Correction - DS 6

# Devoir surveillé du Mardi 7 Janvier

## Exercice 1

1. On peut procéder comme suit :

```
#définition de la fonction f
def f(x):
    return x**2-2

#tracé de la courbe de f
x = np.linspace(1, 2, 100)
y = f(x)
plt.plot(x, y)
plt.show()
```

2. La fonction `balayage` prend en entrée un entier $p$. Elle crée un vecteur ligne `v` de taille $10^p + 1$ dont les coefficients sont en progression arithmétique de 1 à 2. Ainsi pour $p = 3$, `v` contient le vecteur $[1, 1.001, 1.002, \ldots, 1.999, 2]$. Cette fonction parcourt ensuite le vecteur `v` à l'aide d'une boucle `while`, avec pour condition d'arrêt `f(v[i])<0`. Ainsi, si la boucle `while` s'arrête au rang `i`, ce rang satisfait `f(v[i])>=0` et `f(v[i-1])<0`. Puisque $f$ est croissante, et s'annule en changeant de signe en $\sqrt{2}$, il suit :

$$v[i-1] = v[i] - 10^{-p} < \sqrt{2} \leq v[i].$$

En renvoyant `v[i]`, la fonction `balayage` fournit donc une approximation de $\sqrt{2}$ à $10^{-p}$ près.

3. On peut procéder comme suit

```
def suites(n) :
      a = 1
      b = 2
      for k in range(n) :
            m = (a+b)/2
            if f(a)*f(m)<=0 :
                  b = m
            else :
                  a = m
      return a, b
```

On passe $n$ fois dans la boucle `for`. À chaque passage dans cette boucle, on calcule $m$, on teste si $f(a) * f(m) \leq 0$. Si c'est le cas, la variable `a` reste inchangée et `b` reçoit `m`. Sinon, c'est la variable `b` qui reste inchangée et `a` qui reçoit `m`.

4. Créons déjà deux vecteurs `a` et `b` contenant tous les termes des suites $(a_n)$ et $(b_n)$ pour $n = 0, \ldots, 10$. Pour cela, on initialise ces vecteurs avec que des coefficients nuls. Puis, à l'aide d'une boucle `for` et de la fonction `suites`, on remplace à chaque itération le $k$-ème coefficient de ces vecteurs par le terme $a_k$ et $b_k$.

```
a = np.zeros(11)
b = np.zeros(11)
for k in range(11) :
    a[k], b[k] = suites(k)
```

Il reste alors à tracer les points de coordonnées $(k, a_k)$ et $(k, b_k)$ pour $k = 0, \dots, 10$, ce qu'on peut faire à l'aide de la commande `plot`. Sans oublier de créer au préalable un vecteur `x` contenant les abscisses de ces points.

```
x = np.arange(11)
plt.plot(x, a, 'x')
plt.plot(x, b, '+')
plt.legend(['suite (a_n)', 'suite (b_n)']) # pour afficher une légende
plt.show()
```

Les arguments `'x'` et `'+'` permettent de ne pas relier les points et de pouvoir distinguer ceux correspondants à la suite $(a_n)$ ou à $(b_n)$.

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

```
a = np.zeros(11) ; a[0] = 1
b = np.zeros(11) ; b[0] = 2
for k in range(1,11):
    m = (a[k-1]+b[k-1])/2
    if f(a[k-1])*f(m) <= 0 :
        b[k] = m
    else :
        a[k] = m
```

5. Puisque $|\sqrt{2} - a_n| \leq |b_n - a_n|$, il suffit de trouver un rang $n$ pour lequel $|b_n - a_n| \leq \varepsilon$, et de renvoyer $a_n$ qui fournira alors une approximation de $\sqrt{2}$ à $\varepsilon$ près. On va pour cela reprendre en partie le programme précédent. On initialise les variables `a` à $a_0 = 1$ et `b` à $b_0 = 2$. Et tant que $|b_n - a_n| > \varepsilon$, on passe au rang suivant. On va donc effectuer une boucle `while`. Puisque les suites $(a_n)$ et $(b_n)$ sont adjacentes, on sait que cette boucle s'arrêtera bien à un certain rang.

   Voici une possibilité de script.

```
def dichotomie(eps):
      a = 1
      b = 2
      while np.abs(b-a) >= eps :
            m = (a+b)/2
            if f(a)*f(m)<=0 :
                  b = m
            else :
                  a = m
      return a
```

6. On peut procéder ainsi :

```
def suite(n):
    x = 1
    for k in range(n):
        x = (x+2/x)/2
    return x
```

7. Créons déjà un vecteur `y` contenant tous les termes de la suite $(x_n)$ pour $n = 0, \dots, 10$. Il restera alors à tracer les points de coordonnées $(k, x_k)$ pour $k = 0, \dots, 10$, ce qu'on peut faire à l'aide de la commande `plot`. On peut utiliser le script suivant.

```
y = np.zeros(11)
for k in range(11):
    y[k] = suite(k)

x = np.arange(0,11)
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.ones(10)
y[0] = 1
for k in range(1,10):
    y[k] = (y[k-1]+2/y[k-1])/2
```

8. La suite $(x_n)$ est une suite récurrente d'ordre 1. Puisqu'elle converge vers une limite finie $\ell \in [1, 2]$, $\ell$ est un point fixe de $g$. Résolvons :

$$g(\ell) = \ell \Leftrightarrow \ell = \frac{2}{\ell} \Leftrightarrow \ell^2 = 2 \underbrace{\Leftrightarrow}_{\ell>0} \ell = \sqrt{2}$$

9. On peut procéder ainsi.

```
def newton(eps):
    n = 0
    while 3*(1/2)**(2**n) > eps :
        n = n+1
    return(suite(n))
```

---

## Exercice 2

1. On a une probabilité de $\dfrac{x}{x+y}$ de tirer une boule rouge. Donc le programme doit retourner 0 avec une probabilité $\dfrac{x}{x+y}$. Il faut donc que la condition soit satisfaite avec cette probabilité. On va pour cela prendre comme condition $r \leq \dfrac{x}{x+y}$.

```
def tirage(x, y):
    r = rd.random()
    if r <= x/(x+y) :
       return(0)
    else:
       return(1)
```

2. Si $r = 0$ alors on ajoute une rouge sinon on ajoute une blanche, d'où `x = x+1` ou `y = y+1` ; le nombre de rouges ajoutées est donc la différence entre le nombre de boules rouges $x$ au final et le nombre de boules rouges $a$ au départ, soit `Xn = x-a`. Ainsi on a :

```
1  def experience(a, b, n):
2      x = a
3      y = b
4      for k in range(n):
5          r = tirage(x, y) #r = 0 si rouge au tirage k, r = 1 si
   blanche
6          if r == 0 :
7             x = x+1
8          else:
9             y = y+1
10     return(x-a)
```

3. Voici le programme demandé :

```
1  def simulation(a, b, n, m):
2      loi = np.zeros(n+1) #pour stocker les fréquences
3      for k in range(m):
4          r = experience(a, b, n) #valeur de Xn à la k-ième ré
   alisation
5          loi[r] = loi[r]+1 #on augmente l'effectif de Xn = r de 1
6      return(loi/m) #on divise par l'effectif total pour obtenir la
   fré quence
```

4. La distribution des fréquences semble équiprobable, donc on peut conjecturer que $X_n \hookrightarrow \mathcal{U}(\llbracket 0, n \rrbracket)$.

---

## Exercice 3

1. (a) Pour tout $i \in \llbracket 0, N \rrbracket$ et pour tout $k \in \mathbb{N}$, si $(X_k = i)$ est réalisé, alors la population totale se partage en deux catégories : ceux en faveur de $A$, qui sont en proportion $i/N$, et ceux en faveur de $B$, qui sont en proportion $1 - i/N$. Dans ces conditions, la variable $X_{k+1}$ compte le nombre de succès (être en faveur de $A$, de probabilité $i/N$) au cours de $N$ épreuves de Bernoulli identiques et indépendantes (associées aux $N$ personnes interrogées). On en déduit que la loi conditionnelle de $X_{k+1}$ sachant que $(X_k = i)$ est réalisé, est la loi binomiale $\mathcal{B}\left(N, \dfrac{i}{N}\right)$.
   On a donc, pour tout $j \in \llbracket 0, N \rrbracket$ :

$$P_{(X_k=i)}(X_{k+1} = j) = \binom{N}{j} \left(\frac{i}{N}\right)^j \left(1 - \frac{i}{N}\right)^{N-j}.$$

   (b) Il s'agit de compléter le programme de sorte qu'il simule la chaîne de Markov. La variable `X` va contenir au fur et mesure les valeurs $X_0, X_1, \ldots, X_k$ et la liste `L` va permettre d'enregistrer ces valeurs. Avec la question précédente, on décide donc de compléter la ligne 5 avec `X = rd.binomial(N, X/N)`.

   (c) Apparemment, pour une valeur de $n_0$ proche de 0 (comme $n_0 = 3$), le nombre d'électeurs en faveur de $A$ tend à se stabiliser à 0 (et une fois 0 atteint, il est logique que la chaine stationne à cet état puisque personne, dans la population totale, ne vote pour $A$, et donc a fortiori dans un échantillon de cette population), bien qu'il ne soit pas impossible qu'il atteigne la valeur maximal 10 et qu'il y demeure pour des raisons similaires.

De même, pour une valeur de $n_0$ proche de $N$ (comme $n_0 = 8$, avec $N = 10$), le nombre d'électeurs en faveur de $A$ tend à se stabiliser à 10, sans qu'il soit impossible qu'il atteigne 0.

Et pour une valeur intermédiaire, le comportement est erratique : au bout de 20 sondages, le nombre d'électeurs peut atteindre 0 ou 10 et y rester, ou bien continuer à fluctuer entre ces deux valeurs.

2. (a) Si $i > j$, alors $\binom{j}{i} = 0$.

Si $i \leq j$, alors

$$\binom{j}{i} = \frac{j!}{i!(j-i)!} = \frac{j(j-1)\dots(j-i+1)}{i(i-1)\dots 1} = \frac{j}{i} \times \frac{j-1}{i-1} \times \dots \frac{j-(i-1)}{i-(i-1)} = \prod_{k=0}^{i-1} \frac{j-k}{i-k}.$$

(b) D'après ce qui précède, $\binom{j}{i}$ est le produit des $\frac{j-k}{i-k}$ pour $k$ allant de 0 à $i-1$. On complète donc la ligne 7 avec `aux = aux*(j-k)/(i-k)`.

(c) Puisque le terme $(i,j)$ de la matrice $M$ est $P_{(X_k=i)}(X_{k+1} = j) = \binom{N}{j} \left(\frac{i}{N}\right)^j \left(1 - \frac{i}{N}\right)^{N-j}$, on en déduit les instructions suivantes :

```
1 M = np.zeros((N+1,N+1))
2 for i in range(N+1):
3     for j in range(N+1):
4         M[i, j] = C(j,N)*(i/N)**j*(1-i/N)**(N-j)
```

(d) Un état stable de la chaîne est défini par un vecteur ligne probabiliste (ou stochastique) vérifiant $UM = U$. Or $M$ est de la forme :

$$M = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ * & * & * & * \\ * & * & * & * \\ 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Les vecteurs lignes $\begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}$ et $\begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$ conviennent. Donc la chaîne de Markov admet au moins deux états stables.

3. (a) La liste `S` est initialement vide (ligne 2). A chacun des 10000 passages dans la boucle `for`, la variable `X` contient des simulations des valeurs de $X_1, X_2, \dots, X_{100}$ (ligne 4) et la liste `S` est augmenté d'une composante égale à la valeur de la simulation de $X_{100}$ (ligne 5). Donc, à l'issue des 10000 passages dans la boucle `for`, la liste `L` contient 10000 simulations de la variable $X_{100}$.

(b) Puisque `S` contient 10000 simulations de la variable $X_{100}$, alors pour tout entier $i \in [\![0, N]\!]$, `S.count(i)` compte le nombre de fois où $X_{100}$ a pris la valeur $i$. En divisant par 10000, on obtient la fréquence $f_i$ associée. Puisque le nombre de simulations est suffisamment grand, $f_i$ est une bonne estimation de la probabilité $P(X_{100} = i)$ et on peut alors conclure : le vecteur `loiemp` contient des estimations des probabilités $P(X_{100} = 0), \dots, P(X_{100} = 10)$.

(c) Le vecteur `V100` contient le résultat du produit $V_0 \times M^{100} = V_{100}$ (d'après le cours), avec $N = 10$, c'est-à-dire le vecteur qui défini l'état de la chaîne à l'instant 100 :

$$V_{100} = \begin{pmatrix} P(X_{100} = 0) & \dots & P(X_{100} = 10) \end{pmatrix}.$$

(d) D'après les questions précédentes, les résultats affichés sont les vecteurs `loiemp` et `V100` (avec $n = 10000$ et $N = 10$) pour trois valeurs successives de $n_0$ : 3, 5 et 8. Dans chacun de ces trois cas, notons $f_i$ la fréquence d'apparition de $i$ au cours des 10000 réalisations successives de $X_{100}$. On a alors :

- Pour $n_0 = 3$ :

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $f_i$ | $0,7017$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,2983$ |
| $P(X_{100} = i)$ | $0,7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,3$ |

- Pour $n_0 = 5$ :

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $f_i$ | $0,5071$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,4929$ |
| $P(X_{100} = i)$ | $0,5$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,5$ |

- Pour $n_0 = 8$ :

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $f_i$ | $0,1971$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,8028$ |
| $P(X_{100} = i)$ | $0,2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $0,8$ |

On constate que pour tout $i \in [\![0, 10]\!]$, $f_i \simeq P(X_{100} = i)$ donc la simulation de $X_k$ semble juste. En outre, il semble que $P(X_{100} = 0) = 1 - \dfrac{n_0}{10}$ et $P(X_{100} = 10) = \dfrac{n_0}{10}$ donc la loi limite de la suite $(X_k)$ est certainement la loi d'une variable aléatoire $X$ définie par :

$$X(\Omega) = \{0, N\}, \quad P(X = 0) = 1 - \frac{n_0}{N} \quad \text{et} \quad P(X = N) = \frac{n_0}{N}.$$

4. (a) Il faut continuer tant que $X_k$ ne prend ni la valeur 0, ni la valeur $N = 3$. On complète donc la ligne 7 ainsi : `while X != 0 and X != N :`

   Pour la ligne 9, on reprend la réponse de la question 1.(b) : `X = rd.binomial(N, X/N)`

   (b) Le premier graphique permet de représenter l'histogramme associé à la série définie par la liste `T`, c'est-à-dire la liste de 10000 réalisations de la variable aléatoire $T$, pour les classes définie par le vecteur `c`, c'est-à-dire les 11 classes $[0.5, 1.5], [1.5, 2.5], \ldots, [10.5, 11.5]$, respectivement centrées en $1, 2, \ldots, 11$. Cet histogramme permet de visualiser la loi de $T$.

   Le second graphique permet, quant à lui, de visualiser la loi d'une variable aléatoire géométrique de paramètre $1/3$ (en faisant 10000 simulations de cette loi et en affichant l'histogramme des fréquences qui sont alors proches des probabilités théoriques avec la loi faible des grands nombres).

   On constate que ces deux graphiques sont quasi identiques, signe que les lois sont très proches, voire les mêmes. On peut donc penser que la loi de $T$ est la loi géométrique de paramètre $1/3$.

---

## Exercice 4

1. Les attributs `id.vehicule` et `id.annonce` sont des clefs primaires des tables `vehicule` et `annonce` respectivement, car d'après l'énoncé deux enregistrements différents de la table `vehicule` ne peuvent posséder le même identifiant `id.vehicule`, et de même deux enregistrements différents de la table `annonce` ne peuvent posséder le même `id.annonce`.

   L'attribut `id.vehicule` de la table `annonce` est une clef étrangère : il permet d'associer chaque enregistrement de la table `annonce` à un unique enregistrement de la table `vehicule`.

2. Voici le schéma relationnel associé aux tables `vehicule` et `annonce` :

   `vehicule = ((id.vehicule : INTEGER), (marque : TEXT), (modele : TEXT), (prix.neuf : INTEGER));`

   `annonce = ((id.annonce : INTEGER), (#id.vehicule : INTEGER), (annee : INTEGER), (km : INTEGER), (prix.occasion : INTEGER));`

3. Voici la requête SQL demandée :

```
SELECT modele FROM vehicule WHERE marque = 'Dubreuil Motors'
```

4. Cette requête met à jour les enregistrements de la table `annonce` de la façon suivante : elle identifie l'enregistrement de la table `vehicule` associé à chaque annonce grâce à l'identifiant `id.vehicule`, et lorsque l'attribut `prix.neuf` de cet enregistrement est strictement inférieur à l'attribut `prix.occasion` de l'annonce, la valeur de `prix.occasion` est mise à jour en prenant la valeur `prix.neuf` du véhicule.

   Autrement dit, cette requête permet de plafonner le prix de vente d'occasion en s'assurant qu'il ne dépasse pas le prix neuf. La table `vehicule` n'est pas modifiée.

5. Voici la requête SQL demandée :

```
SELECT id.annonce, km, prix.neuf, prix.occasion
FROM annonce INNER JOIN vehicule ON annonce.id.vehicule = vehicule.id.vehicule
```

---