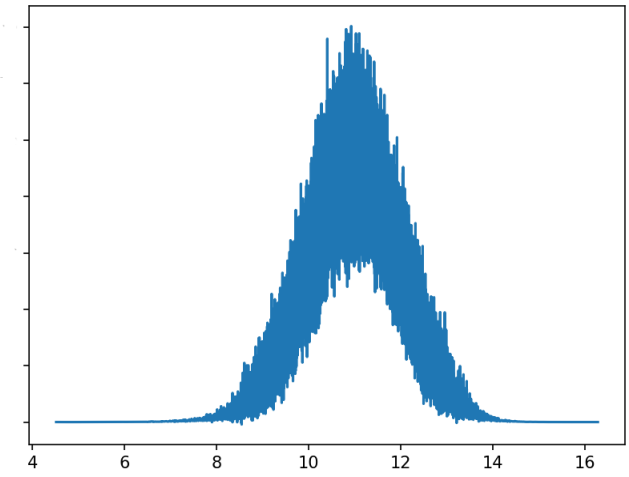




Code Source :

```
plt.hist2d(matrix[0],matrix[1])
plt.hist2d(matrix[1],matrix[2])
plt.hist2d(matrix[2],matrix[0])
```

- **Distribution des données :**

```
def dessiner_ddp_echantillon():
    N = matrix.size
    n = N//10
    p, x = np.histogram(matrix, bins=n) # bin it into n = N//10 bins
    x = x[:-1] + (x[1] - x[0])/2   # convert bin edges to centers
    f = UnivariateSpline(x, p, s=n)
    plt.plot(x, f(x))
    plt.show()
```

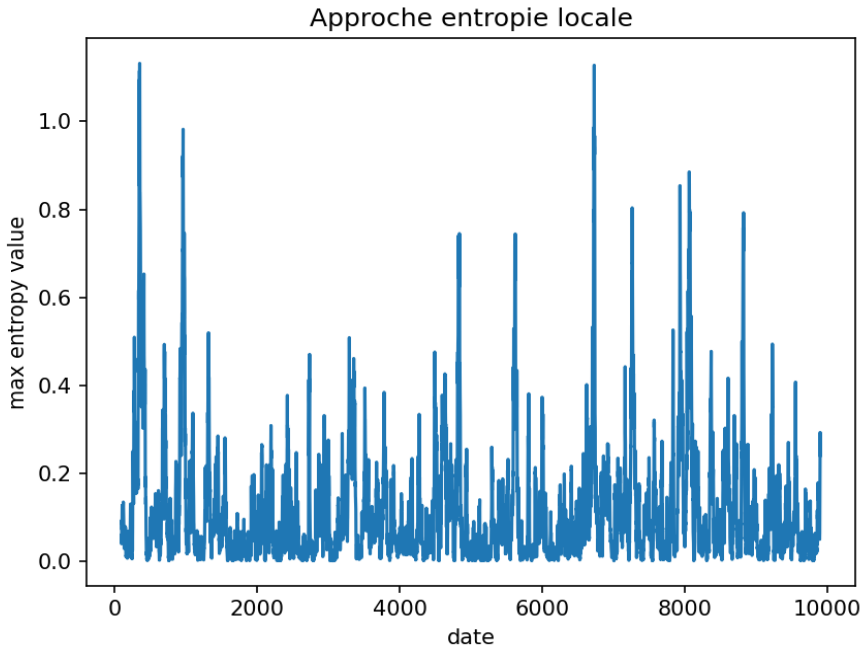


Le graphe résultant resemble à une distribution gausienne donc l'hypothèse de distribution Gaussienne des données elle est satisfaisante globalement.

Exercice 3 :

- Calcul de divergence au niveau local par l'approche de la fenêtre glissante et traçage de courbe et déduction de m* la valeur max de divergence qui représente la date de changement.

