TP

POT-POURRI DE PROGRAMMATION EN PYTHON.

Table des matières

1. Compléments sur les simulations de variables aléatoires

On commence, pour toute la suite de ce TP à importer les bibliothèques et libraires nécessaires à chaque situation :

```
import numpy as np
import numpy.random as rd
```

1.1. **Un exemple de n-lancer d'une pièce de monnaie.** On désigne par n un entier naturel supérieur ou égal à 2. On lance n fois une pièce équilibrée (c'est-à-dire qui donne "pile" avec probabilité $\frac{1}{2}$ et "face" avec probabilité $\frac{1}{2}$), les lancers étant supposés indépendants. On note Z la variable aléatoire égale à 0 si on obtient aucun "pile" pendant ces n lancers et qui, dans le cas contraire, prend la valeur du premier "pile".

1. Rappeler comment simuler l'événement "la pièce tombe sur pile" à l'aide de la fonction rd.rand().

2. Dans la fonction Python suivante, on simule l'expérience en procédant à une suite constituée *a priori* de ces n lancers. À chaque expérience, si on tombe sur "pile", alors on arrête l'expérience et on renvoie le numéro du lancer, sinon on continue les lancers. Si à la fin des n lancers, on n'est jamais tombé sur "pile", alors on renvoie 0. Compléter cette fonction

```
def sumul_Z(n) :
  for i in range(1,n+1) :
    if ........... :
      return
  return
```

3. Dans la fonction Python suivante, on simule l'expérience en procédant à une suite constituée *a priori* d'une infinité de lancers. On effectue cette suite infinie tant qu'on obtient "face" et on compte alors le nombre de lancers effectuées pour obtenir le premier "pile". Si ce nombre est inférieur ou égal à n, Z prend cette valeur, sinon elle prend la valeur 0. Compléter cette fonction.

```
def sumul2_Z(n) :
  Z = 1
  while .............. :
    Z = Z + 1
  if .................. :
    return ..........
  else :
    return
```

4. Calculer, pour tout $k \in [\![0, n]\!], \mathbb{P}([Z = k])$. On traitera à part le cas $k = 0$.

1.2. **Un exemple de duite de tirages dans une urne.** Une urne contient initialement deux boules rouges et une boule bleue indiscernables au touché. On appelle pioche au hasard une boule dans l'urne puis

- Si la boule piochée est bleue, on la remet dans l'urne.
- Si la boule piochée est rouge, on ne la remet pas dans l'urne et on ajoute une boule bleue.

Ainsi le nombre de boules ne change pas à l'issue d'un tirage.

Pour tout entier naturel n, on note Y_n la variable aléatoire égale au nombre de boules rouges présentes dans l'urne à l'issue du n-ième tirage.

1. Soit $n \in \mathbb{N}^*$. Donner $Y_n(\Omega)$.

2. Compléter la fonction suivante, pour que, prenant en argument un entier $n \geqslant 1$, elle renvoie une simulation de la variable Y_n.

```
def sumul_Y(n) :
  r = 2
  for k in range(n) :
    if .............. :
      if ............... :
        r = r-1
  return ...........
```

3. On introduit pour tout $n \geqslant 1$, la variable aléatoire indicatrice de l'événement $[Y_n = 0]$, que l'on note T_n. Autrement dit,

$$T_n = \begin{cases} 1 & \text{si l'événement } [Y_n = 0] \text{ est réalisé} \\ 0 & \text{sinon} \end{cases}.$$

Quelle est la loi de la variable T_n ?

4. Écrire une fonction simul_T qui

- prend en argument un entier $n \geqslant 1$.
- renvoie une simulation de la variable T_n.

1.3. **Une loi usuelle ?** Soit $N \geqslant 3$ un entier. On considère une urne qui contient $N-1$ boules blanches et une seule boule noire. On effectue des tirages sans remise jusqu'à l'obtention de la boule noire et on note X la variable aléatoire égale au rang du tirage où on obtient la boule noire.

1. Donner $X(\Omega)$. La variable aléatoire est-elle finie ou infinie ?
2. Écrire une fonction simul_X qui
   - prend en argument un entier $N \geqslant 3$.
   - renvoie une simulation de X.
3. Recopier et exécuter les instructions ci-contre. Expliquer ce que fait ce script.

```
N = 5  # on fera varier la valeur de N
comptage_val_X = np.zeros(N)
for k in range(10000) :
  i = simul_X(N)
  comptage_val_X[i-1] += 1
freq_val_X = comptage_val_X / 10000
plt.bar(range(1,N+1), freq_val_X)
plt.show()
```

4. En interprétant le tracé obtenu à la question précédente, faire une conjecture sur la loi de X.
5. Démontrer la conjecture précédente.

1.4. **Simulation de variable aléatoire à l'aide de lois usuelles.** À la fête foraine, un jeu est proposé, dont la partie joué coûte 10 euros. Le jeu consiste à lancer une pièce de monnaie jusqu'à tomber sur "pile". On note alors n le rang d'apparition de ce premier "pile". Si $n \geqslant 2$, le joueur ne gagne rien, tandis que si $n \geqslant 3$, le joueur gagne n^2 euros. On note X la variable aléatoire égale au gain algébrique du joueur.

1. Donner $X(\Omega)$. La variable est-elle finie ou infinie ?
2. Écrire une fonction simul_X qui
   - prend en argument un réel $p \in]0,1[$ tel que la pièce tombe sur "pile" avec probabilité p.
   - renvoie une simulation de X.
3. Sans calculer la loi de la variable X, proposer une stratégie informatique qui permette de décider si le jeu est favorable au joueur (en fonction de p).

2. Simulation de couples de variables discrètes.

2.1. **ECRICOME 2022.** On dispose de trois urnes U_1, U_2 et U_3 et d'une infinité de jetons numérotés 1,2,3,4,...

On répartit un par un les jetons dans les urnes : pour chaque jeton, on choisit au hasard et avec équiprobabilité une des trois urnes dans laquelle on place le jeton. Le placement de chaque jeton est indépendant de tous les autres jetons et la capacité des urnes en nombre de jetons n'est pas limité.

Pour tout entier naturel n non nul, on note $X_n^{(1)}$ (resp. $X_n^{(2)}$, $X_n^{(3)}$) le nombre de jetons présent dans l'urne 1 (resp. l'urne 2, l'urne 3) après avoir réparti les n premiers jetons.

Soit Y le nombre de jetons placés lorsque, pour la première fois, deux urnes sont occupées par au moins un jeton.

Soit Z le nombre de jetons placés lorsque, pour la première fois, les trois urnes contiennent chacune au moins un jeton.

1. Compléter la fonction Python suivante pour qu'elle simule les n premiers placements des jetons et qu'elle renvoie un tableau numpy contenant le résultat des simulations de $X_n^{(1)}$, $X_n^{(2)}$ et $X_n^{(3)}$.

```
def remplissage_urne(n) :
  R = ..............
  for k in ......... :
    indice = ...........
    R[indice] += 1
  return R
```

2. On note a (resp. b et c) le nombre de jetons présents dans l'urne 1 (resp. 2 et 3) à un moment donné de l'expérience. Montrer que
   - Au moins une des urnes est vide si et seulement si $abc = 0$.
   - Au moins deux urnes sont vides si et seulement si $(a+b)(b+c)(c+a) = 0$.
3. Compléter la fonction Python suivante pour qu'elle simule l'expérience jusqu'à ce que les trois urnes contiennent au moins un jeton et qu'elle renvoie une liste $[Y, Z]$ contenant le résultat des simulations de Y et Z.

```
def simul_YZ() :
  R = ..........
  rang = 0
  liste = []
  while ..................... :
    rang = rang + 1
    indice = .............
    R[indice] += 1
  ...............
  while ................ :
    rang = rang + 1
    indice = ..................
    R[indice] += 1
  ......................
  return liste
```

4. Écrire un programme qui simule un grand nombre de fois le couple (X, Y), puis donne une estimation de l'espérance de X, de celle de Y et du coefficient de corrélation de (X, Y).

