Correction - DS 10

Devoir surveillé du Mardi 12 Mars

Exercice 1

1. On peut utiliser le code suivant :

```
def exponentielle(lambda):
      u = rd.random()
      return(-(1/lambda)*np.log(1-u))
```

2. (a) On résout : $p = 1 - e^{-\lambda} \Leftrightarrow \lambda = -\ln(1-p)$. Avec le résultat admis, et en se souvenant que `np.floor` est la fonction partie entière sur `Python`, on propose la fonction suivante :

```
def geom(p):
      lambda = - np.log(1-p)
      x = exponentielle(lambda)
      return(np.floor(x)+1)
```

(b) On procède ainsi :

```
x = np.zeros(10000) #vecteur de taille 10000 avec que des 0
for k in range(10000):
     x[k] = geom(0.2) #remplace la k-ème composante par une
simulation
```

(c) La commande `np.mean(x)` calcule la moyenne des composantes du vecteur `x`. Par la loi faible des grands nombres, elle devrait être proche de l'espérance $\frac{1}{p} = \frac{1}{0.2} = 5$. C'est bien ce qu'on obtient ici.

De même, `np.mean((x-m)**2)` calcule la variance des composantes du vecteur `x`. Par (un corollaire de) la loi faible des grands nombres, elle devrait être proche de la variance théorique $\frac{q}{p^2} = \frac{0.8}{0.04} = 20$. C'est bien là aussi ce qu'on observe.

(d) Le premier graphique permet de réprésenter l'histogramme associé à la série définie par le vecteur `x` (créé à la question 2.(b)), c'est-à-dire le vecteur de 10000 réalisation de la variable aléatoire Y, 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 Y.

Le deuxième graphique permet de visualiser la loi d'une variable aléatoire géométrique de paramètre 0.2 (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. Ceci confirme bien que la variable Y ainsi définie dans l'énoncé, suit bien une loi géométrique.

Exercice 2

1. (a) Sur $X(\Omega) =]\lambda; +\infty[$, $F(x) = 1 - \dfrac{\lambda^k}{x^k}$ donc F est dérivable et $F'(x) = \dfrac{k\lambda^k}{x^{k+1}} > 0$.
F est ainsi continue et strictement croissante sur $]\lambda; +\infty[$. Elle réalise donc une bijection de $]\lambda; +\infty[$ dans $]0; 1[$.

Soit $y \in]0; 1[$ fixé. Résolvons l'équation $F(x) = y$ d'inconnue $x \in]\lambda; +\infty[$:

$$F(x) = y \Leftrightarrow 1 - \frac{\lambda^k}{x^k} = y \Leftrightarrow \frac{\lambda^k}{x^k} = 1 - y \Leftrightarrow x^k = \frac{\lambda^k}{1-y} \Leftrightarrow x = \frac{\lambda}{\sqrt[k]{1-y}}.$$

Ainsi, $\forall y \in]0; 1[$, $F^{-1}(y) = \dfrac{\lambda}{\sqrt[k]{1-y}}$.

(b) On en déduit alors la fonction suivante pour simuler une loi de Pareto de paramètres λ et k avec la méthode d'inversion :

```
def Pareto1(lbd, k):
    U = rd.random()
    X = lbd/(1-U)**(1/k)
    return(X)
```

2. (a) Pour tout i, $X_i(\Omega) =]0, 1]$ donc $(\max(X_1, \ldots, X_k))(\Omega) =]0, 1]$ et donc $Y(\Omega) = [\lambda, +\infty[$.
Donc $F_Y(x) = 0$ si $x < \lambda$.
Si $x \geq \lambda$, on a :

$$\begin{aligned}
F_Y(x) &= P(Y \leq x) = P\left(\frac{\lambda}{x} \leq \max(X_1, \ldots, X_k)\right) \\
&= 1 - P\left(\max(X_1, \ldots, X_k) < \frac{\lambda}{x}\right) = 1 - P\left(\bigcap_{i=1}^{k} (X_i < \frac{\lambda}{x})\right) \\
&= 1 - \prod_{i=1}^{k} P(X_i < \frac{\lambda}{x}) \quad \text{(par indépendance des } X_i\text{)} \\
&= 1 - \prod_{i=1}^{k} P(X_i \leq \frac{\lambda}{x}) \quad \text{(car les } X_i \text{ sont à densité)} \\
&= 1 - \prod_{i=1}^{k} \frac{\lambda}{x} \quad \text{(car } X_i \hookrightarrow \mathcal{U}(]0, 1]) \text{ et } \frac{\lambda}{x} \in]0, 1]) \\
&= 1 - \frac{\lambda^k}{x^k}.
\end{aligned}$$

Donc Y suit une loi de Pareto de paramètres λ et k.

(b) On en déduit le programme suivant :

```
def Pareto2(lbd, k):
    U = rd.random(k)
    Y = lbd/np.max(U)
    return(Y)
```

Exercice 3

1. (a) Les voitures de la table `Voiture` sont distinguées par leurs plaques minéralogiques, donc la colonne `Plaque` est la clé primaire de cette table.

   Pour la table `Client`, nous avons le choix entre deux colonnes : `Id_Client` et `Téléphone`. Nous choisirons la plus simple, `Id_Client`.

   Pour la table `Tarif`, la clé primaire est `Atelier`.

   (b) La colonne `Propriétaires` de la table `Voiture` fait référence à la clé primaire `Id_client` de la table `Client`. C'est donc une clé étrangère.

   La colonne `Atelier` de la table `Voiture` fait référence à la clé primaire `Atelier` de la table `Tarif`. C'est donc également une clé étrangère.

2. (a) Cette requête permet de remplacer le nom du chef de l'atelier `Pneu` : `Yves` remplace `Paul`.

   (b) Cette requête permet de sélectionner les marques et les modèles des voitures envoyées à l'atelier `Pneu`, c'est-à-dire :

Renault	Clio
Fiat	Uno
Renault	Twingo

   (c) Cette requête permet de sélectionner les propriétaires des voitures de marque Renault qui sont envoyées à l'atelier `Pneu`, c'est-à-dire :

1
6

   (d) Cette requête permet de supprimer de la table `Voiture` la voiture dont la plaque minéralogique est `DG 103 HY`.

3. (a) La requête est : `SELECT Plaque,Atelier FROM Voiture`

   Le résultat est :

Voiture

DG 103 HY	Carrosserie
EF 334 GA	Pneu
EI 189 KA	Pneu
FA 934 TH	Mécanique
BB 512 IJ	Pneu
FD 246 MT	Mécanique

   (b) La requête est :

```
SELECT Nom FROM Client
INNER JOIN Voiture
ON Voiture.Propriétaire=Client.Id_client
WHERE Marque="Peugeot"
```

   Le résultat est :

Leclerc

(c) La requête est : `SELECT Atelier FROM Tarif WHERE Prix<100`
Le résultat est :

Pneu

(d) La requête est :

```
SELECT Téléphone FROM Client
INNER JOIN Voiture
ON Client.Id_client=Voiture.Propriétaire
WHERE (Marque="Renault") AND (Atelier="Pneu")
```

Le résultat est :

668543252
638899821

(e) La requête est :

```
SELECT Marque,Modèle FROM Voiture
WHERE (Atelier="Carrosserie") OR (Atelier="Mécanique")
```

On aurait également pu faire :

```
SELECT Marque,Modèle FROM Voiture WHERE NOT (Atelier="Pneu")
```

Le résultat est ;

Peugeot	807
Lancia	Delta
Fiat	500

(f) La requête est :

```
UPDATE Client SET Téléphone=0651096754 WHERE Nom="Ridy"
```

(g) Il faut dans un premier temps récupérer l'`Id_client` de Thomas McGregor dans la table `Client`, ce que l'on fait à l'aide de la requête :

```
SELECT Id_client FROM Client WHERE Nom="McGregor"
```

Le résultat est :

4

Il faut maintenant supprimer la ligne de la table `Voiture` correspondant au propriétaire dont le numéro est 4. La requête est :

```
DELETE FROM Voiture WHERE Propriétaire=4
```

On aurait pu rassembler les deux requêtes en une seule :

```
DELETE FROM Voiture
WHERE Propriétaire=(SELECT Id_client FROM Client WHERE Nom="McGregor")
```

Exercice 4

1. L'ensemble des états est $E = [\![0, a+b+1]\!]$. La matrice de transition est la matrice carrée d'ordre $a+b+1$ définie par (avec $q = 1-p$) :

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ q & 0 & p & 0 & \dots & 0 & 0 & 0 \\ 0 & q & 0 & p & \dots & 0 & 0 & 0 \\ 0 & 0 & q & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & 0 & p & 0 \\ 0 & 0 & 0 & 0 & \dots & q & 0 & p \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

   On remarque que les vecteurs-lignes stochastiques :

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$$

   sont des états stables de la chaîne de Markov (ils vérifient la relation $UA = U$). Ils correspondent aux états 0 et $a+b$ qu'on ne peut plus quitter une fois qu'on les a atteint (ces états sont appelés états absorbants).

2. Voici la fonction demandée :

```
def ruinejoueur(p,a,b):
    L = [a]
    while L[-1] > 0 and L[-1] < a+b :
        if rd.random() < p :
            L.append(L[-1]+1)
        else:
            L.append(L[-1]-1)
    return(L)
```

3. Grâce à ce programme, on trace le graphique de la suite (X_n) (en ordonnée) en fonction de l'instant n (en abscisse).

   On remarque sur le graphique que le joueur est ruiné au bout d'environ 90 instants et le jeu s'arrête.

4. Voici la fonction demandée :

```
def ruinejoueurfrequence(p,a,b)
      N = 0
      for k in range(1000):
          L = ruinejoueur(p,a,b)
          if L[-1] == 0 :
              N = N+1
      f = N/1000
      return(f)
```

   On utilise ici la loi faible des grands nombres : une probabilité d'un événement A est la limite de la suite des fréquences de A lorsque l'on fait tendre le nombre de simulations de l'expérience vers $+\infty$.

Exercice 5

1. Sachant $(X = k)$, Y compte le nombre de succès au cours de k épreuves de Bernoulli indépendantes de même paramètre $p = 0,05$. Donc Y suit une loi binomiale de paramètres k et $0,05$.

2. On utilise la question précédente pour compléter la fonction :

```
def simulY():
    x = rd.poisson(20)
    y = rd.binomial(x, 0.05)
    return(y)
```

3. Avec une boucle `for`.

```
def SimulY(N):
    M = np.zeros(N)
    for k in range(N):
        M[k] = simulY()
    return(M)
```

4. Voici la fonction `loipoisson` :

```
def loipoisson(lb)
    V = np.zeros(11)
    V[0] = np.exp(-lb)
    for k in range(1,11):
        V[k] = V[k-1]*lb/k
    return(V)
```

5. La variable `U` est un vecteur ligne contenant un échantillon de 100000 simulations de la variable aléatoire Y.

   La variable `c` est un vecteur ligne contenant les classes pour le tracé de l'histogramme (de -0.5 à 0.5 pour la modalité 0, de 0.5 à 1.5 pour la modalité 1, ..., de 8.5 à 9.5 pour la modalité 9).

   La variable `n` est un vecteur ligne contenant les entiers de 0 à 9.

   Enfin, la variable `V` est un vecteur ligne contenant les 10 premières probabilités théoriques d'une loi de Poisson de paramètre 1.

   Ce programme permet de tracer deux diagrammes en bâtons :

   - Le graphe de gauche représente le diagramme en bâtons des fréquences de l'échantillon contenant les 100000 simulations de Y. Le bâton d'abscisse i indique en ordonnée la fréquence d'apparition de i dans l'échantillon généré.
   - Le graphe de droite représente le diagramme en bâtons des 10 premières probabilités théoriques d'une loi de Poisson de paramètre 1. Le bâton d'abscisse $i \in [\![0, 9]\!]$ indique en ordonnée la probabilité $\dfrac{e^{-1}}{i!}$.

6. Par comparaison des deux diagrammes obtenus, on constate que les fréquences empiriques de notre échantillon correspondant approximativement aux probabilités théoriques. D'après la loi forte des grands nombres, on peut donc supposer que Y suit une loi de Poisson de paramètre 1.

   C'est effectivement le cas : nous avons démontré dans l'exercice 15 du TD 9 que Y suit une loi de Poisson de paramètre $\lambda p = 20 \times 0.05 = 1$.

Exercice 6

1. Pour la fonction `indicemin` :

```
def indicemin(L):
    imin = 0
    for k in range(len(L)):
        if L[imin] > L[k] :
            imin = k
    return imin
```

2. `del L[i]` permet de supprimer de la liste `L` l'élément d'indice `i`.

3. Avec les deux questions précédentes, on propose une fonction qui :

   - cherche l'indice du minimum de la liste ;
   - supprime l'élément correspondant de la liste ;
   - cherche l'indice du minimum dans la liste des éléments restants ;
   - retourne la valeur associée.

   On obtient ainsi la valeur du deuxième minimum de la liste de départ.

```
def min2(L):
    imin = indicemin(L)
    del L[imin]
    imin2 = indicemin(L)
    return L[imin2]
```

4. Voici la fonction complétée :

```
def tri(L):
    M = []
    while len(L) > 0 :
        i = indicemin(L)
        M.append(L[i])
        del L[i]
    return M
```

   On crée la liste vide `M` qui contiendra à la fin du programme la liste triée dans l'ordre croissant des éléments de `L`. Puis, tant que la liste `L` n'est pas vide, c'est-à-dire tant que sa longueur `len(L)` est strictement positive,

   - on cherche l'indice `i` du minimum de `L` avec la commande `i = indicemin(L)`,
   - on ajoute le minimum à la fin de `M` avec la commande `M.append(L[i])`,
   - on le supprime de `L` avec la commande `del L[i]`.

   On obtient ainsi le résultat demandé.
