1. Simulation et calcul matriciel

November 5, 2018

1 Simulation et calcul matriciel

1.1 Préliminaires

Importation de la librairie numpy très utile pour l'algèbre linéaire

```
In [2]: import numpy as np
        from numpy.linalg import matrix_power
```

1.2 Toolbox

```
In [16]: # Créer un vecteur
         vec = np.array([1, 0, 0])
         # Créer une matrice
         matrix = np.array([[1, 0, 0],
                            [0, 1, 0],
                            [0, 0, 1]])
         # Produit matrix vecteur
         np.dot(vec,matrix)
         # Produit d'une matrice
         matrix_power(matrix, 2)
         # Boucle for
         for t in range(0, 3, 1):
             print(t)
         # Boucle while
         t = 0
         while t<2 :
             t = t+1
             print(t)
         #Random vector from a multinomial distribution
         ## nb_exp = nombre d'expérience
         nb_exp = 3
         ## p_vec = vecteur de probabilité
         p_vec = [1/3, 1/3, 1/3]
         ## sample_size = taille de l'échantillon
         sample_size = 3
         print(np.random.multinomial(nb_exp, p_vec, sample_size))
```

```
0
1
2
1
2
[[1 0 2]
 [0 0 3]
 [1 0 2]]
```

Utilisation de la fonction np.array() pour créer des vecteurs et des matrices

```
In [2]: #La loi initial de la cahine de Markov est un vecteur de taille 5
        mu = np.array([0, 1/3, 1/3, 1/3, 0])
        #Fonction print pour visualiser la valeur d'une variable
        print("La loi initiale est donnée par ", mu)
        #La matrice de transition est une matrice
        Q = np.array([[1, 0, 0, 0, 0],
                      [1/2, 1/8, 1/4, 0, 1/8],
                      [1/4, 1/2, 1/8, 1/8, 0],
                      [0, 1/4, 1/2, 1/4, 0],
                      [0, 0, 0, 0, 1]])
        print("La matrice de transition est donnée par ", Q)
```

```
La loi initiale est donnée par  [0.         0.33333333 0.33333333 0.33333333 0.        ]
La matrice de transition est donnée par  [[1.    0.    0.    0.    0.   ]
 [0.5   0.125 0.25  0.    0.125]
 [0.25  0.5   0.125 0.125 0.   ]
 [0.    0.25  0.5   0.25  0.   ]
 [0.    0.    0.    0.    1.   ]]
```

1.3 Question 1

- La multiplication vectoriel/matriciel se fait via la fonction dot().
- On peut évaluer les puissance de matrices via la fonction matrix_power()

```
In [3]: # Extraction de la fonction matrix_power() de la librairie numpy.linalg
        from numpy.linalg import matrix_power
        #Loi de probabilité de X_4
        p_X_4 = mu.dot(matrix_power(Q, 4))
        print(p_X_4)
```

```
[0.71044922 0.08015951 0.06518555 0.02319336 0.12101237]
```

1.4 Question 2

1.4.1 Simulation d'une trajectoire

On va utiliser la fonction np.random.multinomial() pour générer les états de la chaine, on ajoute des éléments à un vecteur via la fonction append(). On itère en utilisant une boucle for.

```
In [4]: #Vecteur espace d'état
        E = np.array([1, 2, 3, 4, 5])
        print(E)

        # Initialisation de la chaine de Markov
        X0 = np.random.multinomial(1, mu, 1).dot(E)
        print(X0)
        # Nous ne voulons pas stocker la valeur dans un vecteur
        print(X0[0])

        # On a besoin des probabilités de transition de l'état 0 vers un autre état => Matrice
        # Première ligne et première colonne
        print(Q[0, 0])
        # Première ligne
        print(Q[0, ])
        #Ligne qui nous intéresse
        print(Q[X0[0]-1, ])


        #L'état suivant
        X1 = np.random.multinomial(1, Q[X0[0]-1, ], 1).dot(E)
        print(X1)

        # La trajectoire avec deux itérations
        X = np.append(X0 , X1)
        print(X)

        #
```

```
[1 2 3 4 5]
[3]
3
1.0
[1. 0. 0. 0. 0.]
[0.25  0.5   0.125 0.125 0.   ]
[2]
[3 2]
```

La trajectoire avec 20 itérations, Utilisation de la boucle for et de la fonction range()

```
In [5]: # Suite des 20 premiers entiers
        print(range(1, 20, 1))
        # de 0 à 1 par pas de 0.1
        print(range(0, 20, 2))

        X = np.random.multinomial(1, mu, 1).dot(E)
        for t in range(0, 20,1):
            X = np.append(X, np.random.multinomial(1, Q[X[t]-1, ], 1).dot(E))

        print(X)
```

```
range(1, 20)
range(0, 20, 2)
[4 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

La trajectoire qui s'arrête à l'absorbtion. On mesure la longueur de la trajectoire grâce à la fonction len().

```
In [8]: X = np.random.multinomial(1, mu, 1).dot(E)
        t=0

        while X[t] != 1 and X[t] != 5 :
            X = np.append(X, np.random.multinomial(1, Q[X[t] - 1, ], 1).dot(E))
            t = t + 1

        print(X)
        # Longueur de la trajectoire = temps avant absorbtion
        print(len(X))
```

```
[3 3 2 3 2 1]
6
```

1.4.2 Calcul de l'espérance du temps d'absorbtion

On réalise 10,000 réplications du temps d'absorption et on calcul la moyenne empirique via la fonction np.mean()

```
In [9]: tauA = np.array([])
        for k in range(0, 10000):
            X = np.random.multinomial(1, mu, 1).dot(E)
            t = 0
            while X[t] != 1 and X[t] != 5 :
                X = np.append(X, np.random.multinomial(1, Q[X[t] - 1, ], 1).dot(E))
                t = t + 1
            tauA = np.append(tauA, len(X))

        print(tauA)
        print(np.mean(tauA))
```

```
[4. 2. 7. ... 3. 4. 2.]
3.8348
```

In [ ]: