TP Chaînes de Markov

Ben Daoud Hamza

January 2023

1 Exercice 1

1.1 Question 1

Pour répondre à cette question, on trace le graph qui régit les interactions entre les différents états, soit :

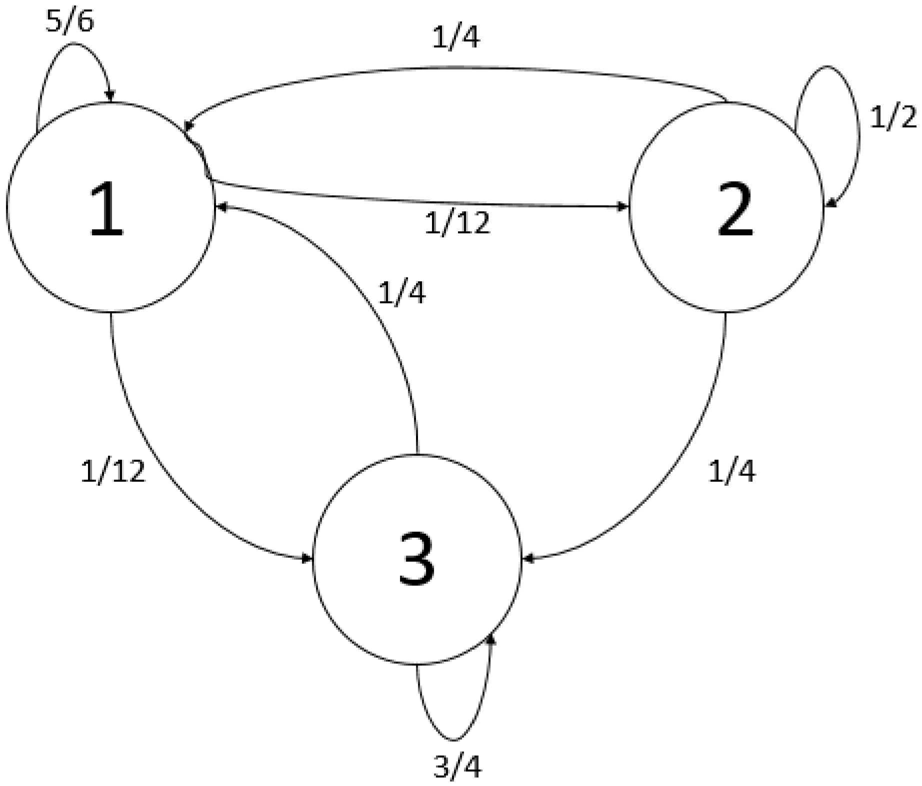

Figure 1 – graph du problème

Ainsi, en analysant le graph la matrice de transition s'écrit :

$$\begin{pmatrix} \frac{5}{6} & \frac{1}{12} & \frac{1}{12} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{3}{4} \end{pmatrix}$$

$\rightarrow$ On remarque ainsi du graphe qu'il existe une seule classe d'équivalence à savoir$\{1, 2, 3\}$ ou tous les états communiquent,ie : $1 \leftrightarrow 2 \leftrightarrow$ 3 en déduit que la chaine est irréductible.
$\rightarrow$ la chaine est bien apériodique, cela peut être justifié par plusieurs méthodes : on peut remarquer que le PGCD de tous les états est égale à 1, on peut encore remarquer que tous les P_{ii} sont strictement positifs, le graph possède ainsi une boucle et l'état 1 par exemple est apériodique, or la chaine est irréductible réccurente, tous les états seront apériodique est donc le graph est apériodique.

1.2 Question 2

L'idée pour résoudre un système de type $\mu.P = \mu$ en python est d'utiliser la méthode "np.linalg.eig()" de la bibliotèque numpy, pour determiner les valeurs(eigenvalues) et les vecteurs propres(eigenvectors) de la matrice de transition, puis on prend les vecteurs propres ayant comme valeur propre 1"dans le script on prend ces valeurs propre avec un erreur de 10^{-6} pour facilité la tache à l'algorithme puisque les valeurs de la matrice de transition ne sont pas précis avec un nombre de chiffre après virgule infini, mais notre problème s'écrit $\mu.P = \mu$ c'est-à-dire que le vecteur propre est à gauche, on doit ainsi chercher les valeurs propres de la transposé.soit le script suivant :

```python
"EX1-QST2"
import numpy as np

A = [[5/6, 1/12, 1/12],
     [1/4, 1/2, 1/4],
     [1/4, 0, 3/4]]

eigenvalues, eigenvectors = np.linalg.eig(np.transpose(A))

# Search for the eigenvector corresponding to eigenvalue 1
for i in range(len(eigenvalues)):
    if abs(eigenvalues[i] - 1.0) < 1e-6:
        invariant_probability = eigenvectors[:,i]
        print(invariant_probability)
```

