tpmarkov

July 9, 2020

1 Introduction aux Processus Stochastiques (PRSTO)

P. Carmona

Pour avancer dans le notebook et exécuter les cellules il faut taper Shift+Enter ou utiliser la barre d'outils ci-dessus et choisir Cell, Run Cell and select Below

1.1 Consignes

Vous répondrez aux questions en modifiant ce notebook. En insérant des cellules de type Markdown pour le texte et des cellules de type code pour le code.

Ensuite vous sauvez ce notebook sous le nom Prenom_Nom_tpmarkov.ipynb et vous le déposez sur Moodle

1.2 Simulation d'une chaîne de Markov

Le langage Python dispose d'une fonction pour simuler suivant une loi discrète *rnd.choice* On dispose d'une matrice de transition, par exemple celle modélisant le climat à Los Angeles.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd
%matplotlib inline


pla=np.array([[0.5,0.5],[0.1,0.9]])
pla
rnd.choice(a=2,p=pla[0])
```

[1]:
```
0
```

On va simuler une chaîne de Markov de matrice de transition pla et issue de 2.

De façon plus générale, le vecteur x contiendra les n premières valeurs simulées d'une chaîne de Markov issue de 2.

[2]:
```
n=50
x=np.arange(n)
x[0]=1
```

```
dim=len(pla[0])
for i in np.arange(1,n):
    x[i]=rnd.choice(a=dim,p=pla[x[i-1]])

x
```

[2]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1])
```

Ecrire une fonction genmarkov(x,n,p) qui simule une chaîne de markov de matrice de transition p issue de x et qui renvoie la valeur $X_{n-1}(\omega)$. On écrira également la fonction vgenmarkov qui elle renvoie le vecteur des valeurs $(X_0(\omega), ..., X_{n-1}(\omega))$.

Pour le climat à Los Angeles, vérifiez que la proportion de temps passé dans chaque état converge vers un nombre qui ne dépend que de cet état et pas du point de départ.

[3]:
```
n=5000
z=vgenmarkov(1,n,pla)
(z==0) #donne un vecteur de booleen
(z==0).sum() #donne le nombre de fois ou l'on est passe dans l'etat 0
(z==0).mean() #donne la moyenne empirique
(z==0).cumsum() #donne le vecteur des sommes cumulees
y=(z==0).cumsum()/np.arange(1,len(z)+1) #donne le vecteur des moyennes de 0
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-3-da6a850e0e43> in <module>()
  1 n=5000
----> 2 z=vgenmarkov(1,n,pla)
  3 (z==0) #donne un vecteur de booleen
  4 (z==0).sum() #donne le nombre de fois ou l'on est passe dans l'etat 0
  5 (z==0).mean() #donne la moyenne empirique


NameError: name 'vgenmarkov' is not defined
```

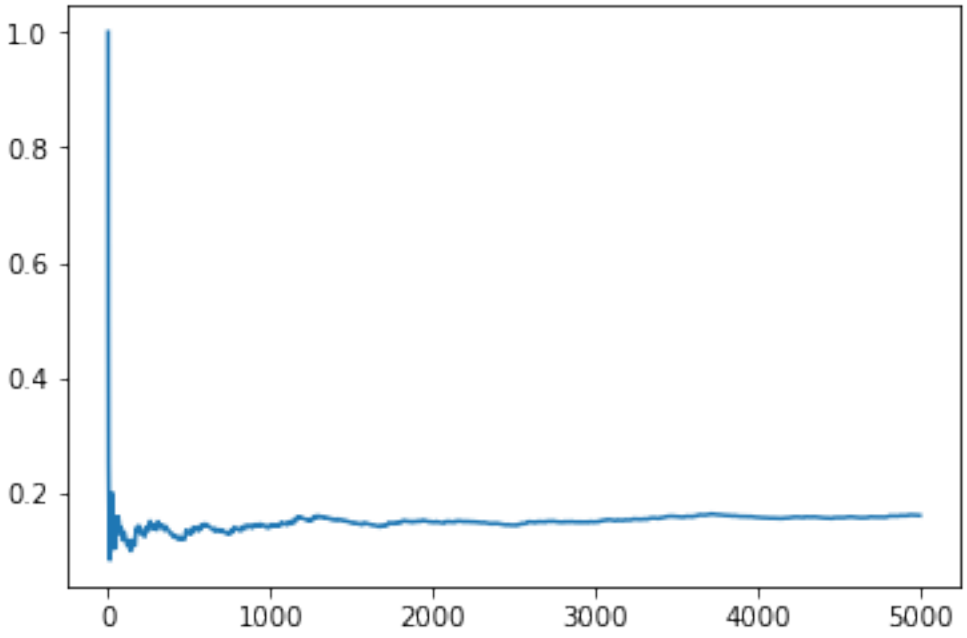
[4]:
```
plt.plot(y)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)
```

```
    <ipython-input-4-9436e2a700a8> in <module>()
----> 1 plt.plot(y)


    NameError: name 'y' is not defined
```

[211]:
```
#on change le point de départ
n=5000
z=vgenmarkov(0,n,pla)
y=(z==0).cumsum()/np.arange(1,len(z)+1) #donne le vecteur des moyennes de 0
plt.plot(y)
```

[211]:
```
[<matplotlib.lines.Line2D at 0x7f6583444c50>]
```



On observe bien la convergence vers 1/6 dans les deux cas.

Ce résultat est-il encore vrai pour les matrices p3 et p4 ci dessous ? Pouvez vous expliquer pourquoi ?

[20]:
```
p3=np.array([[7/20,3/20,1/4,1/4],[3/10,1/4,7/20,1/10],[1/4,1/4,7/20,3/20],[3/
↪10,1/4,1/4,1/5]])
p4=np.array([[1/2,0,0,0,1/2],[0,1/2,0,1/2,0],[0,0,1,0,0],[0,1/4,1/4,1/4,1/4],[1/
↪2,0,0,0,1/2]])
```

1.3 Calcul des probabilités de transition

On utilise également la matrice de transition $p = pt$ suivante

[212]:
```
pt=np.array([[1/2,0,1/2,0],[0,1,0,0],[1/2,0,0,1/2],[0,1/2,0,1/2]])
pt
```

[212]:
```
array([[ 0.5,  0. ,  0.5,  0. ],
       [ 0. ,  1. ,  0. ,  0. ],
       [ 0.5,  0. ,  0. ,  0.5],
       [ 0. ,  0.5,  0. ,  0.5]])
```

Utiliser la loi forte des grands nombres pour calculer, pour les matrices $p3, p4$ des valeurs approchées de $p_{2,3}^{(n)}, p_{3,3}^{(n)}, p_{1,4}^{(n)}$ pour les valeurs de n suivantes $n = 2, 5, 10, 20$. Par exemple, vous faites N=1000 (ou N=10000, ou N=50000) simulations de la chaîne de Markov issue de 2, puis vous comptez la proportion de fois où vous observez la valeur 3

Vérifier vos simulations en utilisant le calcul matriciel. Sous Python, le produit matriciel des matrices p et q s'écrit np.dot(p,q) ce qui est très différent de $p * q$. Ecrivez une fonction puis(p,n) qui renvoie la matrice p^n. (Vous pouvez utiliser un algorithme de calcul rapide de puissance si vous savez le coder).

1.4 Calcul de probabilités et de temps d'absorption

Premièrement, en utilisant la structure de classes, décrire les 3 comportements asymptotiques possibles pour la chaîne de Markov de matrice de transition

[213]:
```
p=np.array([[1,0,0,0,0],[0,1,0,0,0],[3/20,1/10,3/10,1/5,1/4],[1/5,3/20,3/20,7/
↪20,3/20],[0,0,0,0,1]])
p
```

[213]:
```
array([[ 1.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  1.  ,  0.  ,  0.  ,  0.  ],
       [ 0.15,  0.1 ,  0.3 ,  0.2 ,  0.25],
       [ 0.2 ,  0.15,  0.15,  0.35,  0.15],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  1.  ]])
```

En utilisant la loi forte des grands nombres, calculer une valeur approchée de la probabilité que la chaîne de Markov, issue de 4, réussisse à atteindre l'état 2.

Essayons maintenant de répondre à la même question en utilisant le calcul matriciel. Ecrire la matrice de la chaîne absorbee $\tilde{P}$ sous la forme canonique

$$\tilde{P} = \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix}$$

Puis utiliser la fonction np.linalg.inv pour calculer $N = (I - Q)^{-1}$ et enfin retrouver le calcul précédent de la probabilité d'absorption en 2 lorsque l'on part de 4.

