

6. Arbres aléatoires

On s'intéresse à un modèle d'arbres aléatoires qui a une structure markovienne, et qui joue un rôle très important à la fois dans la théorie des graphes aléatoires, et pour les applications en biologie. Expliquons d'abord comment encoder et dessiner des arbres dans **Python**. Un *arbre enraciné* est un graphe $T = (V, E)$ connexe, avec un sommet distingué $r \in V$ (la *racine* de l'arbre), et tel que pour tout autre sommet $v \in V$, il existe un unique chemin de longueur minimale reliant r à v . Cette condition implique que T n'a pas de *cycle*, et que $|V| = |E| + 1$. La *profondeur* d'un sommet v de l'arbre est la distance de graphe entre r et v . Le *profil* de l'arbre est le vecteur

$$\begin{aligned} p(T) &= (p_0, p_1, \dots) \\ &= (\text{nombre de sommets de profondeur } 0, \text{nombre de sommets de profondeur } 1, \dots). \end{aligned}$$

Dans ce qui suit, on travaillera avec des arbres enracinés dont les sommets de profondeur d sont étiquetés par les paires d'entiers (d, k) avec $k \in \llbracket 1, p_d \rrbracket$.

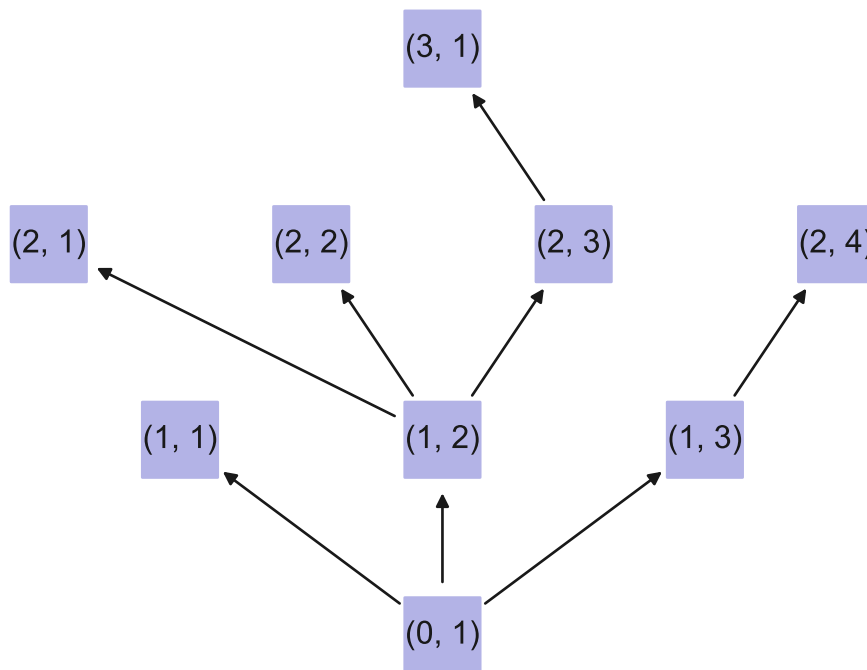


FIG. 6.1. Un arbre enraciné avec 9 sommets.

Le programme suivant, que l'on pourra télécharger, définit une classe `RootedTree` conforme aux définitions données ci-dessus.

```
import numpy as np
import numpy.random as random
import scipy.stats as scs
import matplotlib.pyplot as plt
import networkx as nx
```

```

class RootedTree:
    def __init__(self, max_size=10**6):
        self.depth = - np.ones(max_size, dtype=np.int64)
        self.row_position = np.zeros(max_size, dtype=np.int64)
        self.parent = np.empty(max_size, dtype=object)
        self.children = np.empty(max_size, dtype=object)
        self.depth[0], self.row_position[0], self.children[0] = 0, 1, []
        self.limit, self.profile, self.nodes_number = max_size, [1], 1

    def __repr__(self):
        return f"Rooted tree with {self.nodes_number} vertices"

    def find_index(self, d, k):
        I = (self.depth==d)[:self.nodes_number]
        I *= (self.row_position==k)[:self.nodes_number]
        return int(np.nonzero(I)[0][0])

    def add_children(self, parent, c):
        ip = self.find_index(*parent)
        if len(self.profile) - 1 == self.depth[ip]:
            self.profile.append(0)
        for i in range(self.nodes_number, self.nodes_number + c):
            self.depth[i] = self.depth[ip] + 1
            self.profile[self.depth[i]] += 1
            self.row_position[i] = self.profile[self.depth[i]]
            self.parent[i], self.children[i] = ip, []
        self.children[ip] += list(range(self.nodes_number, self.nodes_number + c))
        self.nodes_number += c

    def networkx(self):
        T = nx.DiGraph({i:self.children[i] for i in range(self.nodes_number)})
        for i in range(self.nodes_number):
            T.nodes[i]["label"] = (int(self.depth[i]), int(self.row_position[i]))
        return T

    def layout(self):
        return {i:np.array([- (self.profile[self.depth[i]] + 1)/2 + self.row_position[i],
            self.depth[i])) for i in range(self.nodes_number)}

    def draw_on_ax(self, ax0, with_labels=False):
        ax0.set_axis_off()
        T = self.networkx()
        if with_labels:
            nx.draw_networkx(T, ax=ax0, pos=self.layout(), node_shape="s",
                node_size=800, node_color=np.array([[0.7,0.7,0.9]]),
                labels={i:T.nodes[i]["label"] for i in range(self.nodes_number)})
        else:
            nx.draw_networkx(T, ax=ax0, pos=self.layout(), node_shape="s",
                node_size=300, node_color=np.array([[0.7,0.7,0.9]]),
                with_labels=False)

