DS3

Correction du devoir surveillé

Exercice 1

1. Pour importer la librairie `numpy`, on peut utiliser au choix les instructions suivantes :

   `from numpy import *` ou `import numpy as np`

   On privilégiera comme d'habitude, et comme suggéré dans l'énoncé, la deuxième instruction.

2. On peut proposer le script suivant :

```
def suite(n):
    u = 0
    for k in range(n):
        u = np.cos(u)
    return u
```

3. Il nous faut déjà créer un vecteur `y` contenant tous les termes de la suite (u_n) pour $n = 0, \dots, 20$. Il restera alors à tracer les points de coordonnées (k, u_k) pour $k = 0, \dots, 20$, ce qu'on peut faire à l'aide de la commande `plot2d`. On peut utiliser le script suivant.

```
y = np.zeros(21)
for k in range(21): # k va de 0 à 20
    y[k] = suite(k)

x = np.arange(0,21)
plt.plot(x,y,'x')
plt.show()
```

   Le terme `'x'` permet de ne pas relier les points.

   À noter que la construction du vecteur `y` comme fait plus haut nécessite à `Python` beaucoup de calculs inutiles : on calcule tous les termes de la suite pour avoir le k-ième terme (c'est ce qui se passe avec l'instruction `suite(k)`), alors qu'on a déjà calculé à l'itération précédente le $(k-1)$-ième terme. Une méthode plus efficace est la suivante :

```
y = np.zeros(21)
for k in range(20):
    y[k+1] = np.cos(y[k])
```

4. On passe à la limite dans l'expression $u_{n+1} = \cos(u_n)$. Comme tout converge et que cos est une fonction continue, on obtient $\ell = \cos(\ell)$. Ainsi, ℓ est un point fixe de cos.

> **Déjà Vu ?**
>
> Il s'agit d'une suite récurrente d'ordre 1, du type $u_{n+1} = f(u_n)$ avec $f = \cos$. Nous avons déjà rencontré de telles suites à de nombreuses reprises (voir par exemple le TD0). Les méthodes d'études de telles suites sont détaillées dans le **Complément 0. Méthodes d'étude d'une suite récurrente d'ordre 1.**

5. On peut proposer le script suivant :

```
x = np.linspace(0,1,100)
y = np.cos(x)
plt.plot(x,x,'b') # 'b' pour couleur bleue
plt.plot(x,y,'r') # 'r' pour couleur rouge
plt.axis("scaled") # pour repère orthonormé
plt.grid() # pour grille
plt.show()
```

ℓ est l'abscisse du point d'intersection des courbes de $x \mapsto x$ et de $x \mapsto \cos(x)$ sur l'intervalle $[0,1]$ (qui est un intervalle stable par cos, et qui contient donc tous les termes de la suite u). Par lecture graphique, on obtient donc $\ell \approx 0,74$.

6. (a) On peut utiliser le script suivant :

```
def premier_entier(p):
    u = 0
    v = 1
    n = 0
    while np.abs(u-v)>10**(-p) :
        u = v
        v = cos(v)
        n = n+1
    return n
```

`u` et `v` contiennent respectivement les termes u_n et u_{n+1}, et on augmente n tant que $|u_n - u_{n+1}| > 10^{-p}$.

(b) En utilisant les fonctions définies plus haut, ainsi que l'inégalité :

$$|u_n - \ell| \leq |u_n - u_{n+1}| \leq 10^{-p},$$

le script suivant fait l'affaire :

```
def approx(p):
    n = premier_entier(p)
    l = suite(n)
    return l
```

7. (a) La fonction h est de classe $\mathscr{C}^1$ sur $[0,1]$ (différence de fonctions de classe $\mathscr{C}^1$), et on a :

$$\forall x \in [0,1], \quad h'(x) = 1 + \sin(x) > 0.$$

Donc h est **strictement décroissante** sur $[0,1]$. Comme de plus h est **continue** sur $[0,1]$ et que $h(0) = -1 < 0$ et $h(1) = 1 - \cos(1) > 0$, le théorème de la bijection assure l'existence d'un unique réel $\alpha \in [0,1]$ tel que $h(\alpha) = 0$. Et comme on a $h(\ell) = 0$, on en déduit que $\alpha = \ell$.

(b) Je vous renvoie à la fin de ce corrigé pour un rappel sur la méthode de dichotomie (voir aussi le corrigé du TP1). On peut alors compléter le script comme suit :

```
def dichotomie(eps):
      a = 0
      b = 1
      while np.abs(b-a)>eps :
            m = (a+b)/2
```

```
6            if h(a)*h(m)<=0 :
7                  b = m
8            else :
9                  a = m
10       return a
```

Exercice 2

(1) La commande `v=np.linspace(-3,17,5)` renvoie un vecteur contenant 5 valeurs équiréparties. Il y a donc 4 intervalles, chacun de longueur égale à $\frac{17+3}{4} = 5$, ce qui donne : `v=np.array([-3,2,7,12,17])`

La commande `v=np.arange(-3,18,5)` renvoie un vecteur dont les coefficients sont en progression arithmétique de raison 5, le premier étant -3 et le dernier ne dépassant pas 18. Cela donne : -3, -3 + 5, -3 + 10, -3 + 15, -3 + 20, ce qui donne effectivement le même vecteur. L'affirmation est donc vraie.

(2) L'instruction `u=np.arange(10,2,-1)` renvoie le vecteur `np.array([10,9,8,7,6,5,4,3])`. Il est formé de 8 valeurs équiréparties entre 10 et 3. Il y a donc bien un équivalent avec `np.linspace`, à savoir `u=np.linspace(10,3,8)`. L'affirmation est donc fausse.

(3) Attention, la multiplication employée ici s'effectue élément par élément. On obtient donc ici la matrice A. La matrice $\begin{pmatrix} 0 & 0 \\ 3 & 3 \end{pmatrix}$ est celle obtenue par le produit matriciel habituel. En `Python`, on obtient ce produit par l'instruction `np.dot(A,J)`. L'affirmation est donc fausse.

(4) L'instruction `A**J` s'effectue coefficient par coefficient, le coefficient (k, ℓ) de cette matrice étant $a_{k,\ell}^{j_{k,\ell}}$. Elle renvoie donc bien la matrice $\begin{pmatrix} 1 & -1 \\ 2 & 1 \end{pmatrix}$. L'affirmation est donc vraie.

(5) L'instruction `A==J` renvoie une **matrice** de booléens, dont le coefficient (k, ℓ) vaut `True` si $a_{k,\ell} = j_{k,\ell}$, `False` sinon. Ici elle renvoie donc $\begin{pmatrix} \texttt{True} & \texttt{False} \\ \texttt{False} & \texttt{True} \end{pmatrix}$. L'affirmation est donc fausse.

(6) `u` est un vecteur dont les coefficients sont 0, 1, 2 et 3. Chaque coefficient est accessible en écrivant `u[i]`, `i` désignant la position de ce coefficient. En `Python`, la numérotation positionnelle débute à 0. Ainsi, le coefficient 0 est désigné par `u[0]`, le coefficient 1 par `u[1]`, le coefficient 2 par `u[2]` et enfin le coefficient 3 par `u[3]`. Il n'y a pas de coefficient `u[4]` ! L'affirmation est donc fausse.

(7) Attention, il s'agit ici d'une boucle sans fin ! La valeur de `x` est inchangée, (égale à 2) tout au long du programme et le test `x<5` est toujours vrai. L'ordinateur affichera la valeur 2 indéfiniment ! Pour afficher les valeurs 2, 3 et 4, il fallait écrire la séquence :

```
1  x=2
2  while x<5 :
3       print(x)
4       x=x+1
```

L'affirmation est donc fausse.

(8) La variable `k` énumère la liste 0, 1. Au premier passage dans la boucle, quand `k` vaut 0, x contient à la fin de l'itération $2 \times 2 = 4$. Au deuxième passage dans la boucle, quand `k` vaut 1, x contient à la fin de l'itération $4 \times 4 = 16$. Ainsi, en sortie de boucle, la variable `x` contient bien la valeur 16. L'affirmation est donc vraie.

(9) La variable x contient le vecteur ligne en progression arithmétique `[1, 2, 3, ... , 10]`, la variable `y` le vecteur ligne de taille 1×10 contenant que des 1. La commande `np.dot(x,np.transpose(y))` correspond alors au produit matriciel :

$$\begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix} \times \begin{pmatrix}1\\2\\\vdots\\10\end{pmatrix} = 1+2+\cdots+10 = \frac{10\times 11}{2} = 55.$$

L'affirmation est donc vraie.

(10) C'est presque bon ! Malheureusement, le variable `x` proposée contient les valeurs (0, 1, 2, 3, 4). La représentation est donc une ligne brisée à 5 points, ce qui est très éloigné du dessin attendu. Pour avoir l'illusion d'une courbe, il fallait définir un vecteur `x` ayant un grand nombre de composantes. On peut proposer par exemple `x=np.arange(0,5,0.01)` ou `x=np.linspace(0,5,100)`. L'affirmation est donc fausse.

Rappels sur la méthode de dichotomie

Considérons une fonction $f : [a,b] \to \mathbb{R}$ **continue** et supposons que f **s'annule en une seule et unique valeur** $\alpha \in]a,b[$ et **y change de signe**. On ne peut pas forcément calculer explicitement α et on doit parfois utiliser des méthodes de calcul approché pour avoir une estimation de la valeur de α.

Une méthode bien connue est le principe de dichotomie. Comme f s'annule et change de signe, on a en particulier $f(a)f(b) < 0$.

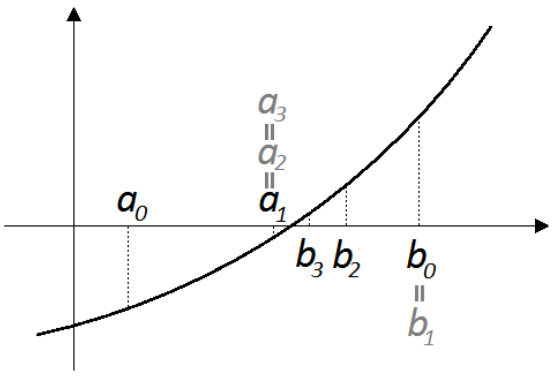
Le principe. On considère $m = \dfrac{a+b}{2}$ le milieu du segment $[a,b]$. Alors deux cas sont possibles :

- soit $f(a)f(m) \leq 0$: alors $f(a)$ et $f(m)$ sont de signe opposé. Comme f est continue, f doit s'annuler sur $[a,m]$ et donc $\alpha \in [a,m]$.
- soit $f(m)f(b) \leq 0$: alors $f(m)$ et $f(b)$ sont de signe opposé et de la même façon, $\alpha \in [m,b]$.

On recommence alors le raisonnement précédent avec l'intervalle obtenu contenant α.

On construit une suite d'intervalles $[a_n,b_n]$ contenant α : Pour cela, on définit trois suites $(a_n)_{n\in\mathbb{N}}$, $(b_n)_{n\in\mathbb{N}}$, $(m_n)_{n\in\mathbb{N}^*}$ par $a_0 = a$, $b_0 = b$ et pour tout $n \in \mathbb{N}$, $m_{n+1} = \dfrac{a_n+b_n}{2}$ et

- si $f(a_n)f(m_{n+1}) \leq 0$, alors $\begin{cases} a_{n+1} = a_n \\ b_{n+1} = m_{n+1} \end{cases}$
- si $f(m_{n+1})f(b_n) \leq 0$, alors $\begin{cases} a_{n+1} = m_{n+1} \\ b_{n+1} = b_n \end{cases}$



On montre alors que :

> **Propriété (Méthode de la dichotomie)**
> Les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ sont adjacentes et elles convergent vers l'unique zéro α de f sur $]a,b[$.

Preuve. Commençons pour cela par montrer par récurrence que pour tout $n \in \mathbb{N}$, on a :

$$a_n \leq a_{n+1} \leq b_{n+1} \leq b_n, \quad b_n - a_n = \frac{b-a}{2^n} \quad \text{et} \quad f(a_n)f(b_n) \leq 0.$$

Init. Par construction, on a soit $a_1 = a_0$ et $b_1 = \dfrac{a_0+b_0}{2}$, soit $a_1 = \dfrac{a_0+b_0}{2}$ et $b_1 = b_0$. Comme $a_0 = a < b = b_0$, on a ainsi l'inégalité $a_0 \leq a_1 \leq b_1 \leq b_0$. D'autre part, $b_0 - a_0 = b - a = \dfrac{b-a}{2^0}$ et $f(a_0)f(b_0) = f(a)f(b) \leq 0$ par hypothèse. Donc la propriété est vraie au rang $n = 0$.

Hér. Soit n un entier naturel. Supposons la propriété au rang n vraie. Montrons la propriété au rang $n+1$.

Par construction, on a soit $a_{n+2} = a_{n+1}$ et $b_{n+2} = \dfrac{a_{n+1}+b_{n+1}}{2}$, soit $a_{n+2} = \dfrac{a_{n+1}+b_{n+1}}{2}$. Par hypothèse de récurrence, $a_{n+1} \leq b_{n+1}$. On a donc dans les deux cas l'inégalité $a_{n+1} \leq a_{n+2} \leq b_{n+2} \leq b_{n+1}$.

De plus, par construction, on a $b_{n+1} - a_{n+1} = \dfrac{b_n - a_n}{2}$. Par hypothèse de récurrence, $b_n - a_n = \dfrac{b-a}{2^n}$. Donc $b_{n+1} - a_{n+1} = \dfrac{b-a}{2^{n+1}}$.

Enfin, comme par hypothèse de récurrence $f(a_n)f(b_n) \leq 0$, $f(a_n)$ et $f(b_n)$ sont de signe opposé. Donc $f(m_{n+1}) = f(\dfrac{a_n+b_n}{2})$ n'a pas le même signe que $f(a_n)$ ou $f(b_n)$. Par définition de a_{n+1} et de b_{n+1}, on a bien $f(a_{n+1})f(b_{n+1}) \leq 0$.

Donc la propriété au rang $n+1$ est vraie.

Par le principe de récurrence, on a bien pour tout $n \in \mathbb{N}$:

$$a_n \leq a_{n+1} \leq b_{n+1} \leq b_n, \quad b_n - a_n = \frac{b-a}{2^n} \quad \text{et} \quad f(a_n)f(b_n) \leq 0.$$

On en déduit donc que (a_n) est croissante, (b_n) est décroissante, et on a :

$$\forall n \in \mathbb{N}, \quad b_n - a_n = \frac{b-a}{2^n} \underset{n\to+\infty}{\longrightarrow} 0.$$

Les suites (a_n) et (b_n) sont donc adjacentes et, d'après le théorème des suites adjacentes, convergent vers la même limite $\ell \in]a, b[$.

Reste enfin à montrer que $\ell = \alpha$. En passant à la limite dans l'inégalité $f(a_n)f(b_n) \leq 0$, on obtient (en utilisant que f est continue sur $[a, b]$) :

$$\lim_{n\to+\infty} (f(a_n) \cdot f(b_n)) = \left(\lim_{n\to+\infty} f(a_n)\right)\left(\lim_{n\to+\infty} f(b_n)\right) = (f(\ell))^2 \leq 0.$$

D'où $(f(\ell))^2 \leq 0$, ce qui implique $f(\ell) = 0$. Et comme par hypothèse f s'annule en une seule valeur $\alpha \in]a, b[$, on a donc $\ell = \alpha$. □