13.1. Simulation d'une chaîne de Markov à temps discret

Ceci est l'une des 100+ recettes gratuites de la Livre de recettes IPython, deuxième édition , par Cyrille Rossant , un guide de l'informatique numérique et de la science des données dans le cahier Jupyter. L'ebook et le livre imprimé sont

disponible à l'achat chez Packt Publishing .

▸ *Texte sur GitHub avec un Licence CC-BY-NC-ND*
▸ *Code sur GitHub avec un Licence MIT*

▸ ***Aller à** Chapitre 13: Systèmes dynamiques stochastiques*
▸ ***Avoir** le cahier Jupyter*

Les chaînes de Markov à temps discret sont des processus stochastiques qui subissent des transitions d'un état à un autre dans un espace d'état. Les transitions se produisent à chaque pas de temps. Les chaînes de Markov se caractérisent par leur manque de mémoire en ce que la probabilité de subir une transition de l'état actuel au suivant ne dépend que de l'état actuel, pas des précédents. Ces modèles sont largement utilisés dans les applications scientifiques et d'ingénierie.

Des processus de Markov en temps continu existent également et nous couvrirons des cas particuliers plus loin dans ce chapitre.

Les chaînes de Markov sont relativement faciles à étudier mathématiquement et à simuler numériquement. Dans cette recette, nous simulerons une chaîne de Markov simple modélisant l'évolution d'une population.

Comment faire...

1. Importons NumPy et matplotlib:

```
import numpy as np
importer matplotlib.pyplot en tant que plt% matplotlib inline
```

2. Nous considérons une population qui ne peut comprendre plus de $N = 100$

individus, et définir les taux de natalité et de mortalité:

```
N = 100 # taille maximale de la population a = .5 / N # taux de
natalité b = .5 / N # taux de mortalité
```

3. Nous simulons une chaîne de Markov sur l'espace fini $0,1, ..., N$

. **Chaque État représente une taille de population. le x le vecteur contiendra la taille de la population à chaque pas de temps. Nous définissons l'état initial sur** $X_0 = 25$

(c'est-à-dire qu'il y a 25 individus dans la population au moment de l'initialisation):

```
nsteps = 1000
x = np.zeros (nsteps) x [0] = 25
```

4. Maintenant, nous simulons notre chaîne. À chaque pas de temps t

, il y a une nouvelle naissance avec probabilité $hache_t$ **et indépendamment, il y a une nouvelle mort avec probabilité** bx_t

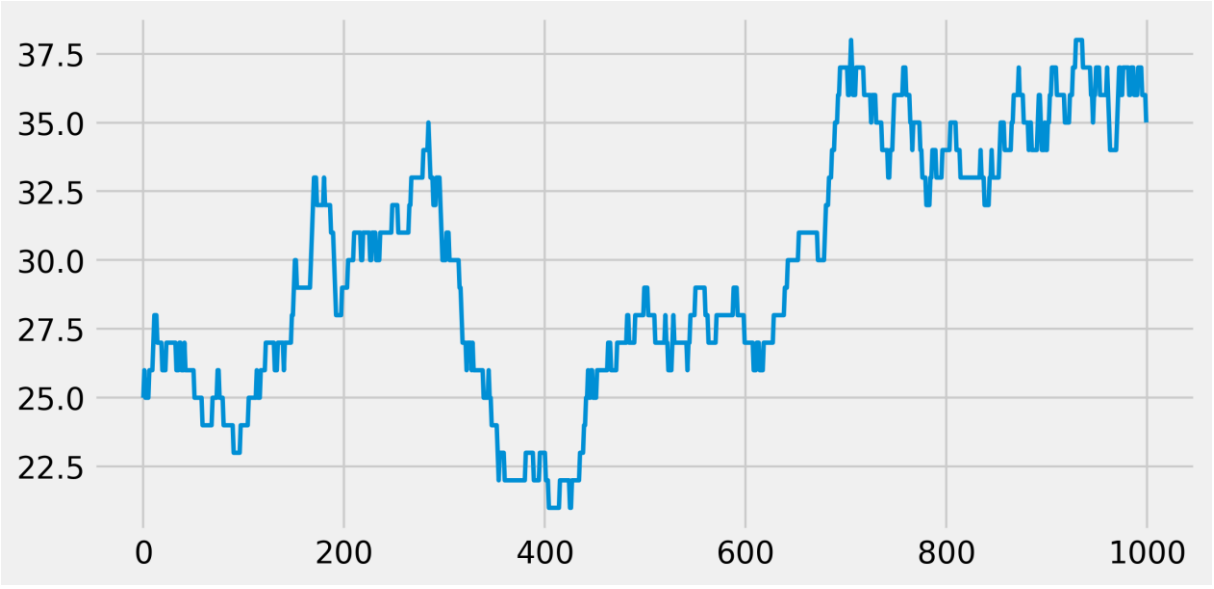
. Ces probabilités sont proportionnelles à la taille de la population à ce moment-là. Si la taille de la population atteint 0 ou N, l'évolution s'arrête:

```
pour t dans la plage (nsteps - 1):
    si 0 <x [t] <N - 1:
        #  Y a-t-il une naissance?
        birth = np.random.rand () <= a * x [t]
        #  Y a-t-il une mort?
        mort = np.random.rand () <= b * x [t]
        #  Nous mettons à jour la taille de la population. x [t + 1] = x [t] + 1 *
        naissance - 1 * décès
    #  L'évolution s'arrête si nous atteignons $ 0 $ ou $ N $. autre:

        x [t + 1] = x [t]
```

5. Regardons l'évolution de la taille de la population:

```
fig, ax = plt.subplots (1, 1, figsize = (8, 4)) ax.plot (x, lw = 2)
```



Nous constatons qu'à chaque pas de temps, la taille de la population peut rester stable, augmenter ou diminuer de 1.

6. Maintenant, nous allons simuler de nombreux essais indépendants de cette chaîne de Markov. Nous pourrions exécuter la simulation précédente avec une boucle, mais ce serait très lent (deux imbriqués pour boucles). Au lieu de cela, nous *vectoriser* la simulation en considérant tous les essais indépendants à la fois. Il y a une seule boucle dans le temps. À chaque pas de temps, nous mettons à jour tous les essais simultanément avec des opérations vectorisées sur des vecteurs. le x Le vecteur contient désormais la taille de la population de tous les essais, à un moment donné. Au moment de l'initialisation, les tailles de population sont définies sur des nombres aléatoires compris entre 0 et N:

```
ntrials = 100
x = np.random.randint (size = ntrials,
                                        bas = 0, haut = N)
```

7. Nous définissons une fonction qui effectue la simulation. À chaque pas de temps, nous trouvons les essais qui subissent des naissances et des décès en générant des vecteurs aléatoires, et nous mettons à jour les tailles de population avec des opérations vectorielles:

```
simulation de déf (x, nétapes):
    "" "Exécutez la simulation." "" Pour _ dans la plage
    (nsteps - 1):
        #  Quels essais mettre à jour? upd = (0 <x) & (x <N
        - 1)
        #  Dans quelles épreuves les naissances ont-elles lieu? naissance = 1 *
        (np.random.rand (ntrials) <= a * x)
        #  Dans quels procès les décès surviennent-ils? mort = 1 * (np.random.rand (ntrials) <=
        b * x)
        #  Nous mettons à jour la taille de la population pour tous les essais x [upd] + =
        naissance [upd] - décès [upd]
```

8. Maintenant, regardons les histogrammes de la taille de la population à différents moments. Ces histogrammes représentent la distribution de probabilité de la chaîne de Markov, estimée avec des essais indépendants (méthode Monte Carlo):

