Correction TP3 PCSI Python

Exercice 1

1)Soit $f : x \mapsto x^3 - 5$. On a : $f\left(\sqrt[3]{5}\right) = 0$, $f(1) = -4 < 0$ et $f(2) = 3 > 0$.
f est continue et strictement croissante sur [1,2], c'est donc une bijection de [1,2] sur [-4,3]. Elle ne s'annule donc qu'une fois en $\sqrt[3]{5}$.

```
def f(x):
    return x**3-5
a,b=1,2
epsilon=10**-6
while b-a>epsilon:
    m=(a+b)/2
    if f(a)*f(m)<=0:
        b=m
    else:
        a=m
print(a,b)
```

Python renvoie : 1.709975 et 1.709976. On a donc $1.709975 \leq \sqrt[3]{5} \leq 1.709976$.

2)Tableau :

boucle	m	$f(a)$	$f(m)$	intervalle contenant c	a	b	$b-a$
$\times\times\times\times$	$\times\times\times\times$	$\times\times\times\times$	$\times\times\times\times$	$\times\times\times\times$	1	2	1
1	3/2	-4	$-13/8$	$c \in [m,b]$ donc $a \longrightarrow m$	3/2	2	1/2
2	7/4	$-13/8$	23/64	$c \in [a,m]$ donc $m \longleftarrow b$	3/2	7/4	1/4

3)• Initialement, $b - a = 1$.
Après, chaque passage dans la boucle $b - a$ est réduit de moitié.
Donc après k itérations de la boucle, on a donc : $b - a = \left(\frac{1}{2}\right)^k$.
Comme à l'issue de ces k itérations, on sort de la boucle, c'est que la condition après `while` n'est plus réalisée donc que $\left(\frac{1}{2}\right)^k \leq 10^{-6}$.
Enfin, à l'issue de la $k-1$-ème itération, on a : $b - a = \left(\frac{1}{2}\right)^{k-1}$, mais on ne sort pas de la boucle.
La condition après `while` n'est donc plus réalisée donc $\left(\frac{1}{2}\right)^{k-1} > 10^{-6}$.

• On résout les inéquations :

$$\left(\frac{1}{2}\right)^k \leq 10^{-6} < \left(\frac{1}{2}\right)^{k-1} \iff \ln\left[\left(\frac{1}{2}\right)^k\right] \leq \ln 10^{-6} < \ln\left[\left(\frac{1}{2}\right)^{k-1}\right]$$

$\iff -k\ln 2 \leq -6\ln 10 < -(k-1)\ln 2 \iff \dfrac{6\ln 10}{\ln 2} \leq k < \dfrac{6\ln 10}{\ln 2} + 1$

$\iff 19.93 \leq k < 20.93.$

Donc k=20.

4)On complète le programme en introduisant un compteur c.

```
def f(x):
    return x**3-5
a,b,epsilon=1,2,10**-6
compteur=0
while b-a>epsilon:
    m=(a+b)/2
    if f(a)*f(m)<= 0:
        b = m
    else:
        a = m
    compteur=compteur+1
print(compteur)
```

Python affiche c=20.

Exercice 2

1)Programme :

```
def balayage(a,b,n):
    h=(b-a)/n
    x=a
    while f(x)*f(x+h)>0:
        x=x+h
    return x,x+h
```

2)On va considérer ici $f : x \mapsto x^3 - 5$ qu'on programme comme d'habitude :

```
def f(x):
    return x**3-5
```

Puis, on appelle la fonction balayage par la commande `balayage(1,2,10**6)`.

Elle retourne le couple de valeurs (1.7099749999415927, 1.7099759999415927).

On a donc $\sqrt[3]{5}$ = 1.709975 à 10^{-6} près.

En introduisant un compteur, on trouve que la boucle est réalisée 709975 fois, c'est beaucoup plus grand que pour la méthode de dichotomie !

Exercice 3

1)a)p est le nombre total d'itérations de la boucle. Donc u_p ne vérifie plus la condition de la boucle `while`, ce qui entraîne que u_p = 0.
On a alors : $u_p = u_{p-1}//2 = 0$, ce qui oblige u_{p-1} à valoir 0 ou 1.

Or, $u_{p-1} \neq 0$ sinon la boucle aurait été exécutée seulement $p-1$ fois.
Donc $u_{p-1} = 1$.

b)Soit $i \in [\![0, p-1]\!]$.
u_{i+1} est le quotient de la division euclidienne de u_i par 2.
On a donc $u_i = 2u_{i+1} + r_i$ ou encore $u_i - 2u_{i+1} = r_i$ avec $r_i \in \{0, 1\}$.
Donc $0 \leq u_i - 2u_{i+1} \leq 1$, puis en multipliant membre à membre par 2^i :

$$0 \leq 2^i u_i - 2^{i+1} u_{i+1} \leq 2^i \quad (*)$$

c)On somme les inégalités $(*)$ pour i allant de 0 à $p-2$:

$$0 \leq \sum_{i=0}^{p-2} \left(2^i u_i - 2^{i+1} u_{i+1}\right) \leq \sum_{i=0}^{p-2} 2^i$$

Par télescopage, $\sum_{i=0}^{p-2} \left(2^i u_i - 2^{i+1} u_{i+1}\right) = 2^0 u_0 - 2^{p-1} u_{p-1} = n - 2^{p-1}$.

Et $\sum_{i=0}^{p-2} 2^i = \dfrac{2^{p-1} - 1}{2 - 1} = 2^{p-1} - 1$.

En reportant la valeur de ces sommes, on a : $0 \leq n - 2^{p-1} \leq 2^{p-1} - 1$
D'où, $2^{p-1} \leq n \leq 2^p - 1$.

d)On déduit : $2^{p-1} \leq n < 2^p$, puis en appliquant le logarithme :
$\ln\left(2^{p-1}\right) \leq \ln(n) < \ln\left(2^p\right)$, c'est-à-dire : $(p-1)\ln(2) \leq \ln(n) < p\ln(2)$.
En divisant membre à membre par $\ln 2$, on conclut : $p - 1 \leq \log_2(n) < p$.
Cela entraîne que $p - 1 = \lfloor \log_2(n) \rfloor$, puis : $p = \lfloor \log_2(n) \rfloor + 1$.

> Remarque
> Le nombre p d'itérations de la boucle est un $O\big(\log_2(n)\big)$. La complexité du programme est donc logarithmique.

2)On introduit un compteur c dans la boucle en l'initialisant à 0.

```
def dicho(n):
    c=0
    while n>0:
        c=c+1
        n=n//2
    return c
```

La fonction retourne c=14, ce qui est conforme au résultat obtenu à la question 1)d).

En effet, d'après la question 1)d), le nombre total d'itérations vaut :

$$p = \lfloor \log_2(10000) \rfloor + 1 = \left\lfloor \frac{\ln(10000)}{\ln(2)} \right\rfloor + 1 \approx \lfloor 13,28 \rfloor + 1 = 14.$$

Exercice 4

1)Programme :

```
def recherche(L,x):
    g=0
    d=len(L)-1
    while g<=d:
        m=(g+d)//2
        if L[m]<x:
            g=m+1
        elif L[m]>x:
            d=m-1
        else:
            return m
    return -1
```

2)a)L=[1,3,7,12,15,20,30] et x=15.

boucle	m	$L[m]$	position de x	g	d
$\times\times\times$	$\times\times\times$	$\times\times\times$	$\times\times\times\times\times\times\times\times\times\times$	0	6
1	3	12	$L[m] < x$ donc $g \longrightarrow m+1$	4	6
2	5	20	$L[m] > x$ donc $d \longrightarrow m-1$	4	4
3	4	15	$L[m] = x$, on retourne 4	$\times\times\times$	$\times\times\times$

b)L=[2,3,5,5,8,10,15,20,30,40] et x=5

boucle	m	$L[m]$	position de x	g	d
$\times\times\times$	$\times\times\times$	$\times\times\times$	$\times\times\times\times\times\times\times\times\times\times$	0	9
1	4	8	$L[m] > x$ donc $d \longrightarrow m-1$	0	3
2	1	3	$L[m] < x$ donc $g \longrightarrow m+1$	2	3
3	2	5	$L[m] = x$, on retourne 2	$\times\times\times$	$\times\times\times$

c)L=[1,4,7,11,15,20,30,50] et x=10

boucle	m	$L[m]$	position de x	g	d
$\times\times\times$	$\times\times\times$	$\times\times\times$	$\times\times\times\times\times\times\times\times\times\times$	0	7
1	3	11	$L[m] > x$ donc $d \longrightarrow m-1$	0	2
2	1	4	$L[m] < x$ donc $g \longrightarrow m+1$	2	2
3	2	7	$L[m] < x$ donc $g \longrightarrow m+1$	3	2

Comme $g > d$, on sort de la boucle. La fonction renvoie -1.
Elle n'a pas trouvé la valeur 10 dans la liste.

3)La fonction retourne 2, ce n'est pas l'indice de la 1ère occurence de 7.

Exercice 5

1)a)Tout se passe bien avec les exemples de listes de l'exercice 4.

b)Prenons L=[1,1,1,1,1,1]. On a : len(L)=6.
Initialement, g=0 et d=5.
On rentre dans la boucle `while`, puis on a : m=(0+5)//2=2.
La condition `else`, c'est-à-dire L[m]=1 est réalisée.
On rentre dans la boucle `while`. On décrémente la variable m de 1, ce qui donne m=1, puis m=0, m=-1, ..., jusqu'à obtenir m=-7.
Lors de la 9ème itération, il est alors impossible à Python de définir L[-7], d'où l'erreur `list index out of range`.

c)Avec l'ajout de `while m>=0`, on n'a plus le problème précédent.
Après avoir trouvé l'indice d'une occurence de x, Python décrémente cet indice par l'instruction `m=m-1`, pour aller chercher la première occurence de x. Le nombre d'opérations effectuées peut être alors élevé si la valeur x figure un grand nombre de fois dans la liste.
Plus précisément, si x apparaît q fois dans la liste, la deuxième boucle peut effectuer jusqu'à q itérations. Dans le pire des cas, on a q=n, auquel cas, cette deuxième boucle peut être exécutée n fois. La complexité peut donc au pire des cas être linéaire, c'est-à-dire en O(n).

2)a)En remplaçant la condition `g<d` par `g<=d`, la boucle `while` pourrait continuer à l'infini.
Prenons par exemple : L=[3] et x=3. On a : g=d=0, puis m=0. La condition `L[m]>=x` est réalisée donc d=m=0. Et on refait la boucle à l'infini !

b)Supposons que x∈L et soit i l'indice de sa première occurence.
b1)Initialement, on a : g=0 et d=n-1 où n=len(L). Comme x∈L, tout indice d'une occurence de x est dans [g,d]. A fortiori, i∈[g,d].
b2)Supposons qu'au début d'une itération, on ait $i \in [g,d]$.
Considérons 2 cas :

• L[m]<x
Comme x=L[i], on a : $L[m] < L[i]$, et la liste L étant croissante (au sens large), on a alors : $m < i$. Par hypothèse, $i \leq d$ donc $i \in [m+1, d]$.
Enfin, quand $L[m] < x$, le test fait : g=m+1. Donc $i \in [g,d]$.

• L[m]≥x
Distinguons deux sous-cas :
– si L[m]>x
Comme x=L[i], on a : $L[m] > L[i]$, et la liste L étant croissante (au sens large), on a alors : $m > i$.
– si L[m]=x
Comme x=L[i], on a : $L[m] = L[i] = x$.
i étant la première occurrence de x, on a : i≤ m.
Par hypothèse, i≥ g donc $g \leq i \leq m$, soit : i∈ $[g,m]$.
Enfin, quand $L[m] = x$, le programme fait : d = m. Donc $i \in [g,d]$.

Remarque
Par une récurrence immédiate, on justifie ensuite que $i \in [g, d]$ à n'importe quel moment de la boucle.

c)A l'entrée de la boucle, on a toujours $g < d$ donc $g \leq m < d$.

En effet, si $g < d$, on a : $g < \dfrac{g+d}{2} < d$, puis $g \leq \left\lfloor \dfrac{g+d}{2} \right\rfloor = m < d$.

