

Python : révisions - Corrigé

Exercice 1 (Suites explicites)

```
1. import numpy as np
def vecteur(n) :
    A = np.arange(n) # A = [0,1, ..., n-1]
    V = A / (1 + A**2)
    return V
```

ou alors :

```
import numpy as np
def vecteur(n) :
    return np.array([ k / (1+k**2) for k in range(n) ])
```

```
2. import numpy as np
def vecteur(n) :
    A = np.arange(1,n+1) # A = [1,2, ..., n]
    V = np.log(A)/A
    return V
```

ou alors :

```
import numpy as np
def vecteur(n) :
    return np.array([ np.log(k) / k for k in range(1,n+1) ])
```

Exercice 2 (Suite à récurrence double)

```
import numpy as np
def vecteur(n) :
    V = np.zeros(n) ; V[0] = 0 ; V[1] = 1
    for k in range(n-2) : # k = 0, 1, ..., n-3
        V[k+2] = (k+1)/(V[k] + V[k+1])
```

Exercice 3 (Paradoxe des anniversaires)

- On considère que chacune des n dates d'anniversaire est uniformément distribuée sur les 365 jours de l'année. On peut décrire l'univers associé à cette expérience aléatoire :

$$\Omega = \llbracket 1, 365 \rrbracket^n = \{(x_1, x_2, \dots, x_n), \forall i \in \llbracket 1, n \rrbracket, x_i \in \llbracket 1, 365 \rrbracket\}$$

(x_i est la date d'anniversaire de la personne $n^\circ i$)

On est alors en situation d'équiprobabilité !

En notant A = "Au moins deux personnes ont la même date d'anniversaire", on a donc :

$$P(A) = 1 - P(\bar{A}) = 1 - \frac{\text{Nb de résultats où toutes les dates sont différentes}}{\text{Nb total de résultats}}$$

c'est à dire, avec un peu de dénombrement :

$$P(A) = 1 - \frac{365 \times 364 \times \dots \times (365 - n + 1)}{365^n} = 1 - \frac{365!}{(365 - n)! \times 365^n}.$$

- Soit $n \in \llbracket 1, 364 \rrbracket$. On a :

$$1 - (1 - p_n) \left(1 - \frac{n}{365} \right) = 1 - \frac{365!}{(365 - n)! \times 365^n} \times \frac{365 - n}{365} = 1 - \frac{365!}{(365 - n - 1)! \times 365^{n+1}} = p_{n+1}$$

- Notons que $P[0] = p_1 = 0$. De plus $P[k] = p_{k+1}$ donc la relation $p_{k+1} = 1 - (1 - p_k)(1 - \frac{k}{365})$ s'écrit :
 $P[k] = 1 - (1 - P[k-1]) \left(1 - \frac{k}{365} \right)$

```
import numpy as np
P = np.zeros(365); # On peut ajouter P[0] = 0, mais ce n'est pas utile
for k in range(1,365) : # k = 1,2, ..., 364
    P[k] = 1 - (1-P[k-1]) * (1 - k/365)
print(P)
```

Exercice 4 (Somme basique)

1. Avec une boucle :

```
def somme(n) :  
    S = 0  
    for k in range(1,n+1) :  
        S = S + (-1)**k / k # ou bien S += (-1)**k / k  
    return S
```

Sans boucle, avec `np.sum` :

```
import numpy as np  
def somme(n) :  
    return np.sum( [ (-1)**k / k for k in range(1,n+1) ] )
```

ou, si on préfère construire un vecteur :

```
import numpy as np  
def somme(n) :  
    A = np.arange(1,n+1)  
    V = (-1)**A / A  
    return np.sum(V)
```

2. On a pour tout $k \in \llbracket 0, n-1 \rrbracket$, $V[k] = S_{k+1}$. Bien-sûr, on pourrait poser simplement :

```
import numpy as np  
def vecsomme(n) :  
    V = np.zeros(n);  
    for k in range(n) : # k = 0, 2, ..., n-1  
        V[k] = somme(k+1)  
    return V
```

Mais ceci est très lourd en calcul, puisque pour remplir chacune des cases du vecteur V on recalcule la somme depuis le début ! Il vaut mieux mettre à profit la relation de récurrence :

$$V[k] = S_{k+1} = S_k + \frac{(-1)^{k+1}}{k+1} = V[k-1] + \frac{(-1)^{k+1}}{k+1}.$$

