IV - Représentation graphique

Le module (ou bibliothèque) matplotlib.pyplot permet de faire de la représentation graphique. Parmi les fonctions de ce module, il y a :

- `plot`.
  `plot(x)` crée une ligne brisée reliant les points dont les abscisses sont contenues dans `[0, ..., n-1]` et les ordonnées dans `x` où n est la longueur ou nombre de termes de `x`.
  `plot(x, y)` crée une ligne brisée reliant les points dont les abscisses sont contenues dans `x` et les ordonnées dans `y`, il faut donc que `x` et `y` contiennent le même nombre d'éléments.
- `show`.
  `show()` affiche les objets graphiques créés.

Exemple 1 Dans l'interpréteur de commandes taper `import matplotlib.pyplot as plt`, ce qui permet de disposer de toutes les fonctions du module matplotlib.pyplot en les préfixant par *plt.*
Valider successivement les instructions : `x=[-1, 3, 5, -0.3, 4]` puis `plt.plot(x)` et `plt.show()`.
Expliquer ce que fait la fonction plot lorsqu'on l'applique à une seule liste.

EXERCICE 1 : représentation graphique de la suite logistique
Écrire une fonction Python de paramètres n et a qui stocke dans une liste les n premiers termes de la suite (u_n) définie par $u_0 = \frac{1}{2}$ et $\forall n \in \mathbb{N},\ u_{n+1} = au_n(1-u_n)$ et représente graphiquement cette liste.
On donnera une autre représentation graphique de cette suite dans un TD ultérieur.
On commencera par importer les bibliothèques `matplotlib.pyplot` et `numpy` dans le fichier .py pour tous les exercices.

Exemple 2 Valider par la touche entrée chacune des instructions suivantes :
`x = [0, 1, 2]`, `y = [0, 3, 0]`, `plt.plot(x, y)` et enfin `plt.show()`. Qu'obtient-on ? Comment expliquer ce résultat ?
Taper `x=[0.5, 1.5]` et `y=[1.5, 1.5]` puis `plt.plot(x, y)`. Expliquer la figure obtenue.

Sous Pyzo, à chaque appel de la fonction plot, la figure obtenue se superpose aux précédentes.
Pour afficher d'autres objets sur une nouvelle fenêtre il suffit de fermer l'ancienne fenêtre et de commander l'affichage des nouveaux graphiques par la commande `plt.show()`.

Pour tracer la courbe d'une fonction on a besoin du module numpy contenant notamment la fonction `linspace` qui génère la subdivision régulière d'un intervalle sous la forme d'un tableau numpy (*np.array*).

Exemple 3 Dans le shell taper `import numpy as np`, ce qui permet de disposer de toutes les fonctions du module numpy en les préfixant pas *np.* Valider l'instruction `np.linspace(0, 1, 11)` puis `np.linspace(0,2, 15)`. Expliquer ce que renvoie `np.linspace(a, b, n)`.

Pour construire la courbe d'une fonction entre a et b, il suffit de relier un grand nombre de points de cette courbe par de petits segments de droite. Cette opération s'appelle interpolation linéaire de la courbe entre a et b.
Si `f` est une fonction et `x` un tableau (*array*) alors `f(x)` est le tableau des images des éléments de `x` par `f`. Par conséquent l'instruction `plt.plot(x, f(x))` crée une ligne brisée reliant les points de la courbe de `f` de coordonnées `x`. La commande `plt.axis([a,b,c,d])` impose une représentation graphique sur le pavé $[a,b]\times[c,d]$ alors que `plt.axis("equal")` impose un repère orthonormé. `plt.grid()` crée un quadrillage et `plt.legend()` fait ressortir les labels que l'on peut mettre en option dans la fonction *plot* sous la forme `plt.plot(x,f(x),label="y=f(x)")`.

