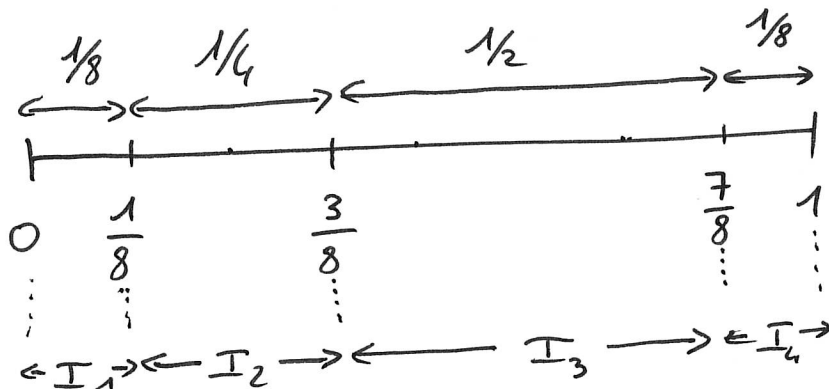


DN 6 Python

(1)

1. On voit :



2. L'énoncé précise que

$$[0, 1] = \bigcup_{i=1}^m I_k ; \text{ et il est clair que cette union est } \underline{\text{disjointe}}$$

Un tirage de U est un réel de $[0, 1]$, qui appartient donc à un intervalle I_k et un seul : k existe et est unique

3. Pour tout $k \in [1, n-1]$

$$U \in I_k \Leftrightarrow q_{k-1} \leq U < q_k \quad \text{donc} \quad P(U \in I_k) = P(q_{k-1} \leq U < q_k) \\ = P(q_{k-1} < U \leq q_k) \\ (U \text{ étant à densité})$$

$$\Rightarrow P(U \in I_k) = F_U(q_k) - F_U(q_{k-1})$$

$$\text{Or } (q_{k-1}, q_k) \in [0, 1]^2 \text{ donc } F_U(q_{k-1}) = q_{k-1} ; F_U(q_k) = q_k$$

et $P(U \in I_k) = q_k - q_{k-1} = p_k$

Pour $k=n$ le calcul est identique avec

$P(U \in I_n) = P(q_{n-1} \leq U \leq q_n)$; mais le passage de l'inégalité stricte à l'inégalité large ne change rien.

4) X est le numéro de l'intervalle où se trouve la valeur de U

et où $X(\omega) = \lfloor 1, n \rfloor$;

et $\forall k \in \lfloor 1, n \rfloor, P(X=k) = P(U \in I_k) = \underline{p_k}$

\Rightarrow La variable X suit bien la loi demandée

5) Avec $\sum_{i=0}^{k-1} p_i = q_{k-1}$ et $\sum_{i=0}^k p_i = q_k$

on a $\left[\sum_{i=0}^{k-1} p_i < U \text{ et } \sum_{i=0}^k p_i \geq U \right]$

$\Leftrightarrow [q_{k-1} < U \leq q_k]$

Il y a donc une erreur d'écriture : lire $\sum_{i=0}^{k-1} p_i \leq U$ et $\sum_{i=0}^k p_i \geq U$

auquel cas les conditions proposées équivalent à

$[q_{k-1} \leq U < q_k]$ ce qui est bien : $U \in [q_{k-1}, q_k[= I_k$

6. Visuellement :

On tire une valeur de U ; on initialise un curseur à 0 et on le décale à chaque tour de boucle :

$$0 \xrightarrow{+p_1} q_1 \xrightarrow{+p_2} q_2 \xrightarrow{+p_3} q_3 \rightarrow \dots \rightarrow q_{k-1} \xrightarrow{+p_k} q_k$$

Jusqu'à "dépasser" U . On renvoie le numéro de l'étape où U a été dépassé.

7. Notons s ce curseur ; et i le numéro du segment considéré.

Initialement $s=0$ et $k=0$

def trace(L) # L est la liste $[p_1, \dots, p_n]$

$u = \text{rd.random}()$ # trace de U

$s = 0$

$k = 0$

while $U \geq s$

$s = s + L[k]$

$k = k + 1$

return k

8. On peut faire tourner beaucoup de fois la fonction, compter le nombre d'occurrences de chaque valeur de U , et vérifier que la fréquence d'apparition de la valeur " i " est $P(U=i) = p_i$.

On peut par exemple coder

$L = [1/8, 1/4, 1/2, 1/8]$ # pour reprendre l'exemple du début de sujet

compteur = np.zeros(len(L)) # liste de n compteurs : la composante d'indice $i-1$ comptera le nb de tirages égaux à i

for k in range(100000):

tirage = tirage(L)

compteur[tirage - 1] = compteur[tirage - 1] + 1

print(compteur / 100000) # pour faire apparaître les fréquences.