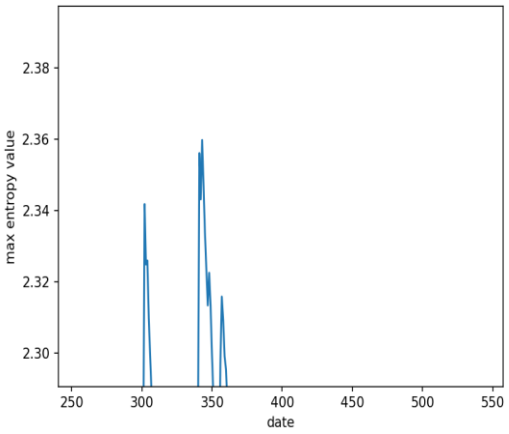
On as procédé de la même manière que l'exo 2 voici les fonctions qui nous'ont donné ce résultat là pour la ville 1 :




```
mat = scipy.io.loadmat('X_pluv.mat')
matrix = mat["X_pluv"]


def demo_entropie_locale():
    x = entropy_local(100, matrix[1])
    y = np.arange(100, x.size + 100)


    plt.title("Approche entropie locale")
    plt.xlabel("date")
    plt.ylabel("max entropy value")
    plt.plot(y, x)
    plt.show()



def entropy_local(r, matrix) :
    l=matrix.size
    entropy_matrix= []
    values=[]
    for k in range(r,l-r):
        U1 = matrix[k-r:k-1]
        U2 = matrix[k+1:k+r]
        m1 = U1.mean()
        m2 = U2.mean()
        S1 = np.var(U1)
        S2 = np.var(U2)
        entropy = 0.5*(m1-m2)**2 * (1/S1**2 + 1/S2**2) + 0.5 * ((S1 / S2)** 2 + (S2 / S1)**2)
-1
        values.append(k)
        entropy_matrix.append(entropy)


    return np.array(entropy_matrix)
```

Exercice 2 :

- Calcul de divergence au niveau global et traçage de courbe et déduction de m* la valeur max de divergence qui représente la date de change



Ville 1

D'après ces 2 graphes on peut déduire qu'il y'a un changement au niveau de la ville 1 vers l'instant 345.

Code Source :

```
def entropy_mono(n, matrix) :
    l=matrix.size
    entropy_matrix= []
    values=[]
    for k in range(n-1,l-n-2):
        U1 = matrix[0:k-1]
        U2 = matrix[k+1 :l]
        m1 = U1.mean()
        m2 = U2.mean()
        S1 = np.var(U1)
        S2 = np.var(U2)
        entropy = 0.5*(m1-m2)**2 * (1/S1**2 + 1/S2**2) + 0.5 * ((S1 / S2)** 2 + (S2 / S1)**2)
-1
        values.append(k)
        entropy_matrix.append(entropy)

    return np.array(entropy_matrix)
```

Puis pour dessiner la courbe :

```
x = entropy_mono(100, matrix[1])
y = np.arange(100, x.size+100)

plt.title("")
plt.xlabel("date")
plt.ylabel("max entropy value")
plt.plot(y,x)
plt.show()
```

Pour faire des analyses pour toutes les villes on a créé ces 2 fonctions :

```
Retourner la valeur de divergence maximal par ville et sa date

def max_entropy_mono(n, matrix):
    l=matrix.size
i=n
```

```
    entropy_matrix= []
    max = 0
    for k in range(n,l-n):
        U1= matrix[0:k-1]
        U2 = matrix[k-1:l-n]
        m1 = U1.mean()
        m2 = U2.mean()
        S1 = np.var(U1)
        S2 = np.var(U2)
        entropy = 0.5*(m1-m2)**2 * (1/S1**2 + 1/S2**2) + 0.5 * ((S1 / S2)** 2 + (S2 / S1)**2)
-1
     i+=1
        if max<entropy:
            max =entropy
    return max , i
```

Retourner un objet ndarray contenant les valeurs max et leurs dates

```
def max_entropy_by_city(matrix):
    l=[]
    n =matrix.size
    for k in range(n):
       x=entropy_mono(10, matrix[k])
       m = np.argmax(x)
       print(m)
       l.append(m)
    return np.array(entropy_matrix)
```

L'approche de l'exo 3 est plus exacte et précise puisqu'il y'a moins d'effets de bords et les intervalles de calcul sont égaux pour chaque instant.

Exercice 4 :

Pour pouvoir analyser et détecter le changement il fallait d'abord calculer la fonction de vraisemblance approché :

$$\mathcal{L}_{[t_m,t_n]}\left(\hat{\mu},\hat{\Sigma}\right) = \left|\hat{\Sigma}_{[m,n]}\right|^{-N/2}(2\pi)^{-NK/2}\exp\left\{-\frac{1}{2}\sum_{m\leq i\leq n}\left(\mathbf{x}_i-\hat{\mu}_{[m,n]}\right)'\hat{\Sigma}^{-1}_{[m,n]}\left(\mathbf{x}_i-\hat{\mu}_{[m,n]}\right)\right\}$$

Pour cela on a créé la fonction :

```
def L_log(matrix,tm,tn):
    K=int(matrix.size/matrix[0].size)
    ecov = empirique_covariance(matrix,tm,tn)
    N=tn-tm
    denominator = la.norm(ecov) ** (-N)
    exposant = np.empty(K)
    for tk in range(tn,tm):
        A = to_column(matrix, 7) - empirique_mean(matrix, 2, 9)
        B = np.linalg.inv(np.linalg.inv(empirique_covariance(matrix, 2, 9)))
        C = np.diag(A)
        exposant+= 0.5* np.dot(np.dot(B, C), A)


    return np.log(denominator) + exposant
```

Puis:

```
def L_global(matrix,t0,tf):
    N=(tf-t0-4)
    result = np.empty(N)
    L=0
    # Creer une matrice colonne qui contient les normes de chaque matrice colonne calculé
```

```
    # Chaque matrice colonne représente la matrice de vraisamblance eu point m
    for k in range(t0+2,tf-2):
        L= L_log(matrix, t0, k) + L_log(matrix, k+1, tf)
        result[k-t0-2]=la.norm(L)

    return result
```

Et pour calculer ça il fallait calculer la moyenne empirique et la covariance empirique selon les formules :

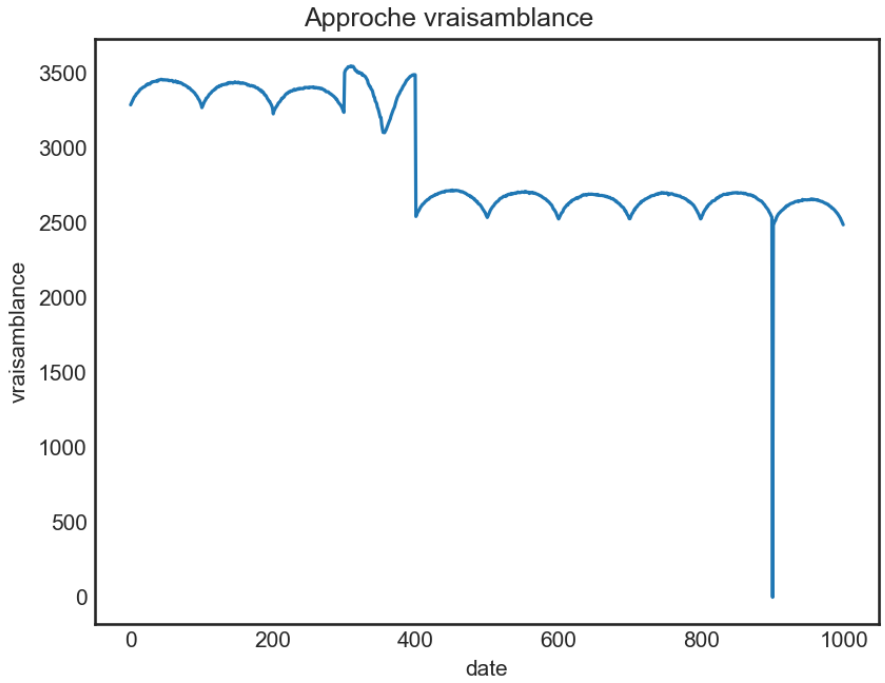
+ Moyenne empirique : $\hat{\mu}_{[m,n]} = \frac{1}{N}\sum_{i=m}^{n} \mathbf{x}_i$

+ Covariance empirique $\hat{\Sigma}_{[m,n]} = \frac{1}{N}\sum_{i=m}^{n}(\mathbf{x}_i - \hat{\mu}_{[m,n]})(\mathbf{x}_i - \hat{\mu}_{[m,n]})'$

Pour cela j'ai créé les deux fonctions :

```
def empirique_mean(matrix,t0,tf):
    n = int(matrix.size / matrix[0].size)
    result = np.zeros(n)
    N=tf-t0
    for k in range(t0,tf):
        result += to_column(matrix,k)
    return result/N



def empirique_covariance(matrix,t0,tf):
    n = int(matrix.size / matrix[0].size)
    result = np.zeros([n,n])
    N = tf - t0
    emean = empirique_mean(matrix, t0, tf)
    for k in range(t0, tf):
        result+= np.outer((to_column(matrix,k) - emean) ,(to_column(matrix,k) - emean))
    return result/N
```

En dessinant les résultats sous forme d'une courbe on obtient :



Pour cette question on a trouvé quelque difficulté puisque le résultat de multiplication des matrices donne de grands nombres qui donne des résultat nan et +/-inf.

Aussi le temps d'exécution est lent et parfois on a des erreurs quand on utilise les fonctions sur l'intervalle [0,10000]. Donc on a utilisé les fonctions pour des intervalles de taille 1000 mesures maximum et on a combiné les résultats de ces intervalles.

Exercice 5 :

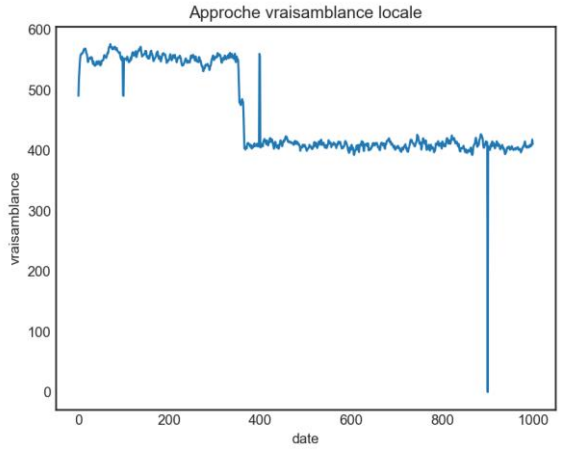
De même manière et en utilisant les fonctions de l'exo4 sous l'approche de la fenêtre glissante :

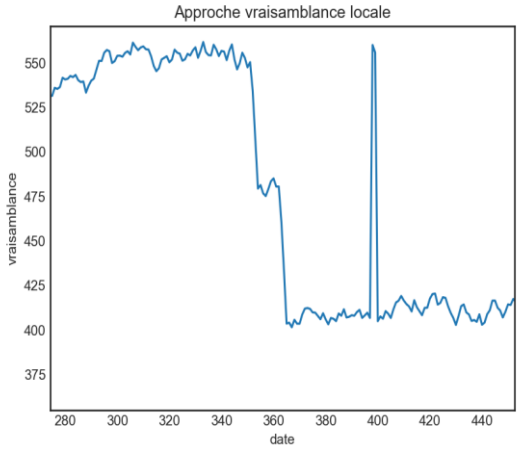
$$\mathcal{L}_p^{[m]}\left(\widehat{\boldsymbol{\mu}},\widehat{\boldsymbol{\Sigma}}\right) = \mathcal{L}_{[\![t_{m-p},t_{m-1}]\!]}\left(\widehat{\boldsymbol{\mu}},\widehat{\boldsymbol{\Sigma}}\right) + \mathcal{L}_{[\![t_{m+1},t_{m+p}]\!]}\left(\widehat{\boldsymbol{\mu}},\widehat{\boldsymbol{\Sigma}}\right)$$

J'ai créé les fonctions suivantes :

```
def L_local(matrix,p,t0,tf):
    result=np.empty(tf-t0)
    for tm in range (t0,tf):
        L = L_log(matrix, tm-p, tm-1) + L_log(matrix, tm+1, tm+p)


        result[tm - t0 - 2] = la.norm(L)
    return result


def test_exo5():
    result = np.zeros(1000)
    for k in range(9):
        p = k * 100
        result[p:100 + p] = L_local(matrix, 10 ,p, 100 + p)
    result[901:1000] = L_local(matrix, 10 ,901, 1000)
    x = result
    y = np.arange(0, x.size)
    plt.title("Approche vraisamblance locale")
    plt.xlabel("date")
    plt.ylabel("vraisamblance")
    plt.plot(y, x)
    plt.show()
```

Le résultat en zoomant est :





Exercice 6 :

Pour cette question on doit faire une analyse locale par fusion de maxima de vraisemblances

Calcul de vraisemblance en monovariée selon les formules de exo4

```
def L_log_city(city,tm,tn):
    city = city[tm:tn]
    mean = city.mean()
    var = city.var()
    N=tn-tm
    denominator = var ** (-N)
    exposant = 0
    for tk in range(tn,tm):
        exposant+= (0.5*(city[tk] - mean)**2)/var
    # En analyse monovarié on a une seule ville dpnc K=1
    K=1
    return np.log(denominator) + np.log(2*np.pi) + N + K + exposant

def L_global_city(city,t0,tf):
    N=(tf-t0-4)
    result = np.empty(N)
    for k in range(t0+2,tf-2):
        result[k-t0-2]= L_log_city(city, t0, k) + L_log_city(city, k+1, tf)

    return result
```