1.5 Détermination de probabilités invariantes

On peut chercher les valeurs et vecteurs propres de la matrice pla

[214]:
```
np.linalg.eig(pla)
```

[214]:
```
(array([ 0.4,  1. ]), array([[-0.98058068, -0.70710678],
        [ 0.19611614, -0.70710678]]))
```

N'oublions pas que le vecteur **1** dont toutes les coordonnées sont des 1 est un vecteur propre avec la valeur propre 1. Ici nous cherchons une probabilité invariante, i.e. un vecteur propre à gauche. Il faut donc déterminer le spectre de la transposée

[215]:
```
v,w=np.linalg.eig(pla.T)
```

[216]:
```
v
```

[216]:
```
array([ 0.4,  1. ])
```

[217]:
```
w
```

[217]:
```
array([[-0.70710678, -0.19611614],
       [ 0.70710678, -0.98058068]])
```

Il faut faire attention lorsque l'on compare des valeurs numériques

[218]:
```
np.sqrt(2)*np.sqrt(2) == 2.0
```

[218]:
```
False
```

Pour comparer, à la précision de la machine utilisée,

[219]:
```
np.isclose(np.sqrt(2)*np.sqrt(2), 2.0)
```

[219]:
```
True
```

Ecrire une fonction probinv(p) qui prenne en paramètre une matrice stochastique p et qui retourne une probabilité invariante pour p. N'oubliez pas de normaliser le vecteur propre pour obtenir une probabilité.

Tester cette fonction avec *pla* et la matrices de transition *pt* et celle de l'exercice 4.10.

1.6 Le théorème ergodique

Rappelez les conditions d'application de ce théorème. Indiquez pour chacun des exercices 4.7, 4.12 si vous avez le droit de l'appliquer et faites le, éventuellement, pour répondre aux questions de ces exercices.

Vous devez déterminer la/les probabilités invariantes, la fonction f dont vous prenez la moyenne temporelle (c'est un vecteur de la bonne dimension) et calculer la moyenne $< f, \pi >$.

[220]:
```
p7=np.array([[2/3,0,1/3],[1/3,1/3,1/3],[0,1,0]])
p7
```

[220]:
```
array([[ 0.66666667,  0.        ,  0.33333333],
       [ 0.33333333,  0.33333333,  0.33333333],
       [ 0.        ,  1.        ,  0.        ]])
```

1.7 Le théorème de convergence vers l'équilibre

Relire attentivement l'énoncé de ce théorème.

Indiquer, pour la matrice *pla* et celle de l'exercice 4.17, vers quelle matrice P^∞ doit converger P^n quand $n \to \infty$.

Vérifier, dans ces deux cas, que la décroissance d'une distance entre P^n et P^∞ est en ρ^n avec $\rho \in]0,1[$. Comparer la valeur de ρ trouvée expérimentalement et la valeur théorique donnée par le théorème de Perron Frobenius.

[221]:
```
def distance(p,q):
    r=(p-q)**2
    return(np.sqrt(r.sum()))
distance(p7,p7)
```

[221]:
```
0.0
```

1.8 Page Rank

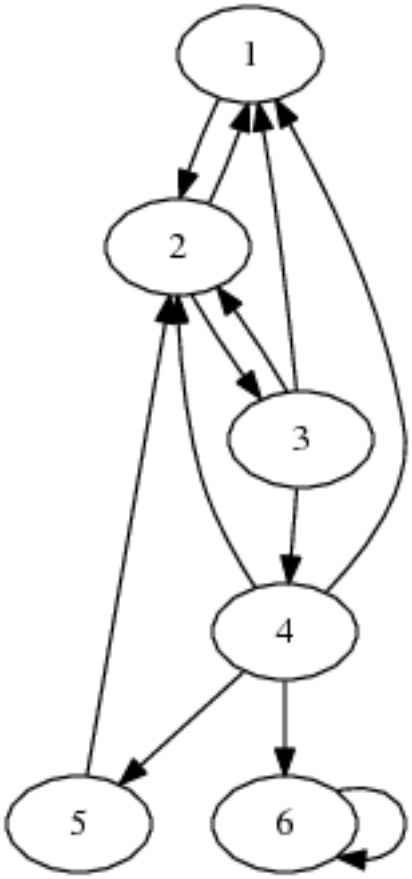
Considérons 6 pages web dont les liens sont donnés par le graphe suivant. Simuler un marcheur qui choisit une page uniformément parmi les liens possibles. Que se passe-t-il ? Pouviez vous le prévoir ?

On rajoute du damping $\alpha \in (0,1)$ ie de l'exploration $1-\alpha$: a chaque étape le marcheur en x choisit un des d_x liens possibles avec la probabilité $\frac{\alpha}{d_x}$ et choisit une page au hasard dans tout le web avec la probabilité $1-\alpha$. Pourquoi y a-t-il une unique mesure invariante ? Trouver la mesure invariante pour $\alpha = 0.5$ et $\alpha = 0.85$. En déduire la page la plus importante dans les deux cas. Commentez.

Commandes Python : np.full

[222]:
```
from IPython.display import Image

fig = Image(filename=('./graphepr.png'))
fig
```

[222]:



[1]:

```
Object `rnd.binom` not found.
```

[ ]: