Nom: Iordache Paul-Tiberiu, Soares Robin

Groupe: 21.4 A

Manuel d’utilisation

Durant notre deuxième semestre, nous avons pu en Informatique et topologie des sciences produire un algorithme python. Nous avons commencé avec l’aide du professeur par faire un programme qui affichait une gaussienne en 1 dimension. Puis nous avons dû faire un programme qui affiche en vue du dessus (au moins) une gaussienne en 2 dimensions. Ce programme affiche aussi les extremums locaux des gaussiennes prises en paramètres. Notre programme peut aussi s'avérer utile en électronique par exemple. En effet, nous pouvons, via notre programme, visualiser des faisceaux gaussiens de la transformée de Fourier. Voici notre manuel d’utilisation :

1) Nous avons importé la bibliothèque “math” pour les calculs comme des racines avec la fonction “sqrt()” et la bibliothèque matplotlib.pyplot pour tracer nos gaussiennes dans un graphique (ligne 1).
Nous avons décidé de définir les bornes de la figures avec xmin et ymin à -5 et xmax et ymax à 5. Nous avons défini le pas comme exigé à 0.1. nbx et nby sont les nombres de x et de y qui seront sur notre graphique.

```
from math import *
import matplotlib.pyplot as plt

# defintion des bornes de la figures
xmin = -5
xmax = 5
ymin = -5
ymax = 5
pas = 0.1
nbx = int((xmax-xmin)/pas)
nby = int((ymax-ymin)/pas)
```

2) Le programme débute en imprimant dans le terminal les données des bornes du graphique. Puis il demande le nombre de gaussiennes à l’utilisateur (ligne 16). Nous pouvons afficher des gaussiennes à l’infini mais nous avons préféré limiter l’utilisateur à 5 gaussiennes maximum. Il lui demande ensuite les paramètres qu’il veut utiliser pour ses gaussiennes. Par rapport au nombre de gaussiennes “nb” introduites par l'utilisateur, nous avons utilisé des structures conditionnelles “if”. Nous avons utilisé la fonction “input” pour que l’utilisateur saisisse les valeurs des gaussiennes.

```python
print("Les bornes vont de -5 à 5 en x et de -5 à 5 en y.")
nb = int(input("Veuillez donner le nombre de gaussiennes que vous voulez afficher (Nous n'en affichons pas plus de 5): "))
if nb > 0:
    sigma1 = float(input("Veuillez saisir les paramètres que vous utiliserez pour la première gaussienne : \nsigma n°1: "))
    sigma2 = float(input("\nsigma n°2: "))
    m1 = float(input("\nespérence n°1: "))
    m2 = float(input("\nespérence n°2: "))
    if nb > 1:
        sigma3 = float(input("Veuillez saisir les paramètres que vous utiliserez pour la deuxième gaussienne : \nsigma n°1: "))
        sigma4 = float(input("\nsigma n°2: "))
        m3 = float(input("\nespérence n°1: "))
        m4 = float(input("\nespérence n°2: "))
        if nb > 2:
            sigma5 = float(input("Veuillez saisir les paramètres que vous utiliserez pour la troisième gaussienne : \nsigma n°1: "))
            sigma6 = float(input("\nsigma n°2: "))
            m5 = float(input("\nespérence n°1: "))
            m6 = float(input("\nespérence n°2: "))
            if nb > 3:
                sigma7 = float(input("Veuillez saisir les paramètres que vous utiliserez pour la quatrième gaussienne : \nsigma n°1: "))
                sigma8 = float(input("\nsigma n°2: "))
                m7 = float(input("\nespérence n°1: "))
                m8 = float(input("\nespérence n°2: "))
                if nb > 4:
                    sigma9 = float(input("Veuillez saisir les paramètres que vous utiliserez pour la cinquième gaussienne : \nsigma n°1: "))
                    sigma10 = float(input("\nsigma n°2: "))
                    m9 = float(input("\nespérence n°1: "))
                    m10 = float(input("\nespérence n°2: "))
                    if nb > 5:
                        print("Nous ne pouvons pas afficher plus de 5 gaussiennes.")
```

3) Puis, nous avons créé une fonction f(x,y), avec laquelle nous pouvons calculer une gaussienne 2D (ligne 43). Nous avons simplifié la loi normale standard à l'aide des variables flottantes a, c, d pour arriver à une formule plus simple. Nous avons utilisé la même fonction pour les 5 gaussiennes f,g,h,k,l.

```python
def f(x,y):
    a = 1 / (sigma1*sigma2 *2*pi)
    c = ((x-m1)/sigma1)**2
    d = ((y-m2)/sigma2)**2
    return (a * exp((-1/2)*(c + d)))
```

