Université de Montpellier **HLMA310**
2019/2020 **Joseph Salmon**

TP N° 3 : `numpy` et `matplotlib`

Objectifs du TP : apprendre à utiliser `numpy` et de `matplotlib` savoir faire de la manipulation de matrices et de l'affichage graphique.

Commencer par nommer votre fichier en suivant la même procédure, et en utilisant `filename` pour votre nom de TP :

```
# Changer ici par votre Prenom Nom:
prenom = "Joseph" # à remplacer
nom = "Salmon" # à remplacer
extension = ".ipynb"
tp = "TP3_HMLA310"
filename = "_".join([tp, prenom, nom]) + extension
filename = filename.lower()
```

Exercice 1. (Dynamique d'une population population avec migrations internes)

On considère P la population d'un pays (que l'on suppose constante au cours du temps), divisée en une population rurale et une population urbaine. On note r_n et u_n les populations rurales et urbaines à l'année n, τ_0 le taux d'exode rural et τ_1 le taux d'exode urbain. On note enfin $p_n = \begin{pmatrix} r_n \\ u_n \end{pmatrix} \in \mathbb{R}^2$ le vecteur décrivant la population totale[1].

Pour les applications numériques on prendra $r_0 = 9$ (millions) et $u_0 = 51$ (millions), $\tau_0 = 0.3$, et $\tau_1 = 0.1$.

1) Montrez (mathématiquement) que l'on peut écrire p_{n+1} comme une transformation linéaire de p_n. En particulier trouvez une matrice $M \in \mathbb{R}^{2\times 2}$ telle que la relation suivante soit vraie pour tout $n \in \mathbb{N}$:
$$p_{n+1} = Mp_n$$
2) Donner l'écriture mathématique de p_n en fonction de n, M et $p_0 = \begin{pmatrix} r_0 \\ u_0 \end{pmatrix}$.
3) Écrire une fonction `Python` appelée `population_array` qui prend en argument n, p_0, τ_0, τ_1 et qui renvoie la matrice suivante :
$$\begin{pmatrix} r_0 & u_0 \\ \vdots & \vdots \\ r_n & u_n \end{pmatrix} \in \mathbb{R}^{(n+1)\times 2} \quad .$$
4) Affichez sur un graphique l'évolution de la population urbaine <u>et</u> rurale sur une période de 10 ans. On ajoutera les éléments suivants sur le graphique :
   - un titre, avec `plt.title`,
   - une légende, avec `plt.legend`,
   - des couleurs différentes pour les deux types de populations,
   - des noms pour l'axe des x et l'axe des y, avec `plt.xlabel, plt.ylabel`,
   - on veillera à ce que l'axe des y commence en 0.
5) Ajoutez sur ce même graphique la somme totale de la population.
6) Construisez un autre graphique de visualisation, similaire à celui de la Figure 1. On pourra utiliser par exemple la fonction `fill_between`

---

1. en particulier $P = u_n + r_n$ pour tout entier n.

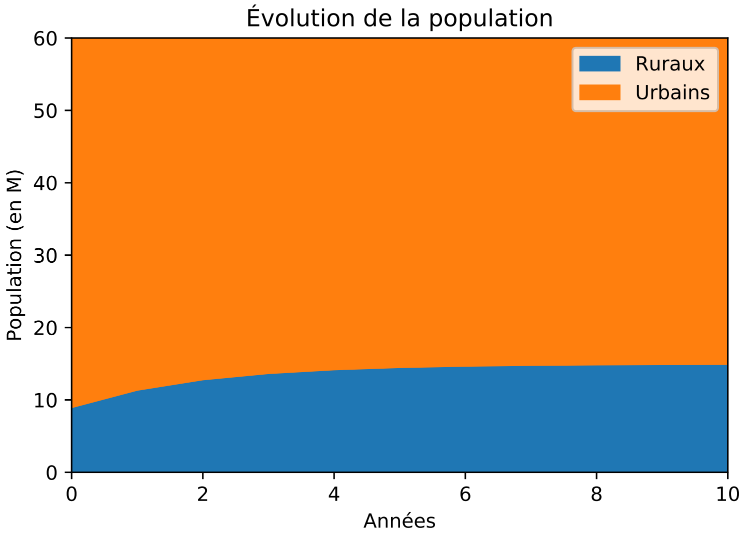


FIGURE 1 – Visualisation alternative

7) Calculez M^{1000} avec la fonction `np.linalg.matrix_power`.
8) Vu les graphiques précédents, interpréter ce que vaut (approximativement) $M^{1000}p_0$.
9) Vérifiez que M est diagonalisable : pour cela trouvez $P \in \mathbb{R}^{2\times 2}$ (inversible) et $D \in \mathbb{R}^{2\times 2}$ (diagonale) telles que : $M = PDP^{-1}$. On pourra utiliser `np.linalg.eig`.[2]
10) Vérifier numériquement avec `np.all_close` que M^{1000} et $PD^{1000}P^{-1}$ sont (presque) égaux.

EXERCICE 2. (Fractales et ensemble de Mandelbrot)

L'ensemble de Mandelbrot[3] est un ensemble (fractal) du plan illustré en Figure 2, et que l'on peut définir de la manière suivante : prenons la suite récurrente de points du plan définie par

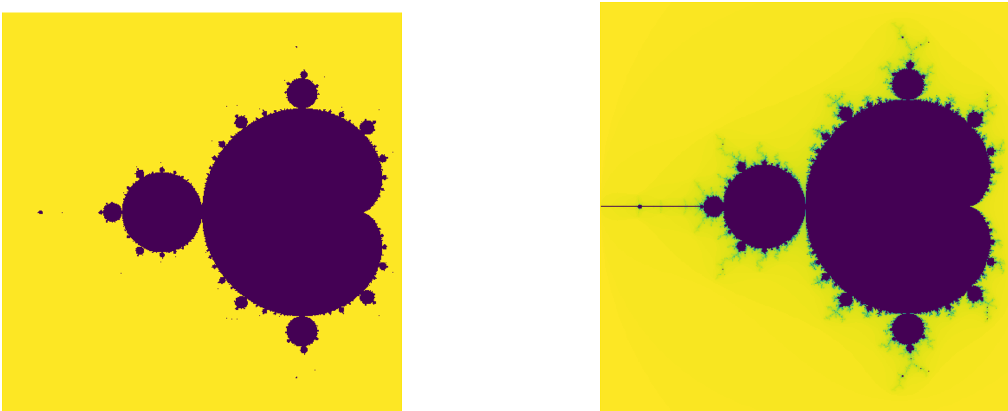
- $p_0 = (x_0, y_0) = (0, 0)$
- $p_{n+1} = (x_{n+1}, y_{n+1})$ qui suit la récurrence suivante :

$$\begin{cases} x_{n+1} = x_n^2 - y_n^2 + a \\ y_{n+1} = 2x_n y_n + b \end{cases} . \tag{1}$$

Si la suite suite $(p_n)_{n\in\mathbb{N}}$ vérifie $\forall n \in \mathbb{N}, \|p_n\| < 2$, alors le point (a, b) appartient à l'ensemble de Mandelbrot, sinon il est dans son complémentaire. En pratique on ne peut vérifier cette propriété que jusqu'à un nombre n inférieur à `max_iteration`, ce que l'on fera pour la partie numérique.

1) Utiliser le code suivant (disponible dans le fichier notebook associé au TP) pour générer une matrice dont les termes valent 1 ou 0 selon que le point de coordonnée associée est ou non dans l'ensemble de Mandelbrot. Créer une fonction `Mandelbrot` prenant comme entrées `max_iteration, hauteur, largeur, xmin, xmax, ymin, ymax` et ressort une telle matrice `mandelbrot_mat`.

2. pour retrouver cette fonction : valeur propre (🇬🇧 : *eigen value*) / vecteur propre (🇬🇧 : *eigen vector*)
3. comme souvent en mathématique, le nom est trompeur et l'on devrait plutôt parler d'ensemble de Julia : https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot



(a) Visualisation en version binaire

(b) Visualisation en version multi-couleurs

Figure 2 – Ensemble de Mandelbrot

```
mandelbrot_mat = np.zeros((largeur, hauteur))
for x in range(hauteur):
    cx = (x * (xmax - xmin) / hauteur + xmin)
    for y in range(largeur):
        cy = (y * (ymin - ymax) / largeur + ymax)
        xn, yn, n = 0, 0, 0
        while (xn**2 + yn**2) < 4 and n < max_iteration:
            tmp_x, tmp_y = xn, yn
            xn = tmp_x**2 - tmp_y**2 + cx
            yn = 2 * tmp_x * tmp_y + cy
            n = n + 1
        if n < max_iteration:
            mandelbrot_mat[y, x] = 1.
```

2) Utiliser la fonction `Mandelbrot` et la fonction `imshow` de `matplotlib` pour afficher une approximation de l'ensemble de Mandelbrot avec les paramètres

```
max_iteration = 100
xmin, xmax, ymin, ymax = -2, 0.5, -1.25, 1.25
largeur, hauteur = 500, 500 # résolution visuelle
```

3) Utiliser une boucle `for` pour afficher les ensembles obtenues pour le nombre d'itérations dans `max_iterations = [1, 2, 5, 10, 20, 50, 100]` (les autres paramètres étant fixés comme précédemment). On veillera à d'abord construire un `array` de taille `largeur` × `hauteur` × 7 nommé `mandelbrot_mats` et qui contient les 7 matrices à afficher. Enfin, on fera une boucle pour afficher chacun des ensemble associés [4]. On pourra s'inspirer du code suivant :

```
fig, axes = plt.subplots(2,4,figsize=(8,4))
axs = axes.ravel()
for i in range(len(max_iterations)):
    axs[i].plot(np.sin(np.linspace(0,1,num=100) * i *2 * np.pi))
plt.show()
```

4) Question bonus : Modifier votre fonction pour qu'elle ressorte un affichage continue au lieu d'un affichage binaire (0/1) de cet ensemble. On pourra par exemple utiliser le premier indice tel que le test est non valide dans la boucle `while`.

4. conseil général : il faut toujours séparer la gestion/création/sauvegarde des données, qui est une tâche chronophage, et la partie affichage qui elle est souvent instantanée.