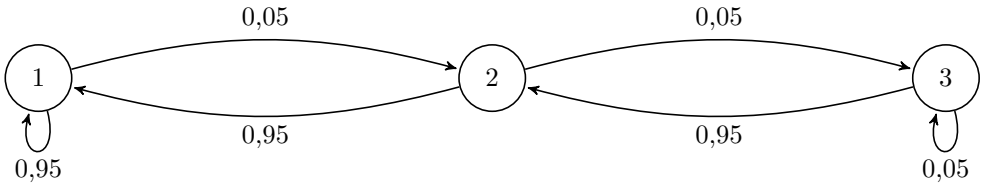
Correction - TP 5

Chaînes de Markov

Exercice 1 (Système de Bonus-Malus des assurances)

1. La variable X_{n+1} dépend uniquement de X_n et pas des précédentes puisque, pour prévoir le taux d'un assuré à l'instant $n+1$, il suffit de savoir son taux à l'instant n. Donc $(X_n)_{n\in\mathbb{N}}$ est bien une chaîne de Markov.

   Voici le graphe probabiliste de la chaîne de Markov :



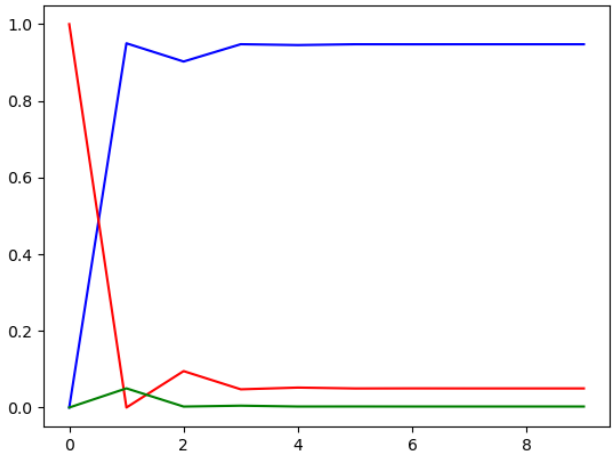
2. L'ensemble des états est $E = [\![1, 3]\!]$. La matrice de transition est la matrice carrée d'ordre 3 suivante :

$$A = \begin{pmatrix} 0.95 & 0.05 & 0 \\ 0.95 & 0 & 0.05 \\ 0 & 0.95 & 0.05 \end{pmatrix}.$$

3. On utilise les commandes suivantes :

```
A = np.array([[0.95, 0.05, 0], [0.95, 0, 0.05], [0, 0.95, 0.05]])
U = np.array([0, 1, 0])
E = np.zeros((10,3))
E[0, :] = U
for k in range(1,10):
    U = np.dot(U, A)
    E[k, :] = U

n = np.arange(0,10)
plt.plot(n, E[:, 0], color = 'blue')
plt.plot(n, E[:, 1], color = 'red')
plt.plot(n, E[:, 2], color = 'green')
plt.show()
```

On obtient le graphique suivant :



On remarque que la suite des lois des X_n semble converger vers un état stable

$$U \simeq \begin{pmatrix} 0.95 & 0.04 & 0.01 \end{pmatrix}.$$

4. (a) On utilise l'instruction suivante :

```
>>> Sp, VP = al.eig(A)
>>> Sp

array([1.     , - 0.2179449    ,  0.2179449])

>>> VP

array([[-0.57735027,   0.01162034,  -0.0118893 ],
       [-0.57735027,  -0.2714383 ,   0.17407247],
       [-0.57735027,   0.96238569,   0.98466107]])
```

On peut montrer par récurrence que, pour tout $n \in \mathbb{N}$, $A^n = PD^nP^{-1}$.
Notons $D = diag(1, \lambda_2, \lambda_3)$. Alors $D^n = diag(1, \lambda_1^n, \lambda_2^n) \underset{n\to+\infty}{\longrightarrow} diag(1, 0, 0)$ car $\lambda_1, \lambda_2 \in]-1, 1[$. On en déduit que :

$$\lim_{n\to+\infty} A^n = P \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} P^{-1}.$$

On effectue ce produit à l'aide de `Python` :

```
>>> Dlim = np.array([[1, 0, 0], [0, 0, 0], [0, 0, 0]])
>>> Alim = np.dot(np.dot(VP, Dlim), al.inv(VP))
>>> Alim

array([[0.9475066,    0.0498688,    0.0026247],
       [0.9475066,    0.0498688,    0.0026247],
       [0.9475066,    0.0498688,    0.0026247]])
```

(b) On peut démontrer que pour tout $n \in \mathbb{N}$, $U_n = U_0A^n$ donc en passant à la limite on obtient $U = U_0 \times (\lim_{n\to+\infty} A^n)$. On obtient alors la limite U de (U_n) en faisant le produit par U_0 :

```
>>> U0 = np.array([0, 1, 0])
>>> np.dot(U0, Alim)

array([0.9475066,  0.0498688,  0.0026247])
```

(c) On sait que $U_n = U_0 \times A^n$ converge vers un état stable U. En prenant n suffisamment grand, on obtient donc une approximation de U proche du résultat obtenu à la question précédente. On vérifie que c'est bien le cas :

```
>>> np.dot(U0, al.matrix_power(A,100))

array([0.9475066,  0.0498688,  0.0026247])
```

On retrouve bien le résultat précédent.

5. (a) Voici la matrice de transition dans le cas d'un mauvais conducteur :

$$B = \begin{pmatrix} 0.85 & 0.15 & 0 \\ 0.85 & 0 & 0.15 \\ 0 & 0.85 & 0.15 \end{pmatrix}.$$

(b) On utilise la commande `al.eig` :

```
>>> B = np.array([[0.85, 0.15, 0], [0.85, 0, 0.15], [0, 0.85, 0.15]])
>>> Sp, VP = al.eig(np.transpose(B))
>>> Sp

array([1.,     -0.3570714,     0.3570714])

>>> VP

array([[-0.9843208    0.5596199  -0.8130422],
       [-0.1737037   -0.7947072   0.4714962],
       [-0.0306536    0.2350874   0.3415460]])
```

On considère alors la colonne de la matrice `VP` associée à la valeur propre 1, c'est-à-dire la première colonne. Pour obtenir la loi stationnaire `V` et donc un vecteur stochastique, on divise par la somme des coefficients :

```
>>> V = VP[:, 0] / np.sum(VP[:, 0])
>>> V

array([0.8280802,   0.1461318,   0.0257880])
```

6. (a) On note M l'événement : "l'assuré est un mauvais conducteur". Pour tout $i \in [\![1,3]\!]$, on note également A_i l'événement : "l'assuré est au taux 3 après un nombre d'année important". On cherche $P_{A_3}(M)$. Avec la formule des probabilités conditionnelles puis des probabilités composées :

$$P_{A_3}(M) = \frac{P(A_3 \cap M)}{P(A_3)} = \frac{P(M \cap A_3)}{P(A_3)} = \frac{P(M)P_M(A_3)}{P(A_3)}.$$

Or, avec le système complet d'événements $(M, \overline{M})$:

$$P(A_3) = P((M \cap A_3) \cup (\overline{M} \cap A_3)) = P(M \cap A_3) + P(\overline{M} \cap A_3) \quad \text{(par incompatibilité)}.$$

Avec la formule des probabilités composées :

$$P(A_3) = P(M)P_M(A_3) + P(\overline{M})P_{\overline{M}}(A_3).$$

Finalement :

$$P_{A_3}(M) = \frac{P(M)P_M(A_3)}{P(M)P_M(A_3) + P(\overline{M})P_{\overline{M}}(A_3)} \simeq \frac{0.5 \times 0.0257880}{0.5 \times 0.0257880 + 0.5 \times 0.0026247} \simeq 0.9076223$$

Il y a donc près de 91% de chances, lorsque l'assuré est au taux 3, pour qu'il soit un mauvais conducteur. Le taux 3 est donc bien discriminant : il s'applique principalement aux mauvais conducteurs.