2.2. **Une expérience en deux étapes.** On lance une pièce de monnaie équilibrée une infinité de fois. On note N le rang d'apparition du premier "pile" obtenu. Si le premier "pile" a été obtenu au rang n, on lance ensuite n fois un dé équilibré à 6 faces. On note alors X le nombre de 6 obtenus et S la somme des résultats obtenus.

1. Écrire une fonction Python, simulNX() qui renvoie une liste $[N, X]$ contenant le résultat de la simulation de N et de X.
2. Écrire une fonction Python, simulNS() qui renvoie une liste $[N, S]$ contenant le résultat de la simulation de N et de S. On calculera S à l'aide d'une boucle for.

2.3. **L'embarras du choix parmi les urnes.** Soit $n \geqslant 2$ un entier fixé. On dispose de n urnes numérotées de 1 à n. Pour chaque $k \in [\![1, n]\!]$, l'urne k est composée de k boules numérotées de 1 à k. On choisit une urne au hasard (uniformément) puis au tire une boule uniformément au hasard dans cette urne. On note

- X_n la variable aléatoire égale au numéro de l'urne choisie.
- Y_n la variable aléatoire égale au numéro de la boule tirée

1. Écrire une fonction simulXY qui prend en argument un entier $n \geqslant 2$ et qui renvoie une liste [X,Y] contenant le résultat de la simulation de X_n et Y_n.
2. Quelle est la loi de Y_n ?

2.4. **Le savant fou.** Un savant fou s'ennuie dans sa tour isolée et dispose d'une pièce usée dont la probabilité de tomber sur "pile" est $p \in]0, 1[$. Il décide de se lancer dans une expérience aléatoire dont le protocole est décrit ci-dessous :

- Il lance la pièce jusqu'à ce qu'elle tombe sur "pile". On note N la variable aléatoire égale au rang du premier "pile".
- Si la pièce est tombée sur "pile" pour la première fois au n-ième lancer, alors il remplit une urne de la manière suivante. Pour tout $k \in [\![1, n]\!]$,

- Si k est pair, alors il lance un dé à 6 faces. Il place alors ensuite autant de jetons numérotés k que de le résultat indiqué sur le dé.
- Si k est impair, alors il lance un dé à 8 faces. Il place ensuite dans l'urne autant de jetons numérotés k que le résultat indiqué sur le dé.

- Une fois l'urne remplie, il procède à n tirages successifs et sans remise dans cette urne, note les résultats obtenus et fait leur somme. On note S la variable aléatoire égale au résultat obtenu.

On souhaite simuler le couple (N, S). Pour cela, on rappelle que

- Si k est un entier, la commande `k % 2 == 0` renvoie `True` si k est pair et `False` si k est impair.
- Si `L` est une liste possédant plus de n éléments, alors la commande `rd.choice(L,n, replace = False)` simule une succession de n tirages sans remise dans cette liste et renvoie une liste contenant les résultats successifs.

Écrire une fonction Python qui renvoie une liste `[N,S]` contenant le résultat de la simulation de N et S.

2.5. **Un test de dépistage.** Chaque jour, le nombre de personnes subissant un dépistage dans une certaine pharmacie est modélisé par une variable aléatoire X suivant une loi de Poisson de paramètre $m = 5$. Chaque test a une certaine probabilité $p = \frac{1}{10}$ d'être positif et les tests sont indépendants les uns des autres. On note N le nombre de tests positifs un jour donné.

1. Écrire une fonction Python nommée `simulN()` qui simule la variable aléatoire N.
2. Créer un échantillon de taille 1000 de cette variable et le comparer avec un histogramme, à un échantillon de même taille d'une loi de Poisson de paramètre $mp = \frac{1}{2}$. Que peut-on conjecturer ?
3. Démontrer cette conjecture (ou relire le calcul du cours où une autre version de ce calcul a été fait).

3. Sélection d'activités

3.1. **Le paradoxe des anniversaires.** L'objectif de cette activité est de visualiser l'évolution de la probabilité qu'au moins deux personnes d'un groupe de n individus fêtent leur anniversaire le même jour, en fonction du nombre d'individus dans le groupe.

On suppose, pour simplifier, qu'une année n'est constituée que de 365 jours et que les naissances sont répartis uniformément tout au long de l'année. On s'intéresse à la date mais pas à l'année (exemple : 10 janvier !)

Déterminer graphiquement avec Python à partir de quelle valeur n, la probabilité qu'au moins deux personnes soient nées le même jour dépasse 1/2. Même question avec 99 %.

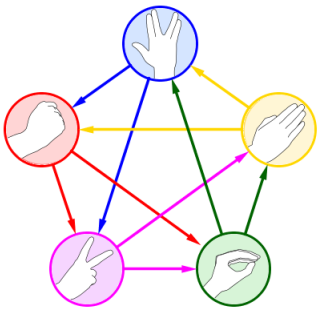
3.2. **Le code de César.** Jules César, général et stratège romain, a été (à ce qu'il parait) le premier militaire officiel à chiffrer ses messages. Sa méthode est assez simple : il décalait les lettres de 3 rangs dans l'alphabet.

On propose alors d'écrire une fonction de codage. Pour raffiner un peu le procédé élémentaire de César, la fonction devra prendre en argument le décalage (vers la droite ou vers la gauche), mais qui ne décale qu'une lettre sur deux (en commençant toujours par la deuxième lettre du mot). On aura en tête qu'en décalant `'z'` de 1 vers la droite, on retombe sur `'a'`.

Par exemple la commande `codage(2,'amicalement')` devra renvoyer `aoieaneoept`.

3.3. **Graphes *Shifumi*.** Inutile de rappeler les règles du *Shifumi* (ou Pierre-Feuille-Ciseaux). Ce jeu connaît aussi des variantes, comme celle (présentée plus bas) de la célèbre série télévisée *the Big Bang Theory*. Un graphe de *Shifumi* est un graphe censé représenter les règles d'une confrontation. Les sommets sont les *figures* possibles. Il y a une arrête d'un sommet vers un autre s'il y a un vainqueur parmi les deux figures, et l'arrête est orientée pour pointer vers le perdant de la confrontation.

1. Représenter graphiquement, puis sous Python le graphe du *shifumi* classique.
2. Représenter le graphe d'un *shifumi* équilibré à 5 sommets (Pierre, Feuille Ciseaux, Lézard, Spock). Le représenter en *Python*.



3. Écrire une fonction Python appelée shifumi(G, nb_manches) qui permet de jouer un duel entre l'utilisateur et l'ordinateur (dont les figues sont choisies aléatoirement et de manière équiprobables) en nb_manches confrontations successives, et selon les règles représentées par le graphe G, et qui affiche le score final.
4. Soit n un entier impair. Écrire une fonction graphe_shifumi(n) qui renvoie un graphe de *shifumi* équilibré à n sommets.

3.4. **Le truel.** Trois gentleman, Mr. White, Mr. grey and Mr. Black se retrouvent opposés dans un *truel*, où chaque adversaire tire à son tour jusqu'à ce qu'il n'en reste qu'un.

Les trois hommes n'ont pas le même niveau de tire : Mr. Black fait mouche à chaque tire, Mr. grey touche son adversaire deux fois sur trois, tandis que Mr. White ne réussit son tire qu'une fois sur trois. Les hommes sont des gentlemen, ils laissent donc Mr. White tirer en premier, puis Mr. Grey et enfin Mr. Black.

