Université de Montpellier **HLMA310**
2018/2019 **Joseph Salmon**

TP n° 3 : `numpy` et `matplotlib`

Objectifs du TP : apprendre à utiliser `numpy` et de `matplotlib` pour de la manipulation de matrice et d'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. Évolution d'une population avec migrations internes

On considère 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.

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) Montrer que l'on peut écrire p_{n+1} comme une transformation linéaire de p_n. En particulier trouver 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 `population_array` qui prend en argument n, p_0, τ_0, τ_1 et qui renvoie une matrice de taille $(n+1) \times 2$ de la forme suivante :

$$\begin{pmatrix} r_0 & u_0 \\ \vdots & \vdots \\ r_n & u_n \end{pmatrix}$$

4) Afficher sur un graphique l'évolution de la population urbaine et rurale sur une période de 10 ans. On ajoutera les éléments suivants sur le graphique :
   - un titre, voir `plt.title`
   - une légende, voir `plt.legend`
   - des couleurs différentes pour les deux types de populations
   - des noms pour l'axe des x et l'axe des y, voir `plt.xlabel,plt.ylabel`
   - on veillera à ce que l'axe des y commence à 0

5) Ajouter sur ce même graphique la somme totale de la population.

6) Construire un autre graphique de visualisation, similaire à celui de la Figure 1.

7) Calculer M^{1000} avec la fonction `np.linalg.matrix_power`.

8) Vu les graphiques précédents, interpréter ce que vaut (approximativement) $M^{1000}p_0$.

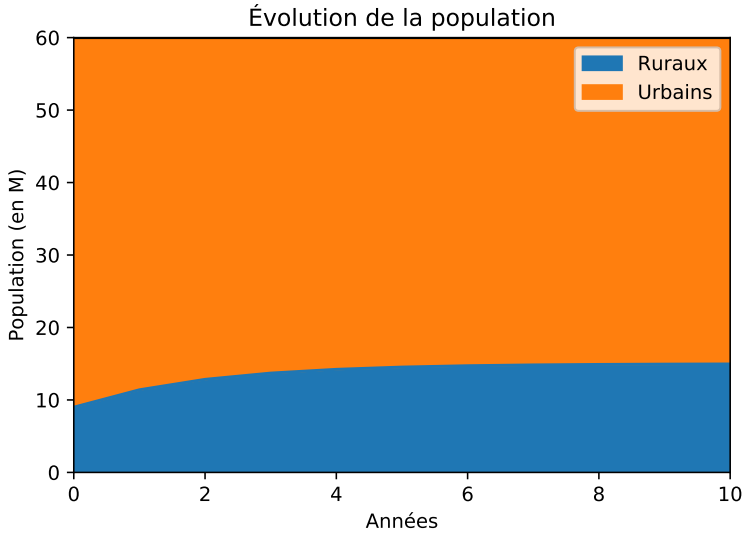


FIGURE 1 – Visualisation alternative

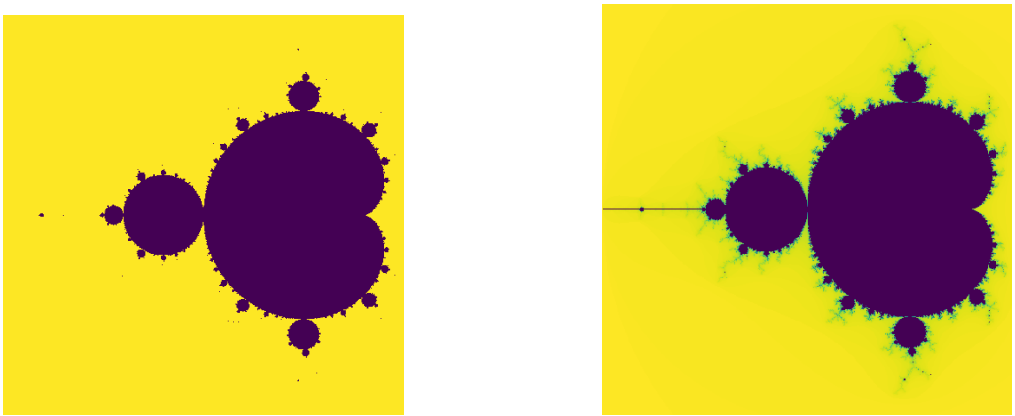
9) Calculer la décomposition spectrale de M à savoir trouver $P \in \mathbb{R}^{2\times 2}$ (inversible) et $D \in \mathbb{R}^{2\times 2}$ (diagonale) telles que : $M = PDP^{-1}$.

10) Vérifier numériquement avec `np.all_close` que M^{1000} et $PD^{1000}P^{-1}$ sont égaux.

EXERCICE 2. Fractales et ensemble de Mandelbrot

L'ensemble de Mandelbrot 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 : $p_0 = (x_0, y_0) = (0, 0)$ et $p_n = (x_n, y_n)$ pour

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



(a) Visualisation en version binaire (b) Visualisation en version multi-couleurs

FIGURE 2 – Ensemble de Mandelbrot

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 [1]. 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. cet ensemble admet une définition alternative et complémentaire que l'on peut retrouver sur https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot

1) Utiliser le code suivant 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`.

```
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 = 200
xmin, xmax, ymin, ymax = -2, 0.5, -1.25, 1.25
largeur, hauteur = 1000, 1000 # résolution visuelle
```

3) Pour désactiver les axes dans toutes les images, lancer la commande :

```
import matplotlib as mpl
mpl.rc('axes.spines', top=False, bottom=False, left=False, right=False)
mpl.rc(('xtick', 'ytick'), color=(1, 1, 1, 0))
```

4) Utiliser des boucles `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` $\times$ `hauteur` $\times$ 7 nommé `mandelbrot_mats` et qui contient les 7 matrices à afficher. Enfin, on fera une boucle pour afficher chacun des ensemble associés [2].

5) 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`.

2. 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.