TP1: Courbes, nuages de points

MP2I Option SI: Informatique tronc commun

Imports

Une fonctionnalité de python est l'import de bibliothèques/librairies/modules. Comme en C (et comme dans tous les langages), on peut utiliser des morceaux de code, des fonctions, venant d'autres fichiers. Il existe de nombreuses bibliothèques très standard en python. Par exemple, `math` contient de nombreuses fonctions et constantes en rapport avec les maths : sinus, exponentielle, logarithme, π, etc...

Pour importer une bibliothèque dans votre code, en haut du fichier, on écrit :

```
1 import math
```

Cette instruction fait que vous avez accès à tout ce qui est défini dans la bibliothèque `math`. Pour accéder à un élément `x` d'une bibliothèque `m`, on écrit `m.x` :

```
1 def perimetre(r):
2   """
3   Renvoie le périmètre d'un cercle de rayon r
4   """
5   return 2 * math.pi * r
6
7 a = math.sin(2) + math.exp(-1)
```

On peut également importer seulement certains éléments de la bibliothèque, par exemple :

```
1 from math import sin, pi
2
3 x = 2 * sin(2 * pi / 51)
```

On peut aussi importer tous les éléments d'une bibliothèque d'un coup :

```
1 from math import *    # le '*' signifie "tout"
2
3 x = 2 * sin(2 * pi / 51)
```

Cette troisième version est à éviter en général, car elle augmente le risque d'importer une fonction portant le même nom qu'une des votre, ce qui causera un conflit.

Chaînes de caractères

En Python, les chaînes de caractères peuvent être vues comme des listes *immutables*, ce qui veut dire qu'on ne peut pas les modifier. On peut facilement récupérer une sous-partie d'une chaîne avec la syntaxe suivante :

```
s = "bonjour"
t = s[3:6] # "jou"
```

Pour une chaîne s, $s[i:j]$ est donc la chaîne constituées des caractères $s[i], s[i+1], \ldots, s[j-1]$. On peut écrire $s[:j]$ à la place de $s[0:j]$ et $s[i:]$ au lieu de $s[i:len(s)]$.

Remarquons que la syntaxe précédente fonctionne aussi sur les listes :

```
l = [2, 3, 5, 7, 11]
t = l[1:4] # liste contenant [3, 5, 7]
```

On peut également concaténer les chaînes de caractères avec `+` :

```
s = "il fait super beau"
t = s[:7] + s[13:] # "il fait beau"
```

Une fonctionnalité un peu plus avancée des chaînes python est le *formatage*. Elle permet d'écrire directement le contenu d'une variable à l'intérieur d'une chaîne de caractères, sans avoir à utiliser `+`. Par exemple :

```
nom = "Pierre de Fermat"
age = 400
s = f"Je m'appelle {nom} et j'ai {age} ans"
```

La syntaxe est donc `f"chaine a formater"`, en mettant entre accolades les variables à utiliser.

Q1. Écrire une fonction `affiche_liste(L)` qui affiche les éléments de L sous la forme suivante :

```
>>> affiche_liste([3, 1, 41, 5]
L[0] = 3
L[1] = 1
L[2] = 41
L[3] = 5
```

Dictionnaires

Python possède nativement des structures de dictionnaires, qui permettent de créer des associations entre des **clés** et des **valeurs**. On peut :

- Créer un dictionnaire vide avec `dict()`
- Déterminer si une clé est dans un dictionnaire avec `k in d`
- Récupérer la valeur associée à une clé avec `d[k]`
- Modifier la valeur associée à une clé avec `d[k] = v`

On peut également itérer sur les clés d'un dictionnaire avec la syntaxe `for k in d`

Par exemple :

```
cris = dict()
cris["chat"] = "miaou"
cris["chien"] = "ouaf"
cris["hibou"] = "hou-hou"

for animal in cris:
  print(f"le cri du {animal} est {cris[animal]}")
```

Q2. Écrire une fonction `occurrences` qui prend en entrée une liste, et renvoie le dictionnaire de ses occurrences, c'est à dire un dictionnaire associant à chaque élément de la liste son nombre d'occurrences. Par exemple :

```
>>> occurrences(["bleu", "rouge", "bleu", "jaune", "rouge", "bleu"]
{"bleu": 3, "rouge": 2, "jaune": 1}
```

Numpy

La bibliothèque numpy est une bibliothèque de calcul scientifique, qui permet d'effectuer facilement des calculs sur des données à plusieurs dimensions, sur des listes de valeurs, etc...