Mr. Black visera toujours, tant que celui-ci sera vivant, Mr. Grey ; et Mr. Grey essaiera toujours lui-aussi de viser Mr. Black. Mr. White en revanche se demande s'il doit commencer par faire tomber Mr. Black, ou Mr. Grey, voire tirer en l'air et les laisser s'entretuer, donnant ainsi lieu à trois stratégies, numérotées 1, 2 et 3.

Comparer les trois stratégies avec Python, via simulation d'un grand nombre de truels pour les trois stratégies et émettre une conjecture sur celle la plus intéressante à suivre pour Mr. White. On pourra commencer par écrire une fonction qui simule les duels entre Mr. White et Mr. Grey avec dans chaque cas, un premier tireur différent.

3.5. **La ruine du joueur.** José se rend au casino *Les requins de la côte* avec s euros en poche ($s \in \mathbb{N}^*$) et l'envie de faire fortune. Après s'être vêtu de ses habits de lumière, il s'installe à une table où, à chaque partie, il gagne avec probabilité $p \in]0,1[$ 1 euro et perd avec probabilité $q = 1 - p$ un euro.

On note N la somme dont dispose le casino (on peut raisonnablement supposer que $s < N$). José décide qu'il arrêtera de jouer s'il devient ruiné (c'est-à-dire lorsque sa fortune tombe à 0) ou lorsque ce sera le cas pour le casino. On admet (même si on pourrait le montrer) que le jeu s'arrête à un moment presque sûrement.

1. Écrire une première fonction casino(s,N,p) qui à chaque appel renvoie le graphe de l'évolution de la fortune de José jusqu'à l'arrêt du processus.
2. On introduit la variable aléatoire T qui prend la valeur 1 si José est ruiné ou 0 si le casino est ruiné. Établir une conjecture sur la loi de T.

3.6. **Théorème de Zeckendorf.** On rappelle que la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est la suite de nombre réels définis par

$$F_0 = 0, \qquad F_1 = 1 \qquad \text{et} \qquad F_{n+2} = F_{n+1} + F_n.$$

Les premiers termes de la suite sont 0,1,1,2,3,5,8,13,21,34,55,...

Théorème : Théorème de Zeckendorf

Pour tout entier naturel $n \geqslant 1$, il existe un unique entier k et un unique k-uplet d'entiers $(c_1, c_2, \cdots, c_k)$ vérifiant

- $c_1 \geqslant 2$.

- $c_i + 1 < c_{i+1}$

tels que

$$n = \sum_{i=1}^{k} F_{c_i}.$$

Cette décomposition de n en somme de nombres de la suite de Fibonacci s'appelle la décomposition de Zeckendorf de n.

Par exemple, $17 = 13 + 3 + 1 = F_7 + F_4 + F_2$ donc $k = 3$ et $(c_1, c_2, c_3) = (2, 4, 7)$.

Le but de cette activité est d'écrire un programme qui renvoie cette décomposition pour un nombre n pris en argument.

1. Écrire une fonction `fibo(k)` qui renvoie le terme F_k de la suite $(F_n)_{n\in\mathbb{N}}$.
2. Écrire une fonction `tri(L)` qui renvoie la liste des termes de `L` triés par ordre croissant.
3. Écrire une fonction `recherche(x,L)` qui prend en argument un réel x et une liste L (déjà triée dans l'ordre croissant), de premier terme inférieur ou égal à x et de dernier terme strictement supérieur à x et qui renvoie le plus grand élément de la liste inférieur ou égal à x.
4. Écrire une fonction `Zeckendorf(n)` qui renvoie la décomposition de Zeckendorf de n.

   On pourra commencer par écrire la liste des nombres de Fibonacci inférieurs à n (ainsi que le premier strictement supérieur) et faire une boucle descendante sur cette liste.