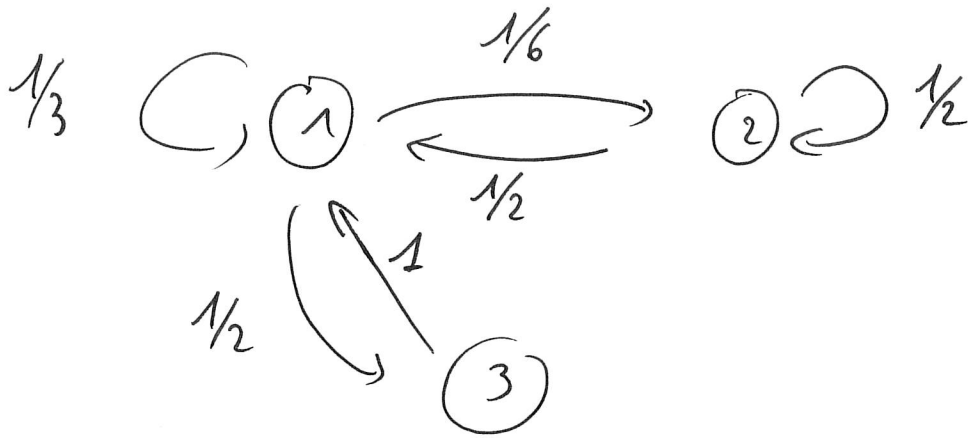
Ceci renvoie $[0.12413, 0.24991, 0.50059, 0.12537]$

alors que $L = [0.125, 0.25, 0.5, 0.125]$: pas mal !

9. C'est essentiellement la même chose ; sauf qu'au lieu de renvoyer k en sortie, on renvoie la k-ième valeur spécifiée qui est $X[k-1]$

=> même code qu'en 7, à part la ligne finale qui devient

| return X[k-1] |



11. Ce code diagonalise la matrice ${}^t\Pi$:

ss 3 valeurs propres sont $\lambda_1 \approx -0,5902$
 $\lambda_2 = 1$
 $\lambda_3 \approx 0,4235$

et 3 rep associés sont: $U_1 = \begin{pmatrix} 0,757... \\ -0,415... \\ -0,642... \end{pmatrix}$, $U_2 = \begin{pmatrix} 0,857... \\ -0,285... \\ -0,428... \end{pmatrix}$, $U_3 = \begin{pmatrix} 0,376... \\ -0,815... \\ 0,441... \end{pmatrix}$
 (respectiv^t)

$$\text{(former les 3 colonnes de } P \text{ tq } {}^t\Pi = P \begin{pmatrix} \lambda_1 & (0) \\ (0) & \lambda_2 \\ & & \lambda_3 \end{pmatrix} P^{-1}$$

On sait que les états stables sont les vecteurs propres de ${}^t\Pi$ pour la rap $\lambda = 1$; de composantes positives et de somme égale à 1.

On s'intéresse donc à la seconde colonne de P , qu'on peut isoler par :

$$\underline{\underline{P[:, 1]}}$$

On la "normalise" en divisant par la somme de ss composantes

$$\underline{\underline{P[:, 1] / \text{sum}(P[:, 1])}} \text{ renvoie } [0.5655., 0.1818., 0.2727...]$$

qui est donc l'unique état stable de la chaîne.

(NB : en fait on a aussi $(\frac{6}{11}, \frac{2}{11}, \frac{3}{11})$ pour unique état stable)

12. Ce programme prend le vecteur $V = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$

et affiche

$$\begin{array}{l} V \Pi \\ V \Pi^2 \\ V \Pi^3 \\ \vdots \\ V \Pi^{20} \end{array}$$

On observe que les valeurs affichées se stabilisent autour de l'état stable donné à la question précédente; et que cela ne dépend pas du V initial

(essayer avec $[1, 0, 0]$; $[0, \frac{1}{2}, \frac{1}{2}]$; $[\frac{1}{5}, \frac{2}{5}, \frac{2}{5}]$; etc.)

Quelle que soit la distribution initiale, la loi de la chaîne converge vers l'état stable (résultat déjà observé en cours)

13. Par propriété de Markov, le choix de l'état suivant dépend de l'état actuel.

Si $X_p = i$, on aura $X_{p+1} = j$ avec proba $P_{(X_p=i)}(X_{p+1}=j)$

Sachant $X_p = i$, l'état suivant ~~est~~ suit la loi

valeur	1	...	j	...	n
proba	$P_{(X_p=i)}(X_{p+1}=1)$		$P_{(X_p=i)}(X_{p+1}=j)$		$P_{(X_p=i)}(X_{p+1}=n)$

On reconnaît ds cette loi de proba la i -ème ligne de la matrice de transit° (donc la ligne d'indice $i-1$, obtenue par $\Pi[i-1, :]$) (4)

def trajectoire(Π , état_initial, nb_etaps)

traj = [état_initial] # liste des posit° successives, qui contiennent le point de départ.
for k in range(nb_etaps):

traj.append(trage($\Pi[\text{traj}[-1]-1, :]$)) (*)

return np.array(traj)

(*) explicat (c'est 1 version assez condensée)

- à tout instant, $\text{traj}[-1]$ est la dernière composante de la liste traj , ie. l'état actuel
- la loi de proba à utiliser se trouve donc dans la ligne $\Pi[\text{traj}[-1]-1]$
- on tire alors la posit° suivante avec la font° de la partie I avec $\text{trage}(\Pi[\text{traj}[-1]-1, :])$
- et on ajoute ce nouvel état à la trajectoire (append)

14. Une matrice en Python étant la liste de ses lignes, il suffit de ranger dans une liste les résultats renvoyés par la fonction précédente pour chaque état de la liste des états initiaux

On appellera `population_init` la liste des états initiaux et on écrit :

```
def trajectoires(M, population_init, nb_etapes):  
    res = []  
    for k in population_init:  
        res.append(trajectoire(M, k, nb_etapes))  
    return np.array(res)
```

appel : on peut itérer directement sur les valeurs contenues dans la liste `population_init`, au lieu de manipuler les indices

(essai : `L = [5, 4, 1] // for k in L: print(k)`)

15. Idée similaire à la question 8 : on va balayer la liste, et compter les "1", les "2" et les "3". Ces compteurs seront rangés ds 1 liste à 3 el's, initialisée à `[0, 0, 0]`

(5)

```
def frequences(L):
    compteurs = np.zeros(3)
    for k in L:
        if k in [1, 2, 3]:
            compteurs[k-1] = compteurs[k-1] + 1

    return (compteurs / len(L))
```

On peut aussi compter le nombre d'occurrences de "1" dans L par
 $\text{sum}(L == 1)$; d'ailleurs \triangle si L est un np.array \triangle

```
def frequences(L):
    return ([sum(L == k) for k in range(1, 4)]) / len(L)
```

np.array
 ↑
 pour pouvoir diviser par len(L) : opérat. non permise sur une liste Python.

Inconvénient de ce second code : il parcourt 3 fois la liste, alors que le premier ne la parcourt qu'une fois (mais on n'attend pas de vous ce genre de préoccupations...)

16. $evol = \text{trajectories}(M, \text{np.ones}(10 \times 6, \text{dtype} = \text{int}), 20)$
d'après la fonction de cette fonction.

17. la dernière colonne s'obtient par
 $C = evol[:, 20]$ (colonne d'indice 20)
(ou aussi $C = evol[:, -1]$: dernière colonne)

le résultat est $[0.54385, 0.18314, 0.27301]$

↗
fract° de "1"
de la colonne
= fract° d'individus
de l'état "1" au bout
de 20 étapes de t_p
↖ ↗
idem pour
"2" et "3"

la dernière colonne de C est en fait constituée de 100000 traces de $X_{20} \dots$ les fréquences d'apparition de chaque valeur donnent une approximation de leur probabilité. le résultat affiché est donc une valeur approchée de la loi de X_{20} ; qui est proche de l'état stable de la question 11.

18. Par le même argument, cette liste de 21 listes à 3 composantes donne une approximation des lois de X_0 (constante = 1, cf les condit° initiales utilisées ici)
 X_1
 X_2
 \vdots
 X_{20}
et on observe cette fois la convergence en loi de (X_n) vers son état stable

19. Non... et on en a souvent parlé en cours!

(6)

NB : une liste de 100000 nbs égaux à 1, 2, 3 de manière équiprobable s'obtient par `rd.randint(1, 4, 100000)`

20. Il suffit d'appliquer la fonction `frequencies` à une trajectoire, générée par la `fact` de la question 13

`print(frequencies(trajectoire(17, 1, 100000)))` en "2", en "3" : état initial au choix

NB : on parle de propriété ergodique quand les statistiques "temporelles" (un individu au cours du temps) sont égales aux statistiques "spatiales" (un gd nb d'individus, à une étape de tps fixée)