(b) On cherche $P_{A_1}(\overline{M})$. Avec le même raisonnement qu'à la question précédente, on obtient :

$$P_{A_1}(\overline{M}) = \frac{P(\overline{M})P_{\overline{M}}(A_1)}{P(M)P_M(A_1) + P(\overline{M})P_{\overline{M}}(A_1)} \simeq \frac{0.5 \times 0.9475066}{0.5 \times 0.8280802 + 0.5 \times 0.9475066} \simeq 0.5336301$$

Il y a donc près de 53% de chances, lorsque l'assuré est au taux 1, pour qu'il soit un bon conducteur. Le système de Bonus-Malus mis en place par l'assureur n'est pas satisfaisant : il y a quasiment autant de mauvais conducteurs que de bons conducteurs qui bénéficient du taux le plus favorable (le 1).

Exercice 2 (Ruine du joueur)

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 :

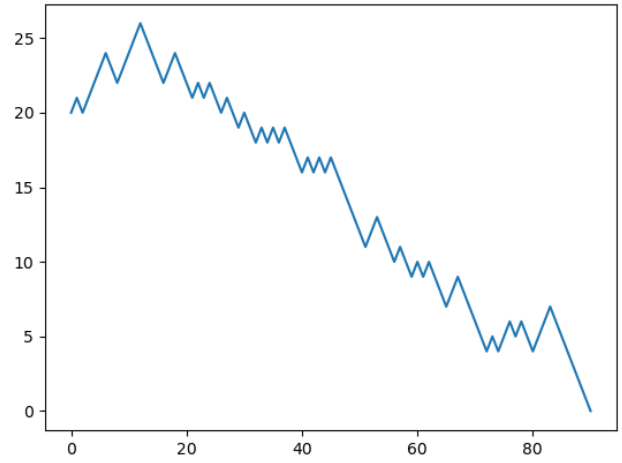
$$\begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$$

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. (a) Grâce à ce programme, on trace le graphique de la suite (X_n) (en ordonnée) en fonction de l'instant n (en abscisse).
On exécute plusieurs fois ce programme. On remarque que le joueur est systématiquement ruiné et le jeu s'arrête, la plupart du temps en moins d'une centaine de parties. Par exemple :



(b) On remarque que si $p < 1/2$ (p éloigné de $1/2$), le joueur est systématiquement ruiné, que si $p > 1/2$ (p éloigné de $1/2$), le casino est systématiquement ruiné (même s'il est beaucoup plus riche), et si $p = 1/2$, le joueur peut être ruiné mais le casino aussi.
Dans tous les cas, le jeu s'arrête au bout d'un temps fini : la chaîne de Markov finit toujours dans un état absorbant.

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 tape par exemple :

```
>>> ruinejoueurfrequence(4/5,20,300,1000)
0.0
>>> ruinejoueurfrequence(2/5,20,300,1000)
1.0
>>> ruinejoueurfrequence(0.51,20,300,1000)
0.452
>>> ruinejoueurfrequence(0.499,20,300,1000)
0.969
```

On remarque que le joueur est systématiquement ruiné si $p < 1/2$ (lorsque p n'est pas trop proche de $1/2$), le casino est systématiquement ruiné lorsque $p > 1/2$ (lorsque p n'est pas trop proche de $1/2$) et, lorsque p est très proche de $1/2$, chacun a ses chances de ruiner l'autre.

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 $+\infty$ (c'est le théorème de la loi faible des grands nombres). Il faudrait donc faire une infinité de simulations de cette expérience pour obtenir la valeur exacte de la probabilité de la ruine du joueur.

Exercice 3 (Google PageRank)

1. On note L_i l'événement : "choisir de passer de la page numéro i à une autre page ayant un lien avec la page i" et donc $\overline{L_i}$: "choisir de passer de la page i à une page au hasard du web".

   On a alors, par la formule des probabilités totales :

   $$a_{i,j} = P_{(X_n=i)}(X_{n+1}=j) = \underbrace{P(\overline{L_i})}_{=c}\underbrace{P_{\overline{L_i}}(X_{n+1}=j)}_{=1/N} + \underbrace{P(L_i)}_{=1-c}\underbrace{P_{L_i}(X_{n+1}=j)}_{=0 \text{ ou } 1/\ell_i},$$

   en fonction de si la page i pointe vers la page j. On obtient bien la formule de l'énoncé.

2. A chaque instant, l'internaute à une probabilité c de quitter la page pour se diriger vers une page quelconque du web et $1-c$ de se diriger vers une page ayant un lien avec elle. Ces choix sont indépendants les uns des autres.

   Si on note X la variable aléatoire égale au nombre de pages consultées par l'internaute jusqu'à ce qu'il relance au hasard sa navigation, alors X suit donc une loi géométrique de paramètre c. Donc $E(X) = \dfrac{1}{c}$ qui doit être égale à 6 d'après l'énoncé. Donc $c = \dfrac{1}{6} \simeq 0.15$.

3. (a) Voici la matrice de transition associée au diagramme :

   ```
   >>> A = 0.15*1/5*np.ones((5,5))+0.85*np.array([[0, 1/3, 0, 1/3, 1/3],
   [1/2, 0, 0, 1/2, 0], [0, 1/2, 0, 0, 1/2], [0, 1/2, 0, 0, 1/2], [0, 1, 0, 0, 0]])
   ```

   (b) On utilise la commande `al.eig` :

   ```
   >>> Sp, VP = al.eig(np.transpose(A))
   >>> Sp

   array([1.,   -0.341857+0.436921j,   -0.341857-0.436921j,   -0.166285,   0.])
   ```

   On considère alors la colonne de `VP` associée à la valeur propre 1, c'est-à-dire la première colonne. Pour obtenir la loi stationnaire `U` et donc un vecteur stochastique, on divise par la somme des coefficients :

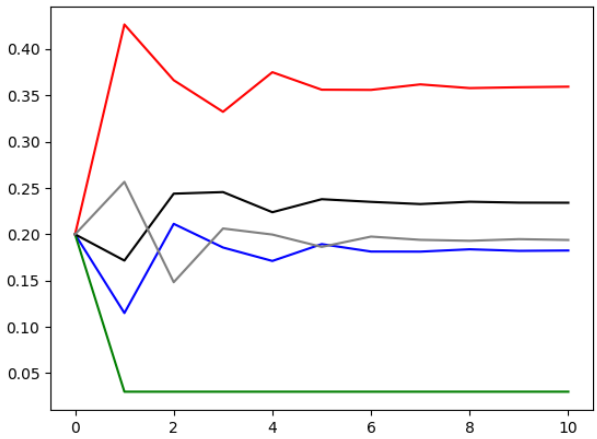
   ```
   >>> U = VP[:,0] / np.sum(VP(:,0))
   >>> V

   np.array([0.182586,  0.359026,  0.03,   0.234319,  0.194068])
   ```

   (c) On utilise les commandes suivantes :

```
1  A = 0.15*1/5*np.ones((5,5))+0.85*np.array([[0, 1/3, 0, 1/3,
   1/3], [1/2, 0, 0, 1/2, 0], [0, 1/2, 0, 0, 1/2], [0, 1/2, 0,
   0, 1/2], [0, 1, 0, 0, 0]])
2  U = np.array([1/5, 1/5, 1/5, 1/5, 1/5])
3  E = np.zeros((11,5))
4  E[0, :] = U
5  for k in range(1,11):
6      U = np.dot(U, A)
7      E[k, :] = U
8
9  n = np.arange(0,11)
10 plt.plot(n, E[:,0], color = 'blue')
11 plt.plot(n, E[:,1], color = 'red')
```

```
12 plt.plot(n, E[:,2], color = 'green')
13 plt.plot(n, E[:,3], color = 'black')
14 plt.plot(n, E[:,4], color = 'gray')
15 plt.show()
```

On obtient le graphique suivant :



On remarque que la suite des lois des X_n semble converger vers l'état stable trouvé précédemment.

(d) La page 2 a le plus gros indice de popularité, suivie des pages 4, 5, 1 puis 3.

Pour qu'une page soit bien référencée dans Google, il faut donc qu'elle soit la cible de beaucoup de liens externes et qu'elle ait peu de liens vers les autres pages. Autrement dit, c'est une page qui se rend très populaire sans faire de pubs pour les autres pages !
