1BCPST2 Représentations graphiques de suites et de fonctions avec Python

1 Bibliothèque graphique et bibliothèque de calcul matriciel

La bibliothèque *matplotlib.pyplot* qui contient des fonctions graphiques est fréquemment importée de la façon suivante :

```
import matplotlib.pyplot as plt
```

• Si `X` et `Y` sont des listes de nombres contenant chacune n termes alors `plt.plot(X,Y)` crée la ligne brisée formée par les segments de droite reliant les points de coordonnées $(X[0], Y[0])$, $(X[1], Y[1]), \dots, (X[n-1], Y[n-1])$ dans cet ordre.
• Si `Y` est une liste de n nombres alors `plt.plot(Y)` crée la ligne brisée formée par les segments de droite reliant les points de coordonnées $(0, Y[0])$, $(1, Y[1]), \dots, (n-1, Y[n-1])$ dans cet ordre.
• La fonction `plt.show()` affiche à l'écran les objets graphiques précédemment créés quand on travaille avec pyzo.

La bibliothèque *numpy* qui contient des fonctions pour le calcul matriciel est importée de la façon suivante :

```
import numpy as np
```

• `np.linspace(a, b, n)` renvoie un tableau *numpy* à une entrée de $\boldsymbol{n}$ nombres régulièrement espacés dont le premier est $\boldsymbol{a}$ et le dernier $\boldsymbol{b}$.
• `np.arange(a, b, h)` renvoie un tableau *numpy* à une entrée de nombres régulièrement espacés dont le premier terme est $\boldsymbol{a}$ (**inclus**) et le dernier $\boldsymbol{b}$ (**exclu**) avec un pas égal à $\boldsymbol{h}$. Si ce $3^{\text{ème}}$ argument n'est pas renseigné alors il vaut 1 par défaut.
• Si `f` est une fonction compatible avec les tableaux *numpy* et `t` un tableau *numpy* alors `f(t)` renvoie le tableau des images des termes de `t` par `f`. La fonction `plt.plot` peut également prendre en argument des tableaux *numpy*.
Le module *numpy* génère un code plus concis et surtout une exécution beaucoup plus rapide pour des tâches répétitives.

2 Représentation graphique d'une suite

Soit $(u_n)_{n\in I}$ une suite indexée par I. En général I est égal à $\mathbb{N}$ ou $\mathbb{N}^*$. On va se limiter au cas où $I = \{n_0, n_0+1, \dots\}$.
La courbe de la suite $(u_n)_{n\geqslant n_0}$ est la ligne brisée obtenue en reliant les points de coordonnées (n_0, u_{n_0}), $(n_0+1, u_{n_0+1}), \dots$ par des segments de droite.

On suppose que l'on dispose de la fonction Python `u` de paramètre `n` qui renvoie u_n.

La fonction `courbe_suite1` qui prend en paramètre l'entier `n` affiche la courbe des n premiers termes de la suite $(u_n)_{n\in\mathbb{N}}$.

```
def courbe_suite1(n):
    """affiche la courbe de (u[0],...,u[n-1])"""
    U = [u(k) for k in range(n)]
    plt.plot(U)
    plt.show()
```

La fonction `courbe_suite2` qui prend en paramètre l'entier `n` affiche la courbe des n premiers termes de la suite $(u_n)_{n\in\mathbb{N}^*}$.

```
def courbe_suite2(n):
    """affiche la courbe de (u[1],...,u[n])"""
    U = [u(k) for k in range(1,n+1)]
    I = [k for k in range(1,n+1)]
    plt.plot(I, U)
    plt.show()
```

On suppose maintenant que $(u_n)_{n\geqslant n_0}$ est définie par la relation de récurrence $\forall n \geqslant n_0,\ u_{n+1} = f(u_n)$ et que l'on dispose de la fonction Python `f` correspondant à la fonction mathématique de même nom.
La fonction `courbe_suite3` qui prend en paramètres l'entier `n` et le premier terme `a` de la suite $(u_n)_{n\geqslant n_0}$ affiche la courbe des n premiers termes de cette suite.

```
def courbe_suite3(n,a):
    """affiche la courbe de (u[n0],...,u[n0+n-1]) si pour tout k, u[k+1] = f(u[k])"""
    U = [a]
    I = [n0]
    for k in range(n0+1,n0+n):
        a = f(a)
        U.append(a)
        I.append(k)
    plt.plot(I, U)
    plt.show()
```

On observe que le module *numpy* n'est pas nécessaire pour représenter graphiquement une suite.

3 Courbe d'une fonction

Soit f une fonction définie sur le segment $[a, b]$.
On rappelle que la courbe de la fonction f est l'ensemble des points de coordonnées $(x, f(x))$ avec $x \in [a, b]$.

Pour représenter une courbe avec Python on procède par discrétisation en créant la ligne brisée qui relie par des segments de droite des points de la courbe régulièrement espacés. Cette ligne brisée est appelée interpolation linéaire de la courbe de f. En diminuant le pas de la subdivision de l'interpolation (ou, ce qui est équivalent, en augmentant son nombre de points) on arrive à "lisser" les points anguleux à l'œil nu ce qui nous permet de confondre l'interpolation et la vraie courbe de f.

On suppose que l'on dispose de la fonction Python `f` correspondant à la fonction mathématique de même nom.
`f` doit être compatible avec les tableaux *numpy*, pour cela elle doit être issue de *numpy* ou définie avec des fonctions *numpy*.

La fonction `courbe_fonction1` qui prend en paramètres les flottants `a`, `b` et l'entier `n` affiche l'interpolation linéaire de $\mathscr{C}_f$ avec n points sur le segment $[a, b]$.

```
def courbe_fonction1(a,b,n):
    """affiche l'interpolation linéaire de Cf sur [a,b] avec n points"""
    x = np.linspace(a,b,n)
    plt.plot(x,f(x))
    plt.show()
```

La fonction `courbe_fonction2` qui prend en paramètres les flottants `a`, `b` et `h` affiche l'interpolation linéaire de $\mathscr{C}_f$ sur l'intervelle $[a, b[$ avec un pas de subdivision égal à `h`.

```
def courbe_fonction2(a,b,h):
    """affiche l'interpolation linéaire de Cf sur [a,b[ avec un pas de subdivision égal à h"""
    x = np.arange(a,b,h)
    plt.plot(x, f(x))
    plt.show()
```

Si la fonction f est strictement monotone alors par le théorème de la bijection elle réalise une bijection de $[a, b]$ sur $f([a, b])$. La courbe de f^{-1} sur $f([a, b])$ est donnée par la fonction suivante :

```
def courbe_reciproque(a,b,n):
    """affiche l'interpolation linéaire de Cf^(-1) sur f([a,b]) avec n points"""
    x = np.linspace(a,b,n)
    plt.plot(f(x),x)
    plt.show()
```

4 Options graphiques

Pour améliorer la lisibilité de la courbe d'une suite ou d'une fonction on peut adjoindre dans la figure courante des objets graphiques obtenus par des fonctions du module *matplotlib.pyplot* que l'on va décrire brièvement.

Commandes	Actions
`plt.plot(x,f(x),label="f")`	Crée la légende `f` pour la courbe obtenue par *plot*
`plt.legend()`	Fait ressortir les légendes dans la figure courante
`plt.grid()`	Crée un quadrillage
`plt.plot(X,Y,'+-r')`	Génère la courbe des points définis par les listes X et Y (abscisses et ordonnées) avec les options : • symbole : '.' point, 'o' rond, 'h' hexagone, '+' plus, 'x' croix, '*' étoile, ... • ligne : '-' trait plein, '- -' pointillé, '-.' alterné, ... • couleur : 'b' bleu , 'r' rouge , 'g' vert , 'c' cyan , 'm' magenta , 'k' noir, ...
`plt.axis('equal')`	Rend le repère orthonormé
`plt.xlim(xmin,xmax)`	Fixe les bornes de l'axe des abscisses
`plt.ylim(ymin,ymax)`	Fixe les bornes de l'axe des ordonnées

Les commandes suivantes sont moins importantes mais peuvent faire ressortir encore plus d'informations

Commandes	Actions
`plt.figure("Exercice 1 question 1")`	Crée la figure nommée *Exercice 1 question 1* qui accueillera les courbes
`plt.title("Titre de la figure")`	Crée le titre *Titre de la figure* pour la figure courante
`plt.xlabel("Titre axe des x")`	Crée le titre *Titre axe des x* pour l'axe des abscisses de la figure courante
`plt.xlabel("Titre axe des y")`	Crée le titre *Titre axe des y* pour l'axe des ordonnées de la figure courante
`plt.text(x,y,"Insertion")`	Insère le texte *Insertion* au point de coordonnées (x,y) de la figure courante
`plt.subplot(n,p,i)`	Crée une matrice de sous-figures à n lignes et p colonnes et choisit la $i^{\text{ème}}$