Le type de base en numpy est le type "array", qui correspond à un tableau, éventuellement multi-dimensionnel, de valeurs. Par exemple :

```
import numpy as np # importe numpy et le renomme en np

# tableau 1D
a = np.array([1, 2, 3, 8])

# tableau 2D
b = np.array([
  [2, 3],
  [-8, 7]
])
```

Sur ce type, les opérations arithmétiques s'effectuent indice par indice, par exemple :

```
a = np.array([10,20,30])
b = np.array([5, 8, 7])
c = a + b  # c contiendra 15, 28 et 37
```

La bibliothèque contient de nombreuses fonctions mathématiques classiques : sinus, cosinus, etc... Ces fonctions opèrent différemment de celles de la bibliothèque `math` car elles s'appliquent sur chaque case d'un tableau numpy. Par exemple :

```
x = np.array([1, 2, 3, 4, 5])
y = np.sin(x) # contient [sin(1), sin(2), sin(3), sin(4), sin(5)
z = x**2
w = np.exp(x * y + z)
```

Tracé de courbes

La bibliothèque `matplotlib.pyplot` posssède de nombreuses fonctionnalités permettant de tracer des courbes, des histogrammes, etc...

Exemple 1. Pour tracer la courbe d'équation $y = x^2$, entre $x = -3$ et $x = 3$ avec 50 points distincts :

```
# il est classique de renommer matplotlib.pyplot en plt, et numpy en np
import matplotlib.pyplot as plt
import numpy as np

# 50 points parfaitement espacés entre -3 et 3
x = np.linspace(-3, 3, 50)
y = x*x # avec numpy, les opérations se font case par case.
        # y est donc un tableau de 50 cases, tq y[i] = x[i] * x[i]

# "r-" signifie que l'on trace une courbe rouge, formée de segments.
plt.plot(x, y, "r-")

# "bo" signifie que l'on trace des points bleus, non reliés
plt.plot(x, y, "bo")

plt.show()
```

Q3. Tracer la courbe de la fonction $x \mapsto e^{-\sin^2(3x+1)}$ entre -10 et 10, avec 500 points.

1 Wikipédia

Wikipédia propose une API (Application Programming Interface), c'est à dire une partie entière du site qui est faite spécialement pour être consultée par du code, où les pages sont des données sous format JSON (JavaScript Object Notation). En particulier, on peut obtenir des statistiques sur le site : nombre de visites d'un article, nombre d'éditions, etc...
Par exemple, pour accéder à la liste du nombre de visites mensuelles de la page "Computer Science", entre février 2015 et octobre 2022, on visite la page :

https ://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents/Computer_Science/monthly/2015020100/2022103100

La chaîne "2015020100" signifie "2015, février, le 01 , à 00h".
Le fichier `wikipedia.py` sur Cahier de Prépa contient une fonction faisant appel à cette API :

```
def nombre_visites(article, debut, fin):
    """
    Renvoie le nombre de visites mensuelles de l'article Wikipedia
    `article` entre les dates debut et fin.
    Entrées:
    - article: nom de l'article. Doit être un article anglais
    - debut et fin: dates limites incluses dans la requête.
      Doivent être au format "YYYYMM", par exemple "201605" pour Mai 2016.
    Sortie: liste de couples [(d1, n1), ..., (dk, nk)] avec chaque
    dj une date et chaque nj le nombre de visites à cette date.
    """
```

Cette fonction utilise les librairies `request` (pour faire des requêtes HTTP) et `json` (pour transformer la réponse en un dictionnaire python). Vous n'avez pas besoin de comprendre en détail comment cette fonction marche, on se contentera dans ce TP de l'utiliser pour tracer quelques courbes.

Q4. Utilisez cette fonction pour voir le nombre de visites mensuelles sur la page “Computer Science" durant l'année 2023.

Q5. Écrire une fonction permettant de calculer le nombre **total** cumulé de visites d'un article entre deux dates.

Q6. Écrire une fonction permettant de calculer la liste des visites cumulées au cours du temps. Par exemple, si un article a reçu 20 visites en mai 2023 , 25 visites en juin 2023 et 12 visites en juillet 2023, la fonction renverra :

```
[(20230501, 20), (20230601, 45), (20230701, 57)]
```

Q7. En utilisant la fonction précédente, écrire une fonction `plot_monthly_views` prenant en entrée une **liste de noms d'articles** et **deux dates** de début et fin, et traçant la **somme cumulées** des visites mensuelles de chaque article entre les deux dates. On peut directement utiliser des dates au format “MM/AAAA" en abscisse avec `pyplot`.

Pour finir, nous allons rajouter une légender à notre courbe. Pour cela, il faudra :

— Préciser le nom de la courbe lorsque l'on appelle `plt.plot` avec l'argument mot-clé `label=...`, par exemple `plt.plot(x, y, 'r-', label="Courbe A")`.

— Positionner l'encadré des légendes avec `plt.legend(loc="upper center")`.

Q8. Modifier la fonction précédente pour rendre le graphique plus lisible, avec une légende. Comparer l'évolution de la popularité de différentes pages Wikipédia au cours des dernières années : “Greenland", “Blackpink", etc...

2 Nuage de points

On peut dessiner des nuages de points avec `matplotlib` :

```
import matplotlib.pyplot as plt
import random

def random_point():
    """
    Renvoie une point au hasard dans le rectangle [-50,50] x [-50, 50],
    à coordonnées entières.
    """
    return (random.randint(-50, 50), random.randint(-50, 50))

l = [random_point() for i in range(100)]

x = [p[0] for p in l] # liste des abscisses
y = [p[1] for p in l] # liste des ordonnées

plt.plot(x, y, 'ro')
plt.show()
```

Enveloppe convexe

Étant donné un nuage de points, on veut déterminer de manière efficace son enveloppe convexe. Plus précisément, on veut obtenir une liste de points à relier pour former le contour de cette enveloppe.

Il existe plusieurs algorithmes permettant de calculer l'enveloppe convexe d'un nuage de points. On étudie ici l'algorithme du scan de Graham :

Algorithme 1 : Algorithme du scan de Graham

Entrée(s) : P une liste de points du plan $\mathbb{R}^2$
Sortie(s) : H liste des points extrêmes de l'enveloppe convexe de P

$S \leftarrow$ pile vide;
Trouver le point d'ordonnée minimale, p_0. S'il y en a plusieurs, prendre celui d'abscisse maximale;
Trier P par angle croissant par rapport à p_0 et à l'horizontale;
Si plusieurs points ont le même angle, garder seulement le plus éloigné de p_0.
pour $p \in P$ **faire**
    tant que $|S| \geq 2$ *et* p *est à droite du segment orienté* $S[-2]; S[-1]$ **faire**
        Dépiler S;
    Empiler p sur S;
retourner S

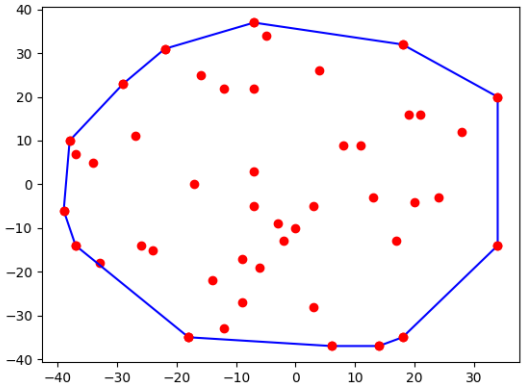

FIGURE 1 – Enveloppe convexe d'un nuage de points

Dans cet algorithme $S[-1]$ est le sommet de S, et $S[-2]$ l'élément du dessous. On pourra implémenter les piles avec des listes python simples, ce qui permettra de les traiter à la fois comme des piles et comme des tableaux. Si T est un tableau, alors `T.append(x)` ajoute x à la fin de T, et `T.pop()` renvoie l'élément à la fin de T et le supprime. Python gère automatiquement le redimensionnement.

L'idée est donc de partir du point de plus faible ordonnée, puis de traiter les autres points en tournant autour du point de départ dans le sens anti-horaire. On supprime au fur et à mesure les points qui nous font tourner à droite, en gardant comme invariant que la pile S contient une liste de points qui tourne toujours à gauche.