FIGURE 2 – probabilité invariante

```
[0.6 0.1 0.3]
```

FIGURE 3 – Résultat

1.3 Question 3

$\rightarrow$Si Adam est en bonne santé le 1 Janvier la loi initiale correspend tous simplement à : $\mu_0 = [1, 0, 0]$.
$\rightarrow$ Maintenant pour les autres valeurs on peut utiliser un compilateur python pour calucluer à chaque fois $\mu_0.P^n$. soit le script suivant :

```python
import numpy as np

def matrix_power(P, n):
    mu_0 = np.array([1, 0, 0])
    result = mu_0
    for i in range(n):
        result = np.dot(result, P)
    return result

P = np.array([[5/6, 1/12, 1/12], [1/4, 1/2, 1/4], [1/4, 0, 3/4]])
n = 5
m=10
i=50
l=100
print(matrix_power(P, n))
print(matrix_power(P, m))
print(matrix_power(P, i))
print(matrix_power(P, l))
```

```
[0.62701743 0.11139243 0.26159015]
[0.60182485 0.10133657 0.29683857]
[0.6 0.1 0.3]
[0.6 0.1 0.3]
```

FIGURE 4 – probabilité μ_n, première condition initiale

ou l'on définit la méthode *matrixpower* qui retourne le produit demandé pour n quelconque, puis on applique cette méthode pour n=5,10,50,100.
On observe que lorsque n devient plus grande, de l'ordre de 100 la probabilité μ_n tend vers la probabilité invariante μ_0 d'ou la convergence en loi.

1.4 Question 4

Pour un autre cas, exemple de cas de maladie $\mu 0$=[0,0,1] on refait la même procédure qu'avant, le résultat est le suivant :

```python
import numpy as np

def matrix_power(P, n):
    mu_0 = np.array([0, 0, 1])
    result = mu_0
    for i in range(n):
        result = np.dot(result, P)
    return result

P = np.array([[5/6, 1/12, 1/12], [1/4, 1/2, 1/4], [1/4, 0, 3/4]])
n = 5
m=10
i=50
l=100
print(matrix_power(P, n))
print(matrix_power(P, m))
print(matrix_power(P, i))
print(matrix_power(P, l))
```

```
[0.55947386 0.07509886 0.36542728]
[0.59726272 0.097751   0.30498628]
[0.6 0.1 0.3]
[0.6 0.1 0.3]
```

FIGURE 5 – probabilité μ_n,deuxième condition initiale

$\rightarrow$ On remarque qu'on a encore convergence en loi.

1.5 Question 5

Idem on génère un script python pour le calcule de puissance en utilisant le méthode np.linalg.matrixpower de la bibliotèque numpy, qui est régit dans le script çi-dessous.
On remarque que pour n assez grand, A_n tend vers la probabilité invariante(cas pour n =100) et donc tous les états sont équiprobables dans ce cas de probabilité égale à μ_0

```
def power_n(A,n):
  return np.linalg.matrix_power(A, n)



print("si n=5 : ",power_n(A,5))
print("si n=10 : ",power_n(A,10))
print("si n=50 : ",power_n(A,50))
print("si n=100 : ",power_n(A,100))
```

```
si n=5 :  [[0.62701743 0.11139243 0.26159015]
 [0.55947386 0.10634886 0.33417728]
 [0.55947386 0.07509886 0.36542728]]
si n=10 :  [[0.60182485 0.10133657 0.29683857]
 [0.59726272 0.09872756 0.30400972]
 [0.59726272 0.097751   0.30498628]]
si n=50 :  [[0.6 0.1 0.3]
 [0.6 0.1 0.3]
 [0.6 0.1 0.3]]
si n=100 :  [[0.6 0.1 0.3]
 [0.6 0.1 0.3]
 [0.6 0.1 0.3]]
```