On peut ensuite avoir $g = m + 1$ ou $d = m$, ce qui conduit toujours dans les 2 cas à $g \leq d$.
La sortie de définitive de la boucle ne peut donc se faire que si $g = d$.

d)Supposons que x∈L et soit i l'indice de la première occurence de x.
D'après la question 2)b), on a : i∈[g,d].
D'après la question 2)c), en sortant de la boucle, on a : g=d. Et donc i=g=d.
La fonction retourne alors g, c'est-à-dire i.

e)Si L[m]= x et que m se trouve être l'indice de la première occurence de x, on a alors : L[m-1]<x. Ecrire l'instruction d=m-1 entrainerait que l'indice m de la première occurence de x n'est plus dans l'intervalle [g,d], ce qui contredit la question 2)b).

f)On prend L=[0,0,0,1,1,1,1,1,2,2,2,2,6,9] et x=2.

boucle	m	$L[m]$	position de x	g	d
× × ×	× × ×	× × ×	× × × × × × × × × ×	0	13
1	6	1	$L[m] < x$ donc $g \longrightarrow m+1$	7	13
2	10	2	$L[m] \geq x$ donc $m \longleftarrow d$	7	10
3	8	2	$L[m] \geq x$ donc $m \longleftarrow d$	7	8
4	7	1	$L[m] < x$ donc $g \longrightarrow m+1$	8	8

On sort de la boucle, puisque g=d. Et la fonction retourne 8.

Exercice 6
1)Le programme devine le nombre qu'on a choisi au préalable.

2)On choisit le nombre 60.

a)Le programme comporte 6 itérations.

b)On commence par importer la fonction `randint` par la commande :
`from numpy.random import randint`.
Puis, on remplace l'instruction `m=(debut+fin)//2` par : `m=randint(debut,fin+1)`.
Et on exécute le programme...

Le nombre d'essais nécessaires est variable. Il peut être inférieur à 6. Mais dans beaucoup de cas, il est supérieur à 6 ! Dans le pire des cas, il peut même être nécessaire d'effectuer 100 essais pour trouver le nombre 60 (avec certes, une suite de tirages particulièrement malchanceuse !)

Exercice 7

1)a)Programme :

```
def communs1(L,M):
    N=[]
    for x in M:
        for y in L:
            if x==y:
                N.append(x)
    return N
```

b)Pour chacun des p éléments de M, on parcourt n éléments de L. La comparaison `x==y` est donc faite np fois.
La fonction `communs1` a donc une complexité en $O(np)$.

2)a)Programme :

```
def communs2(L,M):
    N=[]
    for x in M:
        if recherche(L,x)!=-1:
            N.append(x)
    return N
```

b)Pour chacun des p éléments de M, il faut lui appliquer la fonction `recherche` qui a une complexité en $O\big(\ln(n)\big)$.
La fonction `communs2` a donc une complexité en $O\big(p\ln(n)\big)$.

3)a)Programme :

```
def communs3(L,M):
    N=[]
    i=0
    j=0
    while i<len(L) and j<len(M):
        if L[i]<M[j]:
            i=i+1
        elif L[i]>M[j]:
            j=j+1
        else:
            N.append(L[i])
            i=i+1
            j=j+1
    return N
```

b)A chaque itération, au moins un des deux indices i et j augmente de 1. Comme i ne peut augmenter que n fois et j que p fois, le nombre total

d'itérations est au plus égal à n+p.
Donc la fonction `communs3` a une complexité en $O(n+p)$.

4)Les complexités des fonctions sont respectivement en $O(np)$, $O\big(p\ln(n)\big)$ et $O(n+p)$. En terme de coût, c'est la fonction qui a la complexité la plus faible qui est la plus avantageuse.
On ne peut pas ici donner de réponse tranchée car cela dépend de la valeur de n et de p.

Par exemple, si n et p ont même ordre de grandeur, les complexités respectives seront en $O(n^2)$, $O\big(n\ln(n)\big)$ et $O(2n)$, soit $O(n)$.
C'est alors la fonction 3 qui est la moins coûteuse, puis la fonction 2, puis la fonction 1.
En revanche, si p est négligeable devant n et n grand, c'est la deuxième fonction qui est la moins coûteuse.