Copule et chaîne de Markov

Contents

Dans cette feuille, on s'intéresse à la simulation de vecteurs aléatoires.

- Dans la première partie on simule un vecteur $(\tau_1, \dots, \tau_n)$ dont la dépendance entre les composantes est donnée par une fonction copule, on parle de dépendance spatiale.
- Dans la deuxième partie, on simule un vecteur $(X_0, \dots, X_n)$ dont la dépendance est temporelle, c'est à dire qu'on construit X_{k+1} à partir de X_k et d'un aléa indépendant. C'est un exemple de chaîne de Markov.

2.1. Copule Gaussienne et instants de défaut

On considère la modélisation suivante: soit $\tau_1, \dots, \tau_n$ des variables aléatoires exponentielles de paramètre $\lambda_i > 0$ qui représentent n instants de défaut (instant de défaillance d'un composant en fiabilité, instant de défaut d'une entreprise en finance, instant de mutation en biologie, etc). On suppose que ces n instants sont corrélés par la copule Gaussienne C de matrice de covariance $\Sigma_{i,i} = 1$ et $\Sigma_{i,j} = \rho \in [0, 1]$ pour $i \neq j$.

Dans un premier temps on s'intéresse à la simulation du vecteur $U = (U_1, \dots, U_n)$ correspondant à la copule C i.e. pour tout $i \in \{1, \dots, n\}, U_i \sim \mathcal{U}([0, 1])$ et $C(u_1, \dots, u_n) = \mathbf{P}[U_1 \leq u_1, \dots, U_n \leq u_n]$. On peut montrer que U se représente sous la forme

$$\forall i \in \{1, \dots, n\}, \quad U_i = \Phi\big(\sqrt{\rho} G_0 + \sqrt{1 - \rho} G_i\big),$$

où $(G_0, G_1, \dots, G_n)$ est un vecteur normal standard de $\mathbf{R}^{n+1}$ et Φ est la fonction de répartition gaussienne standard (fonction `norm` dans le module `scipy.stats`).

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
from numpy.random import default_rng
rng = default_rng()
```

2.1.1. Question: simulation de la copule gaussienne

Ecrire une fonction qui prend 3 arguments: `size`, `n` et `rho` et qui renvoie un échantillon de taille `size` de réalisations $(U_1, \dots, U_n)$ obtenues par la construction précédente.
Dans toute la suite `n` est la dimension de la copule qui sera par exemple 10, et non le nombre de réