Révisions 10

Correction du sujet d'Informatique

Exercice 1

1. On utilise `de = np.floor(rd.random()*6)+1` ou `de = rd.randint(1, 7)` .

2. Voici le programme complété :

```
nbr_six_succ = 0
    while nbr_six_succ <>4:
        de = rd.randint(1, 7)
        if de == 6:
            nbr_six_suc = nbr_six_suc+1
        else:
            nbr_six_succ=0
```

3. Voici la fonction demandée :

```
def nb_de_lancers():
    nbr_six_succ = 0
    X = 0
    while nbr_six_succ <>4:
        de = rd.randint(1, 7)
        X = X+1
        if de == 6:
            nbr_six_suc = nbr_six_suc+1
        else:
            nbr_six_succ = 0
    return(X)
```

4. Voici le programme demandé :

```
S = 0
for k in range(1000):
    S = S+nbr_de_lancers()
T = S/1000
print(T)
```

On remarque, en relançant plusieurs fois le programme que le nombre moyens de lancers nécessaires pour obtenir quatre "six" consécutif est toujours autour de 1560. L'estimateur est bien stable (ce qui illustre la loi faible des grands nombres : l'estimateur de Monte-Carlo converge toujours vers l'espérance) et on peut dire que l'espérance de X est proche de 1560.

Exercice 2

1. Voici la fonction `deeq()` :

```
def deeq():
    return(rd.randint(1,7))
```

2. Voici la fonction `depipe()` :

```
def depipe():
    r = rd.random()
    if r<1/2:
        return(6)
    elif r<5/6:
        return(1)
    else:
        return(2)
```

3. Voici la fonction `simulX()` :

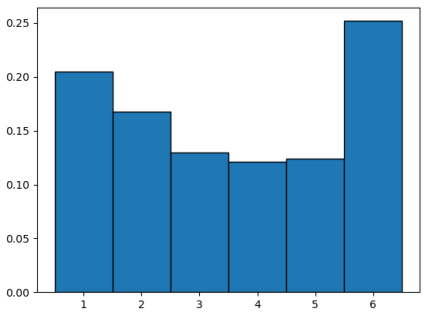
```
def simulX():
    r = rd.random()
    if r<3/4:
        X = deeq()
    else:
        X = depipe()
    return(X)
```

4. Voici les commandes demandées :

```
s = np.zeros(10000)
for k in range(10000):
    s[k] = simulX()

c = np.arange(0.5, 7.5, 1)
plt.hist(s, c, density='True', edgecolor='k')
plt.show()
```

On obtient le graphique suivant :



Exercice 3

1. La suite (X_n) est une chaîne de Markov car c'est une suite de variables aléatoires telle que la variable X_{n+1} dépend de la position de la variable X_n.

   Sachant $X_n = i$, X_{n+1} suit une loi uniforme sur $\{i-1; i+1\}$.

2. Voici la fonction demandée :

```
def deplacements(n):
      x = 0
      for k in range(n):
          if rd.random()<1/2:
```

```
            x = x-1
        else:
            x = x+1
    return(x)
```

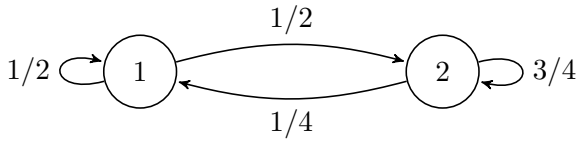
3. Voici le programme complété :

```
e = 0
for k in range(10000):
    x = deplacements(100)
    if np.abs(x)>10:
        e = e+1
t = e/10000
print(t)
```

La loi faible des grands nombres est bien illustrée : l'estimateur de la fréquence est stable autour d'une même valeur (la probabilité) lorsque le nombre de simulations n est proche de $+\infty$. On obtient que la probabilité que l'individu soit éloigné de son point de départ de plus de 10 unités au bout de 100 instants est d'environ $0,27$.

Exercice 4

1. Le graphe probabiliste associé est :



La matrice de transition associé est :

$$A = \begin{pmatrix} P_{(X_n=1)}(X_{n+1}=1) & P_{(X_n=1)}(X_{n+1}=2) \\ P_{(X_n=2)}(X_{n+1}=1) & P_{(X_n=2)}(X_{n+1}=2) \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{pmatrix}.$$

2. Voici le programme demandé :

```
def simulX(n):
    X = np.zeros(n)
    X[0] = 1
    for k in range(n-1):
        if X[k]==1 :
            if rd.random()<1/2 :
                X[k+1] = 1
            else:
                X[k+1] = 2
        else:
            if rd.random()<3/4 :
                X[k+1] = 2
            else:
                X[k+1] = 1
    return(X)
```

3. Rappelons que si $U_n = \begin{pmatrix} P(X_n = 1) & P(X_n = 2) \end{pmatrix}$ est le n-ième état probabiliste, alors :

$$U_n = U_0 \times A^n,$$

avec $U_0 = \begin{pmatrix} 1 & 0 \end{pmatrix}$. On cherche $P(X_7 = 0)$ et c'est donc le coefficient $(1, 1)$ ($(0, 0)$ sur Python) de la matrice A^7.

Voici les instructions demandées (dans la console) :

```
>>> A = np.array([[1/2,1/2],[1/4,3/4]])
>>> P = al.matrix_power(A,7)
>>> P[0,0]
0.3333740
```

Donc $P(X_7 = 1) \simeq 1/3$.

4. (a) Rappelons qu'un vecteur ligne U est un état stable si :
   - $U = UA \Leftrightarrow {}^tU = {}^tA\,{}^tU$ et donc ${}^tU$ est un vecteur propre de ${}^tA$;
   - U est un vecteur ligne stochastique (c'est-à-dire à coefficients positifs de somme égale à 1).

   On cherche donc le vecteur propre associé à la valeur propre 1 et on le "normalise" pour obtenir un vecteur stochastique.

   Voici les instructions demandées (dans la console) :

```
>>> Sp,VP = al.eig(np.transpose(A))
>>> Sp, VP
[0.25  1.]
[[-0.7071068  -0.4472136]
 [ -0.7071068  -0.8944272]]
>>> VP[:, 1]/np.sum(VP[:, 1])
[0.333333  0.666667]
```

   L'état stable serait donc $U = \begin{pmatrix} 1/3 & 2/3 \end{pmatrix}$.

   (b) On vérifie que pour n grand, $U_0 \times A^n \simeq U$ (où U est l'état stable obtenu précédemment). Voici les instructions demandées (toujours dans la console) :

```
>>> P = al.matrix_power(A, 100)
>>> P[0, :]
[0.333333  0.666667]
```

   Il semble qu'on ait bien convergence vers l'état stable U.

Exercice 5

1. Voici la fonction `bin(n, p)` :

```
1 def bin(n,p):
2     X = 0
3     for f in range(n):
4         if rd.random()<p:
5             X = X+1
6     return(X)
```

2. Voici le programme demandé :

```
n = int(input('Donner n : '))
p = float(input('Donner p : '))
X = np.zeros(10000)
for k in range(10000):
    X[k] = bin(n, p)
c = np.arange(-0.5, n+1.5)
plt.hist(X, c, density='True', edgecolor='k')
```

3. (a) La fonction f est la densité de la loi normale $\mathcal{N}(m, \sigma)$.

   (b) Ce graphique illustre le théorème limite centrale et l'approximation d'une binomiale $\mathcal{B}(n, p)$ par une normale $\mathcal{N}(np, np(1-p))$.

   On simule 10000 fois la variable aléatoire $S_n = \sum_{i=1}^{n} X_i$ où les X_i sont indépendantes et suivent toutes une loi binomiale de paramètre n et p. D'après le TLC,

$$S_n^* = \frac{S_n - np}{\sqrt{np(1-p)}} \simeq S \text{ avec } S \hookrightarrow \mathcal{N}(0,1).$$

   Avec les propriétés de la loi normale,

$$S_n = \sqrt{np(1-p)}S_n^* + np \simeq \sqrt{np(1-p)}S + np \hookrightarrow \mathcal{N}(np, np(1-p)).$$

   C'est ce que nous constatons graphiquement puisque la densité de la loi $\mathcal{N}(m, \sigma)$ (où m est la moyenne des simulations donc environ l'espérance np et σ l'écart type donc environ $np(1-p)$) se superpose aux fréquences empiriques de S_n.

Exercice 6

1. (a) Voici le programme demandé :

```
a = int(input('Donner a : '))
n = int(input('Donner n : '))

T = np.zeros(n)
Z = np.zeros(n)
for k in range(n):
    X = rd.uniform(0,a,100)
    T[k] = np.max(X)-np.min(X)
    Z[k] = 2*np.sum(X)/100
print(T)
print(Z)
```

   (b) Les réalisations de Z_{100} ont l'air en moyenne bien réparties autour de 10 contrairement à celles de T_{100}. Par contre, les réalisations de T_{100} ont l'air plus resserré autour de 10 que celles de Z_{100}. On en déduit que T_n semble être un meilleur estimateur de Z_n.

2. (a) On calcule ici les moyennes et les écarts types de T_{100} et Z_{100} pour 10000 réalisations.

   (b) Ces résultats confirment ceux de la question précédente :

   - Z_{100} est en moyenne plus proche que T_{100} de 10.
     On peut donc imaginer que Z_n est sans biais (c'est-à-dire que $E(Z_n) = a$) alors que T_n est biaisé (c'est-à-dire que $E(T_n) \neq a$).

- T_{100} prend des valeurs moins dispersées que Z_{100}.
  On peut également conjecturer que $V(T_n) \leq V(Z_n)$.

3. (a) Voici les commandes demandées :

```
1 c=np.linspace(0,2*a,50)
2 plt.subplot(1,2,1)
3 plt.hist(Z,c)
4 plt.title("Histogramme 1")
5 plt.subplot(1,2,2)
6 plt.hist(T,c)
7 plt.title("Histogramme 2")
8 plt.show()
```

   (b) L'histogramme 1 est centré sur 10 contrairement au deuxième. De plus, ses valeurs sont plus dispersées autoure de 10 par rapport au deuxième. Donc l'histogramme 1 correspond à celui de Z_n et l'histogramme 2 à celui de T_n.

Exercice 7

1. Voici la fonction `fibo` :

```
1 def fibo(k):
2     a = 0
3     b = 1
4     for i in range(1, k+1):
5         aux = b
6         b = a+b
7         a = aux
8     return(a)
```

2. Voici le script demandé :

```
1 L = [fibo(k) for k in range(20)]
```

3. Voici la fonction `recherche` demandé :

```
1 def recherche(x, L):
2     k = 0
3     while L[k]<=x:
4         k = k+1
5     return(L[k-1])
```

4. L'algorithme `Zeckendorf(n)` renvoie la liste contenant la décomposition de Zeckendorf de n. Plus précisément :

   - De la ligne 2 à 6 : On construit une liste `L` contenant les termes de la suite de Fibonnacci $F_0, F_1, \ldots$ jusqu'au premier terme F_i tel que $F_i > n$ inclus.
   - De la ligne 7 à 8 : On définit `k` une variable égale à `n` et `T` la liste vide.
   - Ligne 9 : Tant que `k` n'est pas nul.
   - Ligne 10 : On recherche dans `L` le plus grand élément de la liste `L` qui soit inférieur ou égale à `k`. C'est possible car `L` est bien triée dans l'ordre croissant (la suite de Fibonacci est croissante), de premier terme inférieur ou égal à `n` (c'est $F_0 = 0$) et de dernier terme F_i strictement plus grand que `k`.

- Ligne 11 : On ajoute le terme de la suite de Fibonacci obtenu dans `T`. Il correspond au plus grand terme de la suite plus petit que `k`.
- Ligne 12 : On retranche à `k` le terme de la suite obtenu ligne 10.
- Ligne 13 : On retourne la décomposition de Zeckendorf de `k`.

5. L'algorithme `Zeckendorf` est un algorithme glouton car, à chaque passage dans la deuxième boucle `while`, on choisit le plus grand terme de la suite de Fibonacci plus petit que `k`. Cela correspond bien au principe des algorithmes gloutons : choisir à chaque étape la meilleur (ou la plus grande) solution.

Exercice 8

1. Voici la fonction `indicemin` demandée :

```
1 def indicemin(L):
2     m = 0
3     for k in range(len(L)):
4         if L[k]<L[m]:
5             m = k
6     return(m)
```

2. La fonction `mystere` renvoie la liste `T` correspondant à la liste `L` triée. Plus précisément :

- Ligne 2 et 3 : On définit `T` la liste vide et `n` la longueur de `L`.
- Ligne 4 : On passe dans la boucle `for` autant de fois qu'il y a d'éléments dans `L`.
- Ligne 5 à 7 : A chaque passage dans la boucle, on cherche l'indice `i` du minimum de `L`, on ajoute le minimum `L[i]` dans `T` et on supprime ce minimum de `L`.
- Ligne 8 : On retourne la liste `T` qui contient donc tous les éléments de la liste `L` qui ont été triés.

Exercice 9

1. Ce script permet d'obtenir les valeurs propres de la matrice `A` et des vecteurs propres associées. La commande `L = al.eig(A)` affecte à `L` un couple dont le premier élément `L[0]` contient les valeurs propres et le deuxième `L[1]` contient les vecteurs propres.

2. On a donc :

- $Sp(A) = \{3, 1, -1\}$,
- $\begin{pmatrix} 0.5773503 \\ 0.5773503 \\ -0.5773503 \end{pmatrix}$ et donc $\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$ est un vecteur propre de A associé à la valeur propre 3,
- $\begin{pmatrix} 0 \\ -0.7071068 \\ 0.7071068 \end{pmatrix}$ et donc $\begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$ est un vecteur propre de A associé à la valeur propre 1,
- $\begin{pmatrix} -0.4082483 \\ 0.8164966 \\ -0.4082483 \end{pmatrix}$ et donc $\begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix}$ est un vecteur propre de A associé à la valeur propre -1.