Q9. Dessiner un nuage d'une quinzaine de points sur une feuille, et appliquer l'algorithme de Graham. Vérifier que l'on obtient bien l'enveloppe convexe.

Q10. Étant donné trois points A, B, C, comment peut-on simplement déterminer par calcul si l'angle $\widehat{ABC}$ tourne à gauche ou à droite, sans toutefois calculer explicitement l'angle ?

Q11. Créer une fonction `tourne_droite` prenant en entrée trois points a, b, c et déterminant si l'angle $\widehat{a;b;c}$ tourne à droite.

Q12. Implémenter l'algorithme de Graham, et l'utiliser sur des nuages de points générés aléatoirement. On tracera le nuage de point, et on reliera les points de l'enveloppe convexe.

Q13. Quelle est la complexité de cet algorithme ?

On appelle *point extrême* d'un nuage de points tout point se trouvant sur un angle de l'enveloppe convexe. L'algorithme de Graham renvoie précisément la liste de ces points. Pour $n \in \mathbb{N}$, on note $V(n)$ le nombre moyen de points extrêmes d'un nuage de n points 2D choisis uniformément au hasard dans le carré unité $[0, 1[\times[0, 1[$.

Q14. Écrire une fonction qui prend en entrée un entier n et génère une liste de n points aléatoires uniformément choisis dans le carré unité $[0, 1[\times[0, 1[$ et renvoie le nombre de points extrêmes, c'est à dire le nombre de points faisant partie de l'enveloppe convexe du nuage de points et étant sur des angles (et pas au milieu d'un segment du contour).

Q15. Déterminer empiriquement la loi que semble suivre $V(n)$.

3 Détection de contours

On s'intéresse à la détection des contours dans une image. On représente une image par une matrice de pixels. En python, les librairies `matplotlib.pyplot` et `numpy` possède de nombreux outils permettant de manipuler les matrices de pixels.

On importe classiquement ces deux librairies comme suite :

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

Pour charger une image, on utilise `plt.imread` :

```
1 a = plt.imread("elephant.png")
```

a est alors une matrice de taille $n \times m$, dont chaque élément est un triplet (R, G, B) de flottants compris entre 0 et 1, représentant le contenu de chaque pixel en rouge / vert / bleu. Une image en noir et blanc est simplement représentée par une matrice de $n \times m$ flottants compris entre 0 et 1, représentant les nuances de gris. On peut accéder au pixel de la ligne i et de la colonne j dans la matrice A avec la syntaxe $A[i, j]$, ou alors $A[i][j]$ comme si c'était un tableau de tableaux. On préfèrera la première option.

Le principe de la détection de contours est de dire qu'un pixel fait partie du contour s'il est relativement différent des pixels alentours. Pour cela, on calcule la distance euclidienne moyenne entre le pixel et ses voisins, où un pixel (r, g, b) est vu comme un point dans l'espace $\mathbb{R}^3$. Ainsi, un point sera considéré comme faisant partie du contour si sa couleur est assez différente de celle de ses voisins :

Algorithme 2 : Détection de contours

```
[H] Entrée(s) : A de taille n × m image sous forme de pixels
    Sortie(s) : B image contenant les contours de A
A' ← version noir-et-blanc de A;
B ← matrice nulle de taille n × m;
pour i = 0 à n − 1 faire
    pour j = 0 à m − 1 faire
        d ← la distance moyenne entre A'[i, j] et ses voisins;
        si d > SEUIL alors
            B[i, j] ← 1.0 ;
retourner B
```

On fixe la valeur `SEUIL = 0.15` pour le moment.

Q16. Écrire une fonction `nuances_gris` prenant en entrée une matrice de pixels RGB et renvoyant la matrice noir-et-blanc correspondante, en prenant comme nuance de gris la moyenne des trois couleurs. On pourra utiliser la fonction `np.sum`.

Q17. Écrire une fonction `est_contour` prenant en entrée une matrice de pixels a en noir-et-blanc, deux entiers i, j et déterminant si le pixel $a[i, j]$ fait partie du contour.

Q18. Écrire une fonction `contourer` prenant en entrée un nom de fichier image, et créant un nouveau fichier image avec le contour de l'image d'entrée. Le nom du fichier image sera de la forme : `"{nom_fichier_base}_contour.png"`

Vous pouvez tester cette fonction sur les images mises dans l'archive, ou sur d'autres images de votre choix.