FIGURE 6 – les puissance de A

1.6 Question 6

La fonction peut être complétée de la manière suivante :

```
def adam(X):
    U= np.random.rand(1)[0]
    if (X==1):
      return 1*(U<5/6)+2*(5/6< U < 11/12)+3*(11/12<U)

    if (X==2):
      return 1*(U < 1/4)+2*(1/4 <U < 3/4)+3*(3/4<U)
    if (X==3):
      return 1*(U < 1/4)+3*(1/4 < U)
```

FIGURE 7 – suite de l'algorithme

1.7 Question 7

Une autre manière de visualiser la convergence en loi serait de représenter pour un nombre d'itération de grand ordre la chaine ie génerer plusieurs simulations et observer concrètement cette convergence.c'est le principe de Monte Carlo.

2 Exercice 2

2.1 Question 1

Dans le cas de l'exercice, on a deux états qui garantissent l'arrêt du processus qu'on appèle les élements absorbants de la chaîne à savoir :l'état perte surnommé 0 dans l'exercice, ou Morty perd tous sa fortune, et l'état victoire ou Morty gange 8 euros surnommé 8 dans l'exercice.
Pour prendre compte de ces coditions d'arrêt, il faut qu'on reste dans ces états infinement ie la probabilité de transition de ces états à d'autres est nulle ou encore : $P_{00} = 1$ ou $P_{88} = 1$.

2.2 Question 2

soit le graphe suivant :

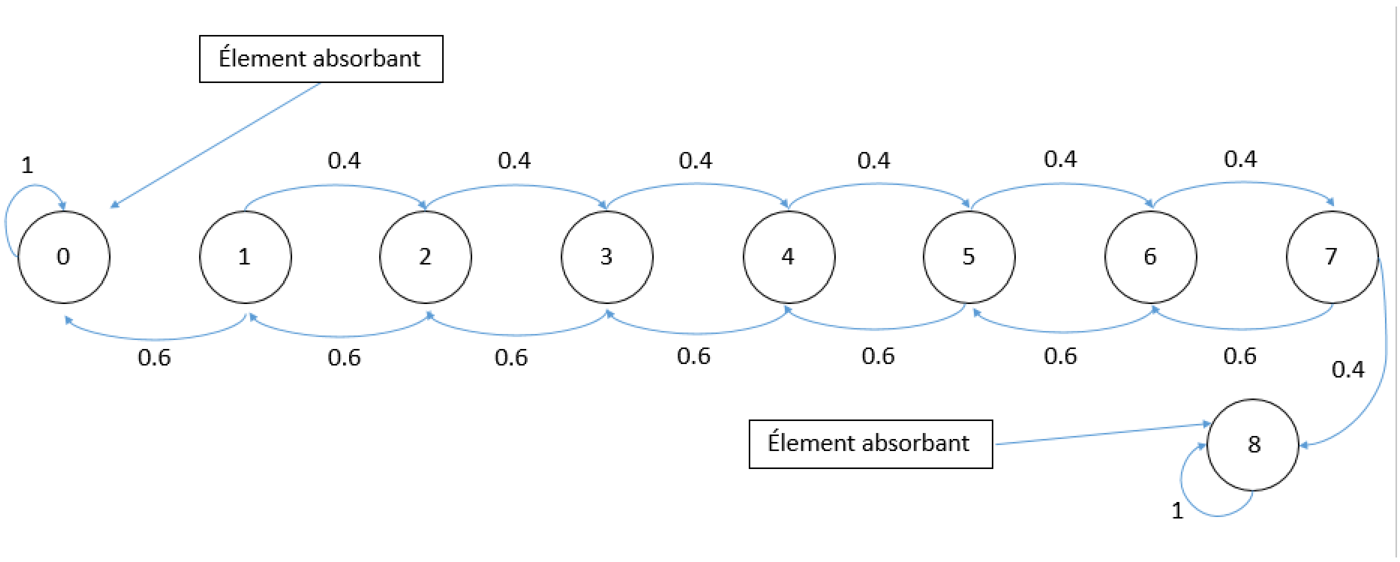

FIGURE 8 – graphe de la chîne

$\rightarrow$ Vu les conditions d'arrêts imposées, il existe 3 classes communiquantes à savoir, $\{0\}$, $\{8\}$ et $\{1, 2, 3, 4, 5, 6, 7, 8\}$, donc la chaîne n'est pas réductible. $\rightarrow$ les composantes fermées sont ainsi l'état 0 et 1 car leur origine et extrémité coincident.

2.3 Question 3

La matrice de transition peut être donné par le script suivant, il suffit de donner 1 aux probabilités de transition d'un élment absorbant à lui même et attribué les probabilités de transition de i à i+1 de 0.4 et de i+1 à i de 0.6 et 0 dans les autres composantes.

```python
A = np.zeros((nmbr_ttl, nmbr_ttl))

for i in range(9):
    for j in range(9):
        if j == i - 1:
            A[i, j] = perte
        elif j == i + 1:
            # ganger un euro
            A[i, j] = gain
        elif j == i:
        # nulle sue la diagonale à l'exeption des 2 élement
            A[i, j] = 1 - perte - gain
        A[0, 0]=1
        A[8, 8]=1
        A[0, 1]=0
        A[8, 7]=0


print("A=",A)
```

FIGURE 9 – Script python

```
A= [[1.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.6 0.  0.4 0.  0.  0.  0.  0.  0. ]
 [0.  0.6 0.  0.4 0.  0.  0.  0.  0. ]
 [0.  0.  0.6 0.  0.4 0.  0.  0.  0. ]
 [0.  0.  0.  0.6 0.  0.4 0.  0.  0. ]
 [0.  0.  0.  0.  0.6 0.  0.4 0.  0. ]
 [0.  0.  0.  0.  0.  0.6 0.  0.4 0. ]
 [0.  0.  0.  0.  0.  0.  0.6 0.  0.4]
 [0.  0.  0.  0.  0.  0.  0.  0.  1. ]]
```

FIGURE 10 – Résultat : matriceA

→ Pour obtenir numériquement la loi on utilise les fonctions définis dans le premier exercice pour calculer à chaque fois la valeur de $\mu_0.P^n$, avec dans ce cas $\mu_0 = [0,0,1,0,0,0,0,0,0]$ car Morty avait 3 euro en début de jeu et P=A la matrice définie par la figure 9.soit :

```python
import numpy as np

def matrix_power(P, n):
    mu_0 = np.array([0,0,0,1,0,0,0,0,0])
    result = mu_0
    for i in range(n):
        result = np.dot(result, P)
    return result

P = np.array([[1,0,0,0,0,0,0,0,0],[0.6,0,0.4,0,0,0,0,0,0],[0,0.6,0,
n = 5
m=10
i=50
l=100
print(matrix_power(P, n))
print(matrix_power(P, m))
print(matrix_power(P, i))
print(matrix_power(P, l))
```

FIGURE 11 – Script python

```
pour n=3 [0.216 0.    0.432 0.    0.288 0.    0.064 0.    0.   ]
pour n=10 [0.56710195 0.10749542 0.         0.16403005 0.         0.1008599
 0.         0.02618819 0.03432448]
pour n=100 [9.03526902e-01 1.25490981e-05 0.00000000e+00 2.01974685e-05
 0.00000000e+00 1.34649790e-05 0.00000000e+00 3.71825129e-06
 9.64231678e-02]
```

FIGURE 12 – Probabilité μ_n

La probabilié ainsi demandé est $\mu_0 = 0.096 \approx 0.1$

2.4 Question 4

Pour simuler la chaîne on utilise le même principe que le premier exercice en se ramenant à une loi uniforme, soit le script suivant :

```python
def Simulation_uniforme(X):
    U= np.random.rand(1)[0]
    if (X==0):
            return 0
    if (X==1):
        if U < 6/10:
            return 0
        else :
            return 2
    if (X==2):
        if U < 6/10:
            return 1
        else :
            return 3
    if (X==3):
        if U < 6/10:
            return 2
        else :
            return 4
    if (X==4):
        if U < 6/10:
            return 3
        else :
            return 5
    if (X==5):
        if U < 6/10:
            return 4
        else :
            return 6
    if (X==6):
        if U < 6/10:
            return 5
        else :
            return 7
    if (X==7):
        if U < 6/10:
            return 6
        else :
            return 8
    if (X==8):
        return 8
```

FIGURE 13 – Simulation en uniform

Maintenant pour estimer la valeur de probabilité il suffit d'iéterer sur un grand nombre de fois,Cela consiste à générer de nombreux scénarios aléatoires en utilisant la matrice de transition A, c'est le principe de monte carlo.