Par concaténation de familles libres (à chaque fois un seul vecteur non nul) des trois sous-espaces propres $E_3(A)$, $E_1(A)$ et $E_{-1}(A)$ (valeurs propres distinctes), la famille $\left(\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix} \right)$

est libre. Comme elle est de cardinale 3 et que $\dim(\mathscr{M}_{3,1}(\mathbb{R})) = 3$, c'est une base de $\mathscr{M}_{3,1}(\mathbb{R})$. Donc A est diagonalisable.

Exercice 10

1. Voici le programme demandé :

```
def dicho(eps):
    a = 1/3
    b = 1
    while np.abs(b-a)>eps :
        m = (a+b)/2
        if m**3+m**2+m-1<0:
            a = m
        else:
            b = m
    return(m)
```

2. Voici le programme demandé :

```
def suiteu(n):
    u = 1
    for k in range(1, n+1):
        u = 1/(u**2+u+1)
    return(u)
```

3. Voici le programme demandé :

```
def approx(eps):
    n = 0
    while (135/169)**n>eps :
        n = n+1
    return(suiteu(n))
```

Exercice 11

1. Voici le programme complété :

```
def alpha(n):
    x=np.arange(0,2,0.0001)
    k=0
    while x[k]*np.log(1+x[k])<1/n**2:
        k=k+1
    return(x[k])
```

2. On trace ici les 50 premiers termes de la suite de terme général $k \times \alpha_k$. On remarque que cette suite semble converger vers 1. On peut donc conjecturer que :

$$\lim_{n\to+\infty} n\alpha_n = 1 \Leftrightarrow \lim_{n\to+\infty} \frac{\alpha_n}{1/n} = 1 \Leftrightarrow \alpha_n \underset{+\infty}{\sim} \frac{1}{n}.$$

On avait effectivement démontré ce résultat dans l'exercice 18 du TD1.

Exercice 12

1. On utilise par exemple la commande :

```
>>> u = (-1)**np.arange(0, 50)/np.arange(1, 51)
```

2. On utilise par exemple la commande :

```
>>> v = np.cumsum(u)
```

3. On utilise les commandes :

```
>>> n = np.arange(1,51)
>>> plt.plot(n, v, 'x')
>>> plt.show()
```

4. D'après le graphique, les suites extraites (S_{2n}) et (S_{2n+1}) semblent être adjacentes : (S_{2n}) est croissante, (S_{2n+1}) est décroissante et $\lim\limits_{n\to+\infty}(S_{2n+1}-S_{2n})=0$. Elles convergent donc vers une même limite $S\simeq 0.7$. On en déduit que $\lim\limits_{n\to+\infty} S_n = S \simeq 0.7$ donc la série harmonique alternée est convergente et sa somme vaut environ 0.7 (voir l'exercice 2 de l'AP4 pour une preuve de ces résultats).

5. Voici le programme demandé :

```
def vitesse(eps):
    S = 1 #valeur de S1
    n = 1
    while np.abs(S-np.log(2))>eps :
        n = n+1
        S = S+(-1)**(n-1)/n
    return(n)
```

Exercice 13

1. On peut prendre `Id` comme clef primaire pour la table `Adhérent` et `Sport` comme clef primaire pour la table `Cotisation`.

   `Sport` est une clef étrangère pour la table `Adhérent` qui fait référence à la clef primaire de la table `Cotisation`.

2. (a) `SELECT Nom, Prénom FROM Adhérent WHERE Sport="Judo"`
   (b) `SELECT Téléphone FROM Adhérent WHERE Nom="Brun"`
   (c) `SELECT Nom FROM Adhérent WHERE Age<30`
   (d) `SELECT Téléphone FROM Adhérent WHERE Sport="Judo" AND Ville="Agen"`
   (e) `SELECT Age FROM Adhérent WHERE NOT Sport="Judo"`

3. (a) `UPDATE Adhérent SET Sport="Karaté" WHERE Nom="Leroy"`
   (b) `UPDATE Ahérent SET Age=Age+1`

4. (a)
```
SELECT Nom,Prénom FROM Adhérent INNER JOIN Cotisation
ON Adhérent.sport=Cotisation.sport WHERE Cotis>=250
```
   (b)
```
SELECT Responsable FROM Adhérent INNER JOIN Cotisation
ON Adhérent.sport=Cotisation.sport WHERE Nom="Leroy"
```