EXERCICE 2 : courbes des fonctions usuelles
Écrire sept fonctions Python qui affichent les courbes des fonctions proposées dans un même graphique. Les paramètres seront a et b correspondant aux bornes inférieure et supérieure de l'intervalle de la représentation $[a,b]$. Le nombre de points utilisés pour les interpolations sera égal à 200. On ajoutera une légende à chaque courbe et on quadrillera.

1. *logarithmes* qui affiche les courbes des fonctions logarithme népérien et logarithme décimal. On se limitera aux ordonnées comprises entre -4 et 3.
2. *exponentielles* qui affiche les courbes des fonctions exponentielles de bases e, 10, 2 et $\frac{1}{2}$. On se limitera aux ordonnées comprises entre 0 et 15.
3. *puissances* qui affiche les courbes des fonctions $x \longmapsto x^2$, $x \longmapsto x^{\frac{1}{2}}$, $x \longmapsto x^{-\frac{3}{2}}$, $x \longmapsto x^{-\frac{1}{2}}$. On se limitera aux ordonnées comprises entre 0 et 10.
4. *polynomes* qui affiche les courbes des polynômes $x \longmapsto x^3 - 2x^2 - 2x + 3$ et $x \mapsto x^4 + x^3 - 5x^2 - x + 2$. On se limitera aux ordonnées comprises entre -20 et 20.
5. *trigonometriques* qui affiche les courbes des fonctions sinus et cosinus. On se placera dans un repère orthonormé.
6. *tangente* qui affiche la courbe de la fonction tangente. On se placera dans un repère orthonormé et on se limitera aux ordonnées comprises entre -5 et 5.
7. *pe_va* qui affiche les courbes des fonctions $x \longmapsto \lfloor x \rfloor$ et $x \longmapsto |x|$. On se placera dans un repère orthonormé.

EXERCICE 3 : représentation graphique des suites usuelles

1. Écrire une fonction python *arithmetique*(r, u, n) qui affiche la courbe des n premiers termes de la suite arithmétique de raison r et de premier terme u. Appeler cette fonction pour les paramètres $2, -3, 10$ et $-1, 1, 10$. Que remarque-t-on ? Comment peut-on caractériser géométriquement les suites arithmétiques ?
2. Écrire une fonction python *geometrique*(q, u, n) qui affiche la courbe des n premiers termes de la suite géométrique de raison q et de premier terme u. Appeler cette fonction pour les paramètres $\frac{1}{10}, 2, 10$ et $10, \frac{1}{3}, 8$. Que remarque-t-on ? Comment appelle-t-on ce type de croissance et de décroissance ? Que se passe-t-il lorsqu'on remplace l'échelle classique par une échelle logarithmique sur l'axe des ordonnées uniquement ?
3. Écrire une fonction python *srl2*(a, b, u, v, n) qui affiche la courbe des n premiers termes de la SRL2 $u_{n+2} = au_{n+1} + bu_n$ vérifiant $u_0 = u$ et $u_1 = v$. On appellera cette fonction pour les paramètres $1, 1, 1, 1, 8$ et $2, -1, 1, 1, 8$ et $2, -2, 1, 1, 8$ et $1, -\frac{1}{2}, 1, 1, 8$.

EXERCICE 4 : marches aléatoires sur une droite

Initialement, un jeton est positionné à l'origine de la droite numérique. On lance une pièce équilibrée et on déplace le jeton d'une unité à droite si le résultat est Pile et d'une unité à gauche si le résultat est Face. On répète cette opération.

1. Écrire une fonction `marche1` qui prend en entrée un entier naturel `n` et qui affiche les n premières positions du jeton en fonction du temps.

```
import random as rd
def marche1(n):
    x, lstx = 0, [0]
    for k in range(...):
        lancer = ...
        if lancer: ...
        else: ...
        ...
    plt.plot( ... )
    plt.show()
```

   Appeler *marche1* pour $n = 10, 100, 1000, 10000, 100000, 1000000$.
2. Écrire une fonction Python *origine1(n)* qui renvoie le nombre de déplacements nécessaires pour que le jeton repasse n fois par l'origine. Appeler *origine1* pour $n = 1, 10, 100$. Que peut-on conjecturer ?.
3. Soit X le nombre de déplacements nécessaires pour revenir une première fois à l'origine. Quelle commande simule la variable aléatoire X ? On admettra que si une variable aléatoire discrète finie ou infinie (programme de BCPST2) admet une espérance μ alors la moyenne de n simulations indépendantes de cette VA tend (en un certain sens) vers μ lorsque n tend vers $+\infty$. Écrire une fonction `moyX(n)` qui renvoie la moyenne de n simulations distinctes de X. Appeler `moyX` pour $n = 5, 50, 500$. Que remarque-t-on ? Que peut-on conjecturer ?

EXERCICE 5 : marches aléatoires dans le plan

Initialement, un jeton est positionné à l'origine d'un repère orthonormé du plan. On lance une pièce équilibrée à deux reprises et on déplace le jeton :

- d'une unité à droite si le résultat des lancers est Pile-Pile,
- d'une unité à gauche si le résultat est Pile-Face,
- d'une unité en haut si le résultat est Face-Pile et
- d'une unité en bas si le résultat est Face-Face.

On répète cette cette opération.

1. Écrire une fonction `marche2` qui prend en entrée un entier naturel `n` et qui affiche la marche aléatoire dans le plan des n premiers déplacements. On pourra compléter le code suivant :

```
def marche2(n):
    x, y, lstx, lsty = 0, 0, [0], [0]
    for k in range(...):
        lancer1, lancer2 = ........., .........
        if lancer1:
            if lancer2: ...
            else: ...
        else:
            if lancer2: ...
            else: ...
        ...
        ...
    plt.plot(..., ...)
```

```
    plt.axis('equal')
    plt.show()
```

Appeler `marche2` pour $n = 10, 100, 1000, 10000, 100000, 1000000$.

2. Écrire une fonction Python *sortie2(n)* qui renvoie le nombre de déplacements nécessaires pour que le jeton sorte du disque de centre l'origine et de rayon n. Appeler *sortie2* pour $n = 10, 100, 1000$. Que peut-on conjecturer ?

Tracé d'un diagramme en barres

Soient `X` et `Y` des listes ou des tableaux de nombres.
`plt.bar(X,Y)` trace le diagramme en barres dont les valeurs et les hauteurs sont contenues dans `X` et `Y`.

EXERCICE 6 : Simulation d'une loi de Bernoulli

Soit X une variable aléatoire de Bernoulli de paramètre p ($0 < p < 1$). On rappelle que X prend la valeur 1 avec la probabilité p et la valeur 0 avec la probabilité $1 - p$.

1. Écrire une fonction `simulBernoulli(p)` prenant en paramètre un nombre $p \in]0, 1[$ qui simule une variable aléatoire qui suit la loi Bernoulli de paramètre p. Elle renverra 0 ou 1 et utilisera la fonction `random` du module `random`. On rappelle que la commande `rd.random()<p` prend la valeur *True* avec la probabilité p et permet donc de simuler un événement de probabilité p.
2. Écrire une fonction `loiBernoulli(p, k, N = 10000)` prenant en paramètres p et un entier $k \in \{0, 1\}$ qui renvoie une approximation de $\mathbb{P}(X = k)$. Cette fonction appellera N fois `simulBernoulli(p)` et renverra la proportion des résultats égaux à k.
3. Écrire une fonction `BarBernoulli(p, N = 10000)` qui affiche le diagramme en barres de la loi de Bernoulli de paramètre p à l'aide de N simulations. La fonction `BarBernoulli` ne doit pas utiliser la fonction `loiBernoulli`. Comparer ce diagramme en barres simulé au diagramme en barres théorique pour les paramètres $\frac{1}{3}$ et 0.9.