```

Détaillons le contenu de ce programme :

- La commande `RootedTree()` crée un arbre avec une racine d'étiquette $(0, 1)$, à laquelle on pourra récursivement greffer de nouveaux sommets. Il stocke les informations dans un tableau avec quelques lignes et un nombre de colonnes indiqué par l'argument `max_size`, qui par défaut prend la valeur 10^6 . Si l'on souhaite manipuler des arbres plus grands, il faudra donc le préciser à l'avance.

- Si T est un arbre, on ajoute c enfants au sommet d'étiquette (d, k) avec la commande `T.add_children((d,k), c)`. Ainsi, l'arbre à 9 sommets ci-dessus est obtenu avec :

```
T = RootedTree()
T.add_children((0, 1), 3)
T.add_children((1, 2), 3)
T.add_children((2, 3), 1)
T.add_children((1, 3), 1)
```

- Les commandes `T.nodes_number` et `T.profile` donnent le nombre de sommets et le profil de l'arbre.
- La méthode `networkx` fait le lien avec le module `NetworkX`, et la méthode `layout` calcule les positions des sommets de l'arbre pour un dessin. On utilisera simplement `T.draw_on_ax(ax0)` pour dessiner l'arbre sur une sous-figure `ax0` d'une figure `Matplotlib`; les étiquettes pourront être affichées avec `with_labels=True`. Ainsi, la figure ci-dessus est obtenue avec :

```
fig, ax0 = plt.subplots()
T.draw_on_ax(ax0, with_labels=True)
plt.show()
```

1. Modèle de Galton–Watson.

Soit X une variable aléatoire à valeurs dans l'ensemble des entiers $\mathbb{N}$. On suppose X intégrable, et on note $m = \mathbb{E}[X]$. Étant donnée une famille $(X_{n,k})_{n \geq 0, k \geq 1}$ de variables indépendantes et toutes de même loi que X , l'*arbre de Galton–Watson* associé à la variable X est l'arbre enraciné aléatoire dont le k -ième sommet de profondeur n a $X_{n,k}$ enfants. Par exemple, l'arbre dessiné précédemment était une réalisation d'un arbre de Galton–Watson avec :

$$\begin{aligned} X_{0,1} &= 3 & ; \\ X_{1,1} &= 0 & ; \quad X_{1,2} = 3 & ; \quad X_{1,3} = 1 & ; \\ X_{2,1} &= 0 & ; \quad X_{2,2} = 0 & ; \quad X_{2,3} = 1 & ; \quad X_{2,4} = 0 & ; \\ X_{3,1} &= 0. \end{aligned}$$

En pratique, un arbre de Galton–Watson peut être infini : il est possible qu'à chaque génération n , le nombre Z_n d'individus de profondeur n soit non nul, et que l'une des variables $X_{n,k}$ avec $k \in \llbracket 1, Z_n \rrbracket$ soit non nulle, de sorte que Z_{n+1} soit encore non nul. Pour rester dans le cadre des arbres enracinés finis, on appellera donc *arbre de Galton–Watson associé à la variable X et coupé à la n -ième génération* le sous-arbre de l'arbre de Galton–Watson obtenu en ne conservant que les sommets de profondeur inférieure ou égale à n .

- (1) Écrire un programme `GaltonWatson(n, X)` qui prend en argument une profondeur n et une variable aléatoire X (définie dans `scipy.stats` et qu'on peut simuler avec `X.rvs()`), et qui renvoie une simulation de l'arbre de Galton–Watson associé à la variable X et coupé à la n -ième génération. Dessiner plusieurs exemples, avec $n = 10$, $X \sim \text{Poi}(\lambda)$ et $\lambda \in \{0.75, 1, 1.25, 1.5\}$. Commenter ces dessins. L'arbre s'éteint-il avant la n -ième génération ? S'il ne s'éteint pas, quel est son comportement ?
- (2) On note Z_n le nombre d'individus de n -ième génération ; le profil de l'arbre obtenu par la commande `GaltonWatson(n, X)` est donc $(Z_0 = 1, Z_1, \dots, Z_n)$. Montrer que

Z_n vérifie la relation de récurrence :

$$Z_{n+1} = \sum_{k=1}^{Z_n} X_{n,k}.$$

En déduire un programme `GWZ(n, X)` qui calcule la suite $(Z_0, Z_1, \dots, Z_n)$ sans s'appuyer sur l'arbre correspondant.

- (3) On note $P(k, l) = \mathbb{P}[X_1 + \dots + X_k = l]$, où les X_i sont des copies indépendantes de la variable X . Montrer que la suite $(Z_n)_{n \in \mathbb{N}}$ de la question précédente est une chaîne de Markov sur $\mathbb{N}$ de matrice de transition P . Si $X \sim \text{Poi}(\lambda)$, que vaut $P(k, l)$? En déduire un programme encore plus simple `GWZ_poisson(n, L)` qui calcule une simulation de $(Z_0, Z_1, \dots, Z_n)$ lorsque la variable de reproduction X suit une loi de Poisson de paramètre L .

2. Probabilité d'extinction.

Étant donné un arbre de Galton–Watson de variable de reproduction X , on note $p_n = \mathbb{P}[Z_n = 0]$ la probabilité pour que l'arbre s'éteigne avant la n -ième génération. On note par ailleurs

$$G(s) = \mathbb{E}[s^X] = \sum_{k=0}^{\infty} \mathbb{P}[X = k] s^k$$

la fonction génératrice de la variable X . Cette fonction est bien définie au moins pour $s \in [0, 1]$.

- (1) Calculer $G(s)$ lorsque $X \sim \text{Poi}(\lambda)$.
- (2) On note $G_n(s) = \mathbb{E}[s^{Z_n}] = \sum_{k=0}^{\infty} \mathbb{P}[Z_n = k] s^k$. Que valent G_0 et G_1 ?
- (3) En décomposant l'espérance

$$\mathbb{E}[s^{Z_{n+1}}] = \sum_{k=0}^{\infty} \mathbb{P}[Z_n = k] \mathbb{E}\left[s^{\sum_{j=1}^k X_{n,j}}\right],$$

écrire une relation de récurrence vérifiée par les fonctions G_n .

- (4) Quel est le lien entre la fonction G_n et la probabilité p_n ? En déduire que :

$$p_n = \underbrace{(G \circ G \circ \dots \circ G)}_{n \text{ compositions}}(0).$$

- (5) Pour $n = 5$, $X \sim \text{Poi}(\lambda)$ et $\lambda \in \{0.75, 1, 1.25\}$, calculer p_n , et comparer ces valeurs aux probabilités empiriques d'extinction d'arbres de Galton–Watson avec ces paramètres (on prendra par exemple des échantillons de taille 1000).
- (6) La *probabilité d'extinction* d'un arbre de Galton–Watson associé à la variable aléatoire X est $p = \lim_{n \rightarrow \infty} p_n$. Montrer que :

$$(m > 1) \Leftrightarrow (p < 1).$$

On pourra remarquer que la fonction G est croissante convexe, et que $m = G'(1)$. Écrire un programme `proba_extinction(L)` qui calcule (une approximation de) p lorsque X suit une loi de Poisson de paramètre L . On pourra agrémenter ce programme d'une représentation graphique du calcul.

- (7) Si $X \sim \text{Poi}(1)$, montrer par une simulation que $\lim_{n \rightarrow \infty} n(1 - p_n) = 2$. Question bonus : démontrer ce résultat.

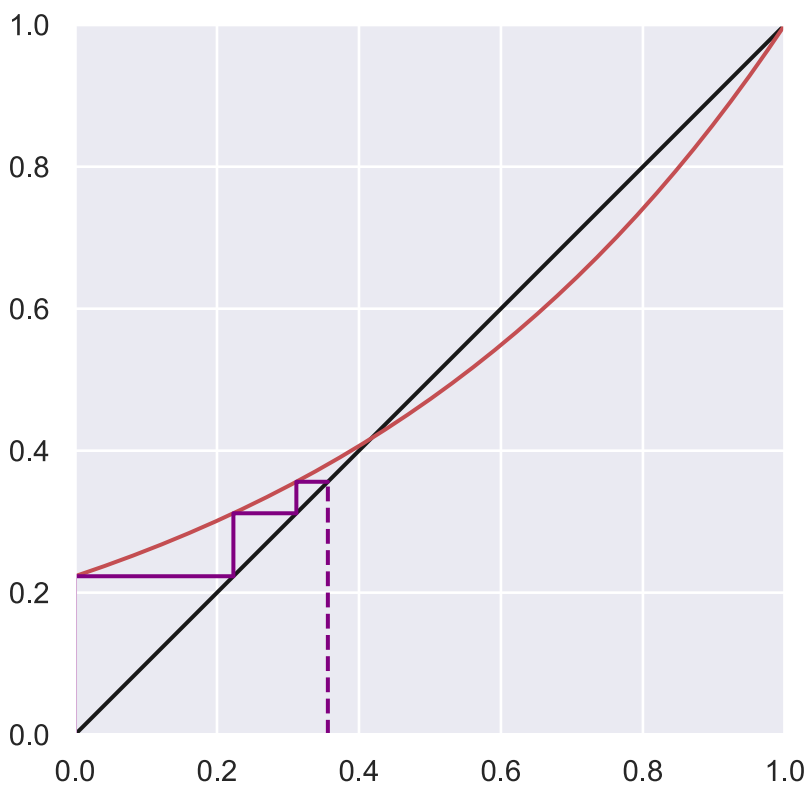


FIG. 6.2. La probabilité d'extinction au temps n est obtenue en itérant n fois la fonction génératrice de X en 0. Ici, $X \sim \text{Poi}(1.5)$ et $n = 3$; $p_3 = 0.3562$.

On peut montrer que si un arbre de Galton-Watson ne s'éteint pas, alors $\lim_{n \rightarrow \infty} Z_n = +\infty$ (sauf dans le cas exceptionnel où $X = 1$ presque sûrement). Autrement dit, on a soit extinction, soit croissance de l'arbre avec un profil de plus en plus large.

$$\mathbb{P}\left[\lim_{n \rightarrow \infty} Z_n = 0\right] + \mathbb{P}\left[\lim_{n \rightarrow \infty} Z_n = +\infty\right] = 1.$$

3. Arbres surcritiques et critiques.

Un arbre de Galton-Watson est dit *surcritique* (respectivement, *critique*) si $m > 1$ (respectivement, si $m = 1$).

- (1) Montrer que $\mathbb{E}[Z_n] = (G_n)'(1)$. En utilisant la relation de récurrence démontrée dans la section précédente, en déduire la valeur de $\mathbb{E}[Z_n]$ en fonction de n et de m .
- (2) On pose

$$M_n = \frac{Z_n}{m^n}.$$

Si $X \sim \text{Poi}(\lambda)$ avec $\lambda \in \{0.75, 1, 1.25, 1.5\}$, dessiner sur l'intervalle de temps $\llbracket 0, N \rrbracket$ des trajectoires de $(M_n)_{n \in \mathbb{N}}$, avec $N = 100$. Illustrer le résultat suivant : lorsque n tend vers l'infini, $(M_n)_{n \in \mathbb{N}}$ converge presque sûrement vers une variable aléatoire M .

- (3) Avec $\lambda = 1.25$ et $n = 100$, dessiner la fonction de répartition empirique de M_n , qui approxime celle de la limite M . Comparer avec la valeur de p calculée dans la section précédente, et la valeur de la fonction de répartition de M en 0. Commenter.

- (4) D'après la dernière question de la section précédente, si $\lambda = 1$, alors $Z_n > 0$ avec probabilité d'ordre $\frac{2}{n}$. On note Z'_n une variable aléatoire de loi :

$$\mathbb{P}[Z'_n = k] = \begin{cases} \frac{\mathbb{P}[Z_n = k]}{\mathbb{P}[Z_n > 0]} & \text{si } k > 0, \\ 0 & \text{si } k = 0. \end{cases}$$

La variable Z'_n représente donc la taille de la n -ième génération d'un arbre de Galton-Watson critique conditionnellement à la non-extinction au temps n . Comme $\mathbb{P}[Z_n > 0]$ ne décroît pas trop vite, on peut simuler Z'_n en tirant au hasard des variables Z_n jusqu'à obtenir un résultat strictement positif. Dessiner la fonction de répartition empirique de la variable $X_n = \frac{Z'_n}{n}$, par exemple avec $n = 100$. Comparer avec la fonction de répartition d'une variable de loi $\text{Exp}(2)$. Que peut-on conjecturer ?