Calcul du maximum de vraisemblance et sa date pour une ville donné

```
def vraissamblance_max(city,t0,tf):
    result=np.zeros(2)
    l=0
    date=0
    matrix=L_global_city(city,t0,tf)
    n = len(matrix)
    for k in range(n):
        if l<matrix[k]:
            l=matrix[k]
            date=k
    result[0]=l
```

```
    result[1]=int(date)
    return result
```

Enfin calcul de la matrice de fusion de max de vraisemblance :

```
def fusion_vraissamblance(matrix):
    L=np.zeros(len(matrix))
    i=0
    for city in matrix:
        L[i]=vraissamblance_max(city,10,1000)[1]
        i+=1
    return L
```

On obtient alors les dates de changement pour les 100 villes de l'échantillon :

```
[341. 342. 335. 336. 335. 346. 344. 342. 342. 344. 360. 343. 342. 341.
 344. 342. 343. 356. 344. 344. 353. 342. 341. 353. 342. 342. 336. 341.
 347. 344. 343. 361. 343. 346. 347. 342. 338. 341. 343. 344. 344. 343.
 342. 343. 346. 335. 334. 355. 341. 343. 343. 345. 323. 343. 344. 327.
 330. 309. 333. 342. 342. 321. 343. 344. 343. 330. 342. 341. 346. 339.
 341. 324. 352. 344. 341. 337. 337. 335. 347. 343. 330. 334. 344. 339.
 345. 335. 344. 341. 346. 343. 323. 341. 339. 334. 333. 343. 341. 342.
 340. 343.]
```

Donc la moyenne et la variance sont :

```
Moyenne  de la date de changement :  340.92
 Variance de la date de changement :  55.293600000000005
```

Problème :

Dans un premier temps, nous avons traité ce sujet en croyant qu'il y 'avait une image car en exécutant :

```
print("la taille de notre image est : ", mat['Pixel'].size,"*",
mat['Pixel'][0].size )
```

Et on a obtenu le résultat suivant :

```
la taille de notre image est:  768 * 256

Process finished with exit code 0
```

Mais après on a compris qu'il s'agit plutôt de l'intensité de 3 pixels mesuré en plusieurs instant (256 instants). Nous ne savons pas trop si c'était ça ce que représente les données ou pas puisque l'énoncé est vague et même les valeurs de l'intensité des 3 pixel n'était pas usuelles.

Nous avons choisi d'utiliser la fusion de max de vraisemblances pour détecter le changement donc nous avons créé les fonctions suivantes :

```
def L_log_picture(picture,tm,tn):
    picture = picture[tm:tn]
    mean = picture.mean()
    var = picture.var()
    N=tn-tm
    denominator = var ** (-N)
    exposant = 0
    for tk in range(tn,tm):
        exposant+= (0.5*(picture[tk] - mean)**2)/var
    K=1
    return np.log(denominator) + np.log(2*np.pi) + N + K + exposant

#print(L_log_picture(matrix[0],0,256))

def L_global_picture(picture,t0,tf):
    N=(tf-t0-4)
    result = np.empty(N)
    for k in range(t0+2,tf-2):
        result[k-t0-2]= L_log_picture(picture, t0, k) + L_log_picture(picture, k+1, tf)

    return result
#print(L_global_picture(matrix[0],0,256))
```

Nous avons ensuite utilisé le max de vraisemblance en s'appuyant sur les formules des exercices précédents :

```
def vraissamblance_max(picture,t0,tf):
    result=np.zeros(2)
    l=0
    date=0
    matrix=L_global_picture(picture,t0,tf)
    n = len(matrix)
    for k in range(n):
        if l<matrix[k]:
            l=matrix[k]
            date=k
    result[0]=l
    result[1]=int(date)
    return result

#print(vraissamblance_max(matrix[0],0,256))
```

Puis la fusion de max de vraisemblance :

```
def fusion_vraissamblance(matrix):
    L=np.zeros(len(matrix))
    i=0
    for picture in matrix:
        L[i]=vraissamblance_max(picture,0,256)[1]
        i+=1
    return L

#print(fusion_vraissamblance(matrix))
```

Puis en exécutant tout ça nous avons conclus que les 3 changements sont effectués à l'instant 125

```
[125. 125. 125.]
```