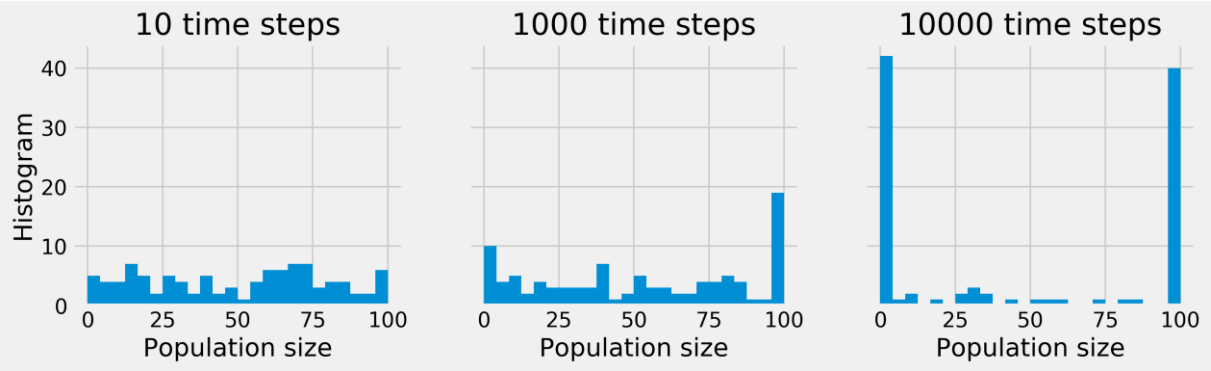
```
bins = np.linspace (0, N, 25) nsteps_list = [10, 1000, 10000]

fig, axes = plt.subplots (1, len (nsteps_list),
                                            figsize = (12, 3), sharey = True)

pour i, nsteps en énumérer (nsteps_list):
    ax = axes [i] simuler (x, nsteps) ax.hist (x,
    bins = bins)

    ax.set_xlabel ("Taille de la population") si i == 0:

        ax.set_ylabel ("Histogramme") ax.set_title (f "{nsteps} pas de
    temps")
```



Alors qu'au départ, la taille des populations semble uniformément répartie entre 0 et N

, ils semblent converger vers 0 ou N après un temps suffisamment long. En effet, les états 0 et N

sont **absorbant;** une fois atteinte, la chaîne ne peut quitter ces états. De plus, ces états sont accessibles depuis n'importe quel autre état.

Comment ça fonctionne...

Mathématiquement, une chaîne de Markov à temps discret sur un espace E

est une séquence de variables aléatoires $X_1, X_2, \ldots$

qui satisfont la propriété Markov:

$$\forall\, n \geq 1,\ P(X_{n+1} \mid X_1, X_2, \ldots, X_n) = P(X_{n+1} \mid X_n)$$

Une chaîne de Markov (stationnaire) est caractérisée par la probabilité de transitions $P(X_j \mid X_{je})$

. Ces valeurs forment une matrice appelée **matrice de transition.** Cette matrice est la matrice d'adjacence d'un graphe orienté appelé **diagramme d'état.** Chaque nœud est un état et le nœud je est connecté au nœud j

si la chaîne a une probabilité de transition non nulle entre ces nœuds.

Il y a plus...

La simulation d'une seule chaîne de Markov en Python n'est pas particulièrement efficace car nous avons besoin d'un pour boucle. Cependant, la simulation de nombreuses chaînes indépendantes suivant le même processus peut être rendue efficace avec la vectorisation et la parallélisation (toutes les tâches sont indépendantes, donc le problème est **embarrassamment parallèle).** Ceci est utile lorsque nous nous intéressons aux propriétés statistiques de la chaîne (exemple de la méthode de Monte Carlo).

Il existe une vaste littérature sur les chaînes de Markov. De nombreux résultats théoriques peuvent être établis avec l'algèbre linéaire et la théorie des probabilités.

Il existe de nombreuses généralisations de chaînes de Markov à temps discret. Les chaînes de Markov peuvent être définies sur des espaces d'états infinis, ou avec un temps continu. De plus, la propriété Markov est importante dans une large classe de processus stochastiques.

Voici quelques références:

- Chaînes de Markov sur Wikipédia, disponibles sur https://en.wikipedia.org/wiki/Markov_chain
- Absorbing Markov chains on Wikipedia, available at https://en.wikipedia.org/wiki/Absorbing_Markov_chain
- Les méthodes de Monte-Carlo sur Wikipédia, disponibles sur https://en.wikipedia.org/wiki/Monte_Carlo_method