Modélisation du vélib

MPRO

2024-2025

1 Description du problème

On regardera avec profit le chapitre "Open migration processes", page 30 et suivantes, du livre *Lectures on stochastic networks* par F. Kelly et E. Yudovin disponible à ce lien

On considère un système de vélos partagés, type Vélib, où les vélos sont disponibles dans des stations dédiées et peuvent être empruntés pour faire des trajets d'une station à une autre.

On suppose que la capacité des stations est illimitée pour l'accueil des vélos, c'est-à-dire que l'on peut toujours rendre un vélo dans une station. On suppose aussi que tous les trajets se font d'une station à une autre exclusivement, il n'est pas possible de revenir à la même station.

2 Modèle

Pour limiter la complexité calculatoire, on ne considère que le cas où il y 5 stations et N vélos. On vous propose le modèle suivant (que vous n'avez pas le droit de ne pas suivre) basé sur le modèle des colonies : on a 5 colonies qui représentent les stations et 20 colonies qui représentent les routes entre chaque paire de stations. Il y a au total 25 colonies, on note

N_{ii} : le nombre de vélos disponibles dans la colonie

N_{ij} : le nombre de vélos en route de la station i vers la station j

Les demandes de vélos à la station numéro i se font selon un processus de Poisson d'intensité dépendant de la station, exprimée en 1/min dont les valeurs sont données ci-dessous.

[ ]:
```
from numpy import *
import numpy as np
lambda_=np.array([2.8,3.7,5.5,3.5,4.6])/60.
print(lambda_)
```

Les temps de traversée (en minutes) suivent des lois exponentielles dont les paramètres sont données dans la case ci-dessous

[ ]:
```
T=np.array([[0,3.,5.,7.,7.],[2.,0,2.,5.,5.],[4.,2.,0,3.,3.],[8.,6.,4.,0,2.],[7.
 ↪,7.,5.,2.,0]])
print(T)
```

Lors de son départ d'un station, un vélo se dirige vers une autre station selon les probabilités données dans la case ci-dessous

[ ]:
```
R=np.array([[0,0.2,0.3,0.2,0.3],[0.2,0.,0.3,0.2,0.3],[0.2,0.25,0.,0.25,0.3],[0.
↪15,0.2,0.3,0.,0.35],[0.2,0.25,0.35,0.2,0.]])
print(R)
```

3 Simulation en Python

3.1 Représentation de l'espace d'états

Il y a les vélos disponibles en stations que l'on noté N_{ii} et les vélos en cours de transition de la station i vers la station j dont on a noté le nombre par N_{ij}.

** La commande np.random.choice **

Pour se simplifier la vie, on doit, plusieurs fois, faire appel à la commande

`np.random.choice(valeurs,p=proba)`

qui retourne un tirage dans *valeurs* selon le vecteur de probabillités *proba*

Ainsi

`np.random.choice(3,p=[0.2,0.5,0.3])`

renvoie 0 avec probabilité 0.2, renvoie 1 avec probabilité 0.5, etc.

[ ]:
```
for i in np.arange(5):
    print(np.random.choice(3,p=[0.2,0.5,0.3]))
```

Il en découle qu'il est plus simple de représenter un état sous forme d'un vecteur de longueur 25 $E = (E_i, 0 \leq i \leq 24)$, version dépliée de la matrice 5x5 $N = (N_{i,j}, 0 \leq i, j \leq 4)$. Ce qui veut dire que

$$N_{ii} \Longleftrightarrow E[6 * i]$$
$$N_{ij} \Longleftrightarrow E[5 * i + j]$$

4 Fonctions

On fera trois fonctions

```
def transition(E):
    ...
    return Tpos
```

Cette fonction *transition* prend un état E et renvoie le vecteur des transitions possibles Tpos. Une façon de représenter les transitions possibles est la suivante : + Tpos[6*i] vaut le taux de départ de la station i s'il y reste au moins un vélo + Tpos[5*i+j] vaut le taux de transition de N-> $N_{ij} - 1$, $N_j + 1$

```
def saut(E):
    ...
    return tempsSejour,caseADecrementer,caseAIncrementer
```

Cette fonction *saut* fait appel à la précédente et retourne, le temps de séjour dans l'état E, l'indice de l'élément de E qui va être diminué de 1 (noté *depart*) et l'indice de E qui va être augmenté de 1 (noté *arrivee*)

```
def trajectoire(Einitial,horizon)
    ...
    return Tvide/horizon
```

Cette fonction simule une trajectoire partant de l'état *Einitial*, jusqu'au temps *horizon*. Elle retourne le vecteur *Tvide* de longueur 5 qui contient le pourcentage du temps pendant lequel chaque station est vide.

[ ]:
```
def transition(etat):
    ...
    return Tpos

def saut(etat):
    ...
    return -log(np.random.rand())/tau,caseADiminuer,caseAAugmenter

def trajectoire(etat,horizon):
    ...
    return Tvide/horizon
```

4.1 Simulation

- *Einitial* est le vecteur représentant l'état initial
- *Nbiter* le nombre de réalisations
- *Horizon* l'horizon temporel de chaque réalisation

[ ]:
```
def simulation(Einitial,Nbiter,Horizon):
    stationsVides=np.empty(5)
    for k in np.arange(Nbiter):
        stationsVides=np.vstack((stationsVides,trajectoire(Einitial,Horizon)))
    print("Pourcentage du temps où les stations sont vides\n")
    for i in np.arange(5):
        m=1-np.sum(stationsVides[:,i])/Nbiter
        print('Station '+str(i)+' : {:.3f}'.format(m))
        v=1.96*np.std(stationsVides[:,i])/np.sqrt(Nbiter)
        print('Intervalle de confiance : [{:.3f}'.format(m-v)+', {:.3f}'.
↪format(m+v)+']')
        print("-------------\n")
```

4.2 Calcul de la probabilité stationnaire pour 1 vélo

Vous pouver avantageusement regarder le chapitre

Pour valider votre code, rien de tel que de le vérifier sur une situation où l'on peut calculer les quantités intéressantes de façon théorique. Pour 1 vélo, on doit avoir

$$\sum_{i=0}^{4}\sum_{j=0}^{4} N_{i,j} = 1$$

donc l'espace d'états se résume à l'ensemble des vecteurs à 25 composantes où une seule composante est non nulle et vaut 1, il y a donc 25 états !

Vous devez 1. construire la matrice 25x25 de transition du système Vélib dans ce cas, noté P

2. remplacer la dernière colonne de cette matrice par une colonne de 1
3. construire le vecteur de taille 25 : $v = (0, \cdots, 0, 1)$
4. résoudre $P^t x = v$
5. exprimer le pourcentage du temps où la station i est vide en fonction de x
6. comparer ces valeurs théoriques et exactes à celles données par votre simulation. Tant qu'elles ne coïncident pas, i.e. tant que les valeurs théoriques ne sont pas dans l'intervalle de confiance, c'est que vous avez un problème...

Pour indication, vous devez trouver comme probabilité stationnaire

```
array([ 0.17546728,  0.15550435,  0.13452462,  0.15566816,  0.15927423])
```

[ ]:
```
# Calcul des probabilités stationnaires pour 1 vélo
Einitial=np.zeros(25)
Einitial[0]=1
simulation(Einitial,1000,10000.)

# Test de l'accord des valeurs avec Einitial=[1,0,0,...]
```

4.3 Calcul pour les données réelles

Maintenant, il ne vous reste plus qu'à faire tourner votre programme avec 50 vélos et répondre à la même question : quelle est la probabilité qu'une station donnée soit vide à l'état stationnaire ?

4.3.1 Expliquer pourquoi on peut choisir un état de départ quelconque.

[ ]:
```
Einitial=np.zeros(25)
for i in np.arange(100):
    Einitial[np.random.randint(25)]+=1
simulation(Einitial,1000,10000.)
```

[ ]:
```
```