Notons de plus que $V[0] = S_1 = -1$. Cela conduit au programme suivant :

```
import numpy as np  
def vecsomme(n) :  
    V = np.zeros(n); V[0] = -1  
    for k in range(1,n) : # k = 1, 2, ..., n-1  
        V[k] = V[k-1] + (-1)**(k+1) / (k+1)  
    return V
```

Exercice 5 (Coefficients binomiaux)

1. (a) $\binom{n}{p} = \frac{n!}{p!(n-p)!}$. (b)

```
import numpy as np  
def binome(p,n) :  
    fact1 = np.prod( range(1, n+1) ) # calcule n!  
    fact2 = np.prod( range(1, p+1) ) # calcule p!  
    fact3 = np.prod( range(1, n-p+1) ) # calcule (n-p)!  
    return fact1 / (fact2 * fact3)
```

Ce programme est très coûteux en calculs car, pour calculer $\binom{100}{3}$, il demande d'effectuer $3 + 100 + 97 = 200$ multiplications. On peut faire bien-mieux !

2. (a) On peut aussi écrire : $\binom{n}{p} = \frac{n \times (n-1) \times \dots \times (n-p+1)}{p \times (p-1) \times \dots \times 1} = \prod_{k=0}^{p-1} \frac{n-k}{p-k}$.

(b) Il suffit de calculer ce produit de p termes en Python, par exemple avec `np.prod` :

```
def binome(p,n) :  
    b = np.prod( [ (n-k)/(p-k) for k in range(p) ] )  
    return b
```

Dans ce cas, le calcul de $\binom{100}{3}$ requiert seulement 3 produits !

Exercice 6 (Sommes exploitant une relation de récurrence)

1. L'idée est de remarquer que l'on veut calculer la somme $\sum_{k=0}^n a_k$ avec $a_0 = 1$ et la relation de récurrence $a_k = \frac{x}{k} \times a_{k-1}$. Ainsi :

```
def somme(x,n) :  
    a = 1 ; S = a  
    for k in range(1, n+1) : # k = 1, ... , n  
        a = (x / k) * a  
        S = S + a  
    return S
```

Il est important de s'assurer du bon fonctionnement de ce programme en constatant les valeurs contenues dans les variables après chaque passage de la boucle :

Avant la boucle : $a = 1, S = 1$

Après 1 passage : $k = 1, a = x, S = 1 + x$

Après 2 passages : $k = 2, a = \frac{x^2}{2}, S = 1 + x + \frac{x^2}{2}$

Après 3 passages : $k = 3, a = \frac{x^3}{6}, S = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$

\vdots

Après n passages : $k = n, a = \frac{x^n}{n!}, S = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots + \frac{x^n}{n!}$

2. Même chose pour la somme $\sum_{k=0}^n a_k$ avec $a_0 = 1$ et la relation de récurrence $a_k = \frac{-x^2}{(2k)(2k-1)} \times a_{k-1}$.

```
def somme(x,n) :  
    a = 1 ; S = a  
    for k in range(1, n+1) : # k = 1, ... , n  
        a = (-x / ( (2*k)*(2*k-1) ) ) * a  
        S = S + a  
    return S
```

Exercice 7 (Matrice en forme de N)

1.

```
import numpy as np  
N = np.array( [ [1,0,1], [1,1,1], [1,0,1] ] )
```

2.

```
def matriceN(n) :  
    A = np.eye(n) # matrice identite de taille n : des 1 sur la diagonale  
    A[:,0] = 1 # on ajoute des 1 sur la premiere colonne  
    A[:,n-1] = 1 # on ajoute des 1 sur la derniere colonne  
    return A
```

Exercice 8 (Matrice en forme de Z)

```
def matriceZ(n) :  
  
    A = np.zeros( (n,n) ) # matrice nulle de taille n x n  
  
    A[0,:] = 1 # on ajoute des 1 sur la premiere ligne  
    A[n-1,:] = 1 # on ajoute des 1 sur la derniere ligne  
  
    # On remplit ensuite "l'anti-diagonale" avec des 1 :  
    for k in range(n) : # k = 0, 1, ..., n-1  
        A[k,n-1-k] = 1  
  
    return A
```

Exercice 9 (Approximation de π)

Notons que la fonction $f : x \mapsto \cos(\frac{x}{2})$ est continue, strictement décroissante sur $[0, 4]$, avec $f(0) = \cos(0) = 1$ et $f(4) = \cos(2) < 0$. On peut lui appliquer la méthode de dichotomie pour déterminer une valeur approchée du point où f s'annule (a.k.a π).

Attention, comme on est dans le cas d'une fonction d'abord positive, ensuite négative, il faut "décaler" la bonne variable lorsque $f(c) > 0$... Un dessin peut aider à s'y retrouver.

On veut une valeur approchée à 10^{-5} près : c'est ce qui jouera le rôle du ε .

Enfin, on veut afficher une seule valeur et non pas un encadrement. On peut donc afficher a ou b , au choix !

```
import numpy as np

a = 0; b = 4 # initialisations de a et b

while b-a > 10**(-5) : # tant que a et b sont distants d'au moins 10**(-5)

    c = (a+b)/2 # calcul du point milieu

    if np.cos(c/2) > 0 : # si la fonction est positive au point c
        a = c # on decale le a
    else :
        b = c # on decale le b

print(a) # a la fin de ces operations, on renvoie a (ou b)
```

On peut aussi, si on préfère, définir au préalable la fonction $f : x \mapsto \cos(x/2)$ en Python, puis l'utiliser dans le script précédent.

Exercice 10 (Approximation de $\sqrt{2}$)

Première façon : méthode de Dichotomie

On exploite le fait que $\sqrt{2}$ est l'unique solution de l'équation $x^2 = 2$ sur le segment $[0, 2]$ par exemple.

On applique l'algorithme de dichotomie ! Cette fois la fonction $x \mapsto x^2$ est croissante : d'abord en dessous de 2, puis au dessus de 2. Un dessin peut aider à savoir quelle variable il faut "décaler" dans chacun des cas.

On veut une valeur approchée à 10^{-5} près : c'est ce qui jouera le rôle du ε .

Enfin, on veut afficher une seule valeur et non pas un encadrement. On peut donc afficher a ou b , au choix !

```
a = 0; b = 2 # initialisations de a et b

while b-a > 10**(-5) : # tant que a et b sont distants d'au moins 10**(-5)

    c = (a+b)/2 # calcul du point milieu

    if c**2 < 2 : # si la fonction est en dessous de 2 au point c
        a = c # on decale le a
    else :
        b = c # on decale le b

print(a) # a la fin de ces operations, on renvoie a (ou b)
```

Deuxième façon : méthode du point fixe

On définit la fonction $g : x \mapsto \frac{2}{3} \left(x + \frac{1}{x} \right)$ ainsi qu'une suite $u : u_0 > \sqrt{2}$, et $\forall n \in \mathbb{N}, u_{n+1} = g(u_n)$.

- (a) Pour tout $x > 0$, on a les équivalences :

$$g(x) = x \iff \frac{2}{3} \left(x + \frac{1}{x} \right) = x \iff \frac{2}{3} (x^2 + 1) = x^2 \iff \frac{1}{3} x^2 - \frac{2}{3} = 0 \iff x^2 = 2 \iff x = \sqrt{2}.$$

L'unique point fixe de g sur \mathbb{R}_+^* est donc $\sqrt{2}$.

- (b) Etude rapide de la fonction g , dérivable sur \mathbb{R}_+^* comme somme de fonctions usuelles :

$$\forall x > 0, g'(x) = \frac{2}{3} \left(1 - \frac{1}{x^2} \right)$$

donc $g'(x) > 0 \iff x > 1$. On obtient donc le tableau de variations :

x	0	1	$+\infty$
$g(x)$	$+\infty$	$4/3$	$+\infty$

Puisque $\sqrt{2} > 1$, on a en particulier : $g(]\sqrt{2}, +\infty[) =]g(\sqrt{2}), +\infty[=]\sqrt{2}, +\infty[$.

Ainsi, l'intervalle $]\sqrt{2}, +\infty[$ est stable par g , c'est à dire que : $\forall x > \sqrt{2}, g(x) > \sqrt{2}$.

Puisqu'on a choisi justement $u_0 > \sqrt{2}$, par récurrence immédiate il en résulte que : $\forall n \in \mathbb{N}, u_n > \sqrt{2}$.

(c) Soit $n \in \mathbb{N}$. Etudions le signe de $u_{n+1} - u_n$:

$$u_{n+1} - u_n = \frac{2}{3} \left(u_n + \frac{1}{u_n} \right) - u_n = -\frac{1}{3}u_n + \frac{2}{3} \frac{1}{u_n} = \frac{2 - u_n^2}{3u_n} < 0$$

car $u_n > \sqrt{2}$. Il en résulte bien que la suite u est strictement décroissante.

Conclusion : La suite u est décroissante et minorée par $\sqrt{2}$, donc d'après le théorème de la limite monotone, elle converge vers un réel $\ell \geq \sqrt{2}$. En passant à la limite dans l'égalité $u_{n+1} = g(u_n)$, on obtient $\ell = g(\ell)$. Ainsi, ℓ doit être un point fixe de g , c'est à dire forcément $\ell = \sqrt{2}$.

2. (a) Pour tout $n \in \mathbb{N}$, on sait déjà que $u_n - \sqrt{2} > 0$. De plus, on a

$$u_n^2 - 2 = \underbrace{(u_n - \sqrt{2})}_{>0} \underbrace{(u_n + \sqrt{2})}_{>\sqrt{2}} \geq (u_n - \sqrt{2})(\sqrt{2} + \sqrt{2}) \geq (u_n - \sqrt{2}) \times 2$$

donc effectivement, $u_n - \sqrt{2} \leq \frac{1}{2} (u_n^2 - 2)$.

(b) Pour approcher $\sqrt{2}$ à 10^{-5} près, l'idée est de calculer la valeur de u_n jusqu'à ce que :

$$\frac{1}{2}(u_n^2 - 2) \leq 10^{-5}.$$

L'encadrement précédent garantit alors que $0 < u_n - \sqrt{2} \leq 10^{-5}$, c'est à dire que $\sqrt{2} < u_n < \sqrt{2} + 10^{-5}$: u_n sera proche de $\sqrt{2}$ à 10^{-5} près !

On peut choisir la valeur que l'on souhaite pour u_0 , du moment que $u_0 > \sqrt{2}$. Bien-sûr, on gagne à prendre une valeur déjà "proche" de $\sqrt{2}$, par exemple 2.

```
u = 2 # initialisation
while 0.5 * (u**2 - 2) > 10**(-5) :
    u = (2/3) * (u + 1/u) # on applique la relation de recurrence
print(u) # apres tous ces passages de boucle, on affiche u
```

3. (a) Pour tout $x > 1$, on a

$$|g'(x)| = \left| \frac{2}{3} \left(1 - \frac{1}{x^2} \right) \right| = \frac{2}{3} \left(1 - \frac{1}{x^2} \right) \leq \frac{2}{3};$$

On peut donc choisir $M = \frac{2}{3} \in]0, 1[$.

(b) On note que pour tout $n \in \mathbb{N}$, on a : $u_{n+1} - \sqrt{2} = g(u_n) - g(\sqrt{2})$

Or, puisque $\forall t \in]\sqrt{2}, u_n[$, $|g'(t)| \leq M$, l'IAF nous apprend que

$$|g(u_n) - g(\sqrt{2})| \leq M|u_n - \sqrt{2}| \quad \text{i.e.} \quad |u_{n+1} - \sqrt{2}| \leq M|u_n - \sqrt{2}|.$$

Puisque $u_n - \sqrt{2} > 0$, les valeurs absolues sont inutiles : on obtient $u_{n+1} - \sqrt{2} \leq M(u_n - \sqrt{2})$

De là, une récurrence immédiate conduit à : $\forall n \in \mathbb{N}, u_n - \sqrt{2} \leq M^n(u_0 - \sqrt{2})$.

(c) On pose $u_0 = 2$, donc on a $u_0 - \sqrt{2} = 2 - \sqrt{2} \leq 1$ de sorte que : $\forall n \in \mathbb{N}, 0 < u_n - \sqrt{2} \leq M^n$.

Pour que u_n soit une approximation de $\sqrt{2}$ à 10^{-5} près, il suffit donc qu'on ait $M^n \leq 10^{-5}$, i.e :

$$\left(\frac{2}{3} \right)^n \leq 10^{-5} \iff n \ln(2/3) \leq -5 \ln(10) \iff n \geq \frac{-5 \ln(10)}{\ln(2/3)} = \iff n \geq \frac{5 \ln(10)}{\ln(3) - \ln(2)} \simeq 28,4.$$

En choisissant $n = 29$, on est donc certain que $u_n = u_{29}$ est une approximation de $\sqrt{2}$ à 10^{-5} près !
On peut alors se contenter du script suivant ; où l'on fait 13 passages de boucle :

```
u = 2 # initialisation
for k in range(29) : # 29 passages
    u = (2/3) * (u + 1/u) # on applique la relation de recurrence
print(u)
```