4) Pour la suite, nous avons effectué une fonction gaussiennes(x,y) qui effectue la somme du nombre de gaussiennes qui ont été demandées à l'utilisateur. Nous avons utilisé des alternatives "if" pour savoir le nombre de gaussiennes à afficher.

```python
def gaussiennes(x,y):
    if nb >= 5:
        return f(x,y) + g(x,y) + h(x,y) + k(x,y) + l(x,y)
    elif nb == 4:
        return f(x,y) + g(x,y) + h(x,y) + k(x,y)
    elif nb == 3:
        return f(x,y) + g(x,y) + h(x,y)
    elif nb == 2:
        return f(x,y) + g(x,y)
    elif nb == 1:
        return f(x,y)
    else:
        return 0
```

5) Ensuite, nous avons initialisé Z et Zbas, deux listes vides qui contiennent respectivement les valeurs de la fonction gaussiennes(x,y), et une attribution en entier en fonction des valeurs des gaussiennes. Nous avons aussi créé deux autres listes vides: Liste_x et Liste_y . Ces deux listes vont nous servir pour trouver les extremums locaux des gaussiennes.

```
Z = []
Zbas = []
Liste_x = []
Liste_y = []
```

6) Après avoir initialisé les listes, nous allons pouvoir les utiliser. A la ligne 97, nous commençons par une boucle for qui démarre avec i = 0 et s'arrête lorsque i = nbx +1. Cette première boucle “for” va servir à prendre tous les x avec un pas de 0,1. Dans cette boucle, nous initialisons deux listes L et Lbas qui, à chaque i ++, se videront. Nous créons une deuxième boucle “for” (ligne 101), qui va démarrer avec j = 0 et terminer avec j = nby + 1. Cette seconde boucle va nous servir à prendre chaque y du graphique. Après avoir initialisé x et y, nous initialisons w qui va faire appel à la fonction gaussiennes() vue plus tôt. La valeur prise par w va être directement insérée dans la liste L. Pour trouver les extremas locaux de chaque gaussienne, nous avons décidé d’utiliser la structure conditionnelle alternative “if”. Cette condition va chercher s’il existe une valeur plus élevée que w autour de son x et de son y. Si oui, le programme passe à la suite. Sinon, il ajoute dans des listes les valeurs x et y pour lesquelles w = gaussiennes(x,y) est un extrema local.

PS : Nous avons Liste_y qui ajoute x car notre valeur x est initialisée en fonction de i qui calcule les lignes. Pareil pour y, il est initialisé en fonction de j qui calcule les colonnes du graphiques. Dans notre liste, nous avons [[y1, y2, y3...],[y1,y2,y3...],...]. Donc nous stockons les (colonnes) valeurs des y pour une valeur de x puis ainsi de suite.

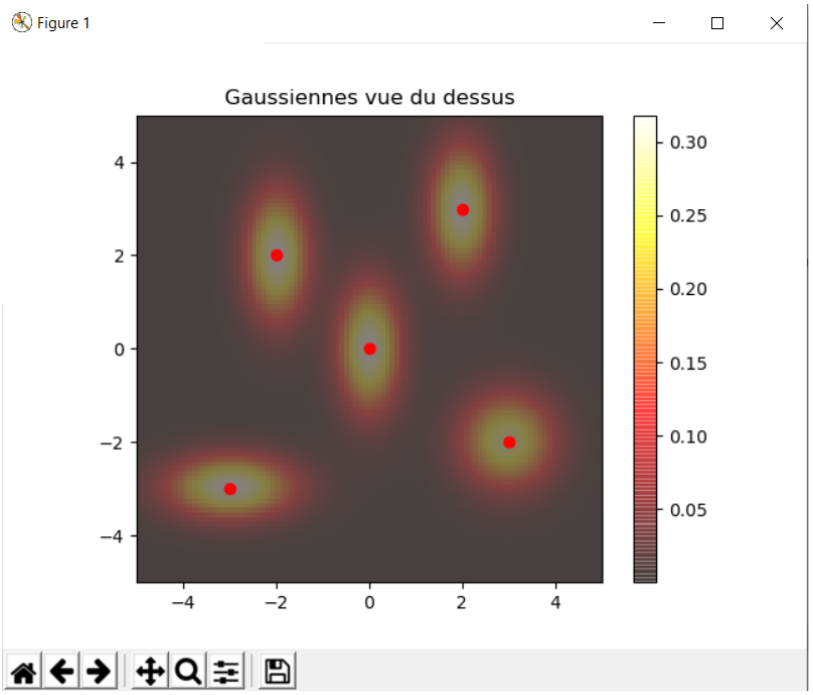
Pour la suite, le programme ajoute 1 à la liste Lbas. La boucle for tourne jusqu’à ce que j soit égal à nbx+1, puis le programme ajoute L à la liste Z et Lbas à la liste Zbas.
Z est une liste de liste (ou matrice) qui contient les valeurs de la fonction gaussiennes(x,y). C’est via cette matrice que nous allons pouvoir afficher nos gaussiennes. Zbas est aussi une matrice qui contient des listes de listes de 1. La matrice Lbas va nous servir à afficher la couleur du fond de notre graphique.

```
Z = []
Zbas = []
Liste_x = []
Liste_y = []
for i in range(nbx+1):
    y = ymin + i*pas
    L = []
    Lbas = []
    for j in range(nby+1):
        x = xmin + j*pas
        w = gaussiennes(x,y)
        if (w > gaussiennes(x-pas,y) and w > gaussiennes(x+pas,y) and w > gaussiennes(x,y-pas) and w > gaussiennes(x-pas,y-pas) and
            w > gaussiennes(x+pas,y-pas) and w > gaussiennes(x-pas,y+pas) and w > gaussiennes(x,y+pas) and w > gaussiennes(x+pas,y+pas)):
            Liste_y.append(y)
            Liste_x.append(x)
        L.append(w)
        Lbas.append(1)
    Z.append(L)
    Zbas.append(Lbas)
```

7) Enfin, après avoir effectué tous nos calculs, nous pouvons afficher nos résultats. La première fonction “plt.imshow” de la bibliothèque "matplotlib.pyplot" nous sert à tracer la fonction contenue dans notre matrice Z. Les paramètres origin et cmap sont respectivement l’endroit où est placé l’objet, la couleur de celui-ci. Le paramètre extent sert à placer notre graphique de xmin à xmax et de ymin à ymax. La fonction "plt.colorbar" est la barre que l'on voit après l'exécution de notre programme (sur la droite du graphique) et nous aide à voir l'échelle de nos gaussiennes. Pour afficher les extremas de chaque gaussienne, nous utilisons deux listes x_max et y_max ou l’on stocke les listes des x et y calculées auparavant. La fonction “plt.plot” sert à ajouter les extremas sur notre figure. “ro” est la couleur attribuée aux points. . La deuxième fonction “plt.imshow” ajoute les zones définies par la matrice Zbas.
Pour afficher les fonctions de matplotlib, il faut faire appel à "plt.show".

```
# tracé de la fonction f stockée dans matrice Z
plt.imshow(Z, extent=[xmin, xmax, ymin, ymax], origin='lower',
           cmap='hot', alpha=0.5)

# ajout de l'echelle de valeurs
plt.colorbar()

# points extremas de chaques gaussiennes

x_max = [Liste_x]
y_max = [Liste_y]

# ajout de ces extremas sur la figure
plt.plot(x_max, y_max,'ro')

# ajout d'un titre au dessus de notre graphique
plt.title("Gaussiennes vue du dessus")

# ajout des zones definies par Zbas
plt.imshow(Zbas, extent=[xmin, xmax, ymin, ymax], origin='lower',
           cmap='hot', alpha=0.5)

plt.show()
```

8) Voici le résultat obtenu de notre programme lorsque l’on demande 5 gaussiennes.



Pour conclure, nous avons donc réussi à afficher les gaussiennes en vue du dessus et ses extremums locaux. Cependant, nous aurions pu améliorer notre programme. Notamment lorsque nous calculons les extremums locaux. Effectivement, à la ligne 104, notre suite d'itérations "if" calcule plusieurs fois les mêmes points. Nous aurions pu faire en sorte de ne calculer que les points "utiles" et non tous ceux se trouvant autour de notre "x,y". Nous avons essayé de raccourcir notre programme le plus possible mais il est sûrement possible de le raccourcir encore plus et donc de faciliter son exécution.