UPS TOULOUSE III **MODELISATION**
M1-SID, TP3 **2023-2024**

Chaines de Markov et application au Pagerank

1 Introduction

Le but de ce TP est de vous faire simuler des chaînes de Markov, d'estimer leurs probabilités invariantes et d'appliquer celà au Pagerank de Google.

En fin de séance, votre TP devra être déposé sur le dépot :
https://www.dropbox.com/request/HlOfx3x1dmzmPK71OLgn

IMPORTANT : A des fins de triage, merci de nommer votre fichier NOM_PRENOM.ipynb. Eviter TP3_NOM_PRENOM ou toute autre variante.

2 Simulation de chaines de Markov

- L'espace d'états d'une chaine de Markov est identifié à

$$E = \{0, 1, 2, \ldots, N-1\}.$$

Ecrivez une fonction *markov_step(P, x)* prenant en entrée une matrice de transition P et un $x \in E$. La sortie est la réalisation aléatoire d'un pas de la chaine de Markov.

- Ecrivez également une fonction *markov_steps(n, P, x)* dont la sortie est la liste des réalisations d'une trajectoire de taille $n \in \mathbb{N}$. On pourra invoquer la fonction précédente dans une récurrence.

- Parallélisation : Lorsque x est une liste, faites en sorte que votre code fonctionne et rende une liste de réalisations indépendantes de chaines de Markov. Pour *markov_steps(n, P, x)*, retourner une liste de listes.

- Dessiner une trajectoire du processus de vie ou de mort de matrice de transition :

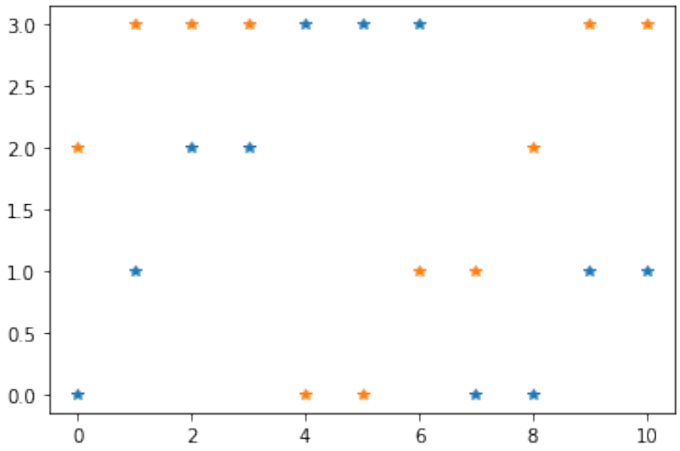
$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

In fine, on pourra executer :

```
import matplotlib.pyplot as plt
trajectory = markov_steps(10, P, [0,2])
print(trajectory)
plt.plot( trajectory, '*' )
```

Et le résultat devra ressembler à :

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



3 Mesure invariantes

Considérez une des matrices de transitions vue en TD. On rappelle le Lemme vu en cours :

Lemme 1. Soit P une matrice de transition irréductible et apériodique. Alors P^n converge vers une matrice dont toutes les lignes sont égales à la mesure invariante.

Estimer la probabilité invariante par la méthode de la puissance.
Ensuite, discutez des avantages et inconvénients des méthodes suivantes :
— la méthode de la puissance.
— la résolution de $\pi P = \pi$.
— la méthode "Monte-Carlo" consistant à simuler des trajectoires et estimer π par le théorème ergodique.

4 Pagerank

4.1 Baby version

Un internaute est susceptible d'aller sur 3 sites :
— le site A contient un lien vers lui-même, un lien vers B et un lien vers C.
— le site B comporte 5 liens vers lui-même, un vers A et un vers C.
— le site C comporte un lien vers A, 7 liens vers B et 4 vers C.
Au départ l'internaute choisit au hasard l'un des trois sites. Par la suite la probabilité de passer d'un site (à l'instant n) vers un autre (à l'instant n + 1) est proportionnelle au nombre de liens au premier site vers le deuxième.

Questions :

- Entrer la matrice de transition dans Python.

- Simuler les déplacements d'un internaute et donner un plot pour $n = 1000$. Quel ranking donneriez-vous à l'oeil nu ? Quel theorème invoquez-vous de façon implicite ?

- Donner rigoureusement le Pagerank des trois sites.

4.2 Version plus réaliste

Le défi du Pagerank consiste à tout de même estimer une mesure invariante pour de très grandes matrices, parfois connues seulement partiellement.

Questions :

1) Une difficulté majeure consiste déjà avoir des graphes qui modélisent de façon réaliste le réseau internet : grand graphe, avec des degrés de sommets très faibles. Tirer une matrice de

transition aléatoirement avec des entrées indépendantes (puis normaliser) est un très mauvais choix. Pourquoi ?

2) Voici un code. Que fait-il ?

```
import numpy as np

# Erdos-Renyi model
def sample_adjacency(degree, n):
    result = np.zeros( (n,n) )
    for i in range(n):
        random_indices = np.random.randint(n, size=degree)
        result[ i, random_indices ] = 1
    return result

# Random permutation model
def sample_adjacency_v2(degree, n):
    result = np.zeros( (n,n) )
    for i in range(degree):
        permutation = np.random.uniform( )
        permutation_matrix = np.zeros( (n,n) )
        for j in range(n):
            permutation_matrix[j,permutation[j]] = 1
        result     = result + permutation_matrix
    return result

def normalize_matrix(matrix):
    result = matrix
    for i in range(0, matrix.shape[0]):
        result[i,:] = result[i,:]/np.sum(result[i,:])
    return result

def sample_transitions(degree, n):
    return normalize_matrix( sample_adjacency(degree, n) )

def sample_transitions_v2(degree, n):
    return normalize_matrix( sample_adjacency_v2(degree, n) )
```

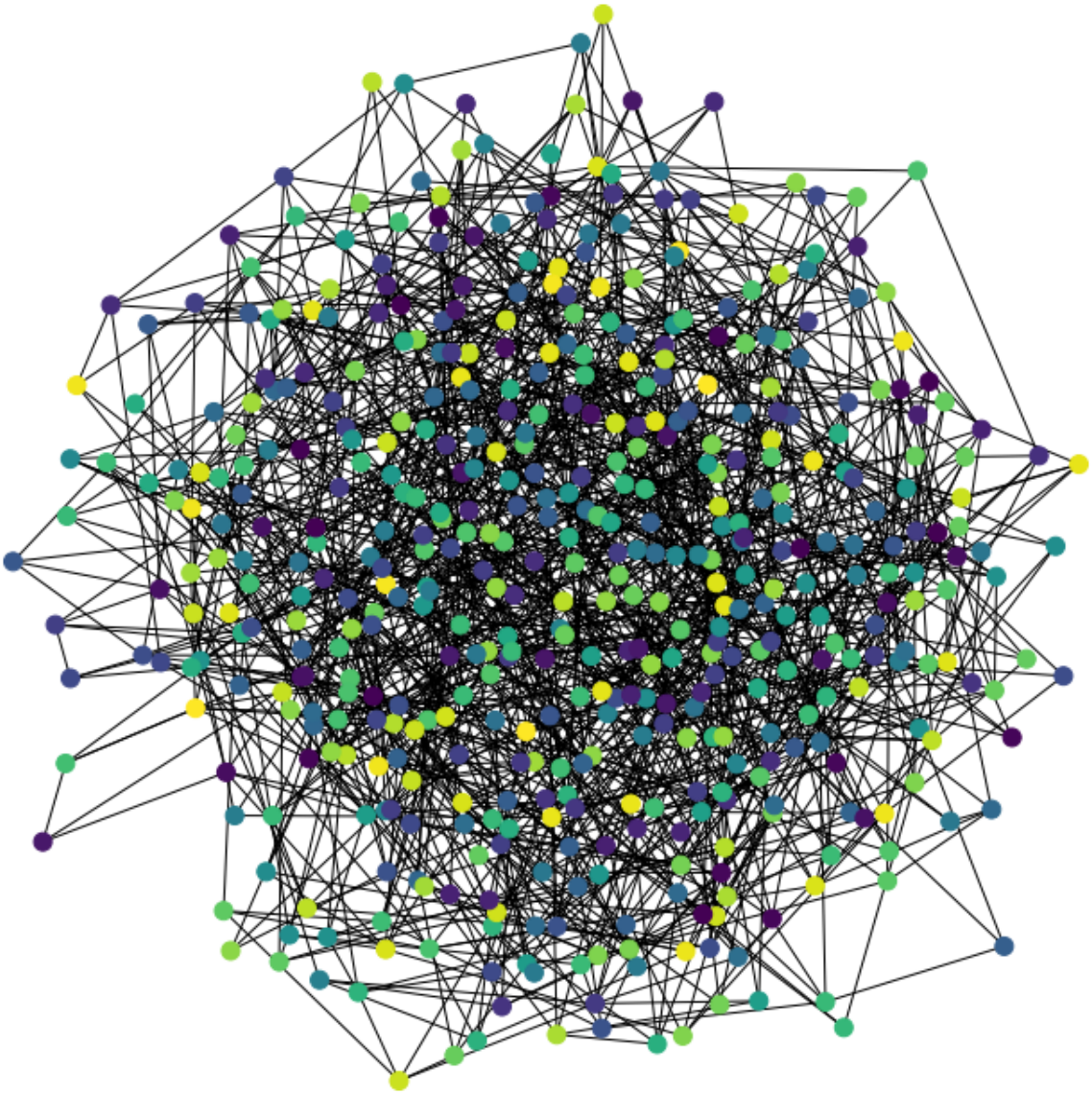
3) Le code suivant produit un dessin du résultat :

```
import matplotlib.pyplot as plt
import networkx as nx

def show_graph(adjacency_matrix, invariant_measure=None):
    m,n = adjacency_matrix.shape
    assert( n == m )
    #
    rows, cols = np.where(adjacency_matrix > 0)
    edges = zip(rows.tolist(), cols.tolist())
    gr = nx.Graph()
    gr.add_edges_from(edges)
    #
    if invariant_measure is None:
        invariant_measure = np.random.rand(n)
    nx.draw(gr, node_size=100, node_color=invariant_measure)
    plt.show()

plt.rcParams['figure.figsize'] = 10, 10
adj_matrix = sample_transitions(3,500)
```

```
show_graph( adj_matrix )
```



Est-ce que cela semble un bon modèle pour le graphe d'internet ? Pour les graphes de connexion dans des réseaux sociaux ? On critiquera l'allure du graphe et sa connectivité.

4) A nouveau, quelle est la meilleure méthode pour estimer la mesure invariante π ? On discutera des cas $n \in \{10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9\}$.

Implémenter les deux seules méthodes à votre disposition.

On devra produire des plots de ce type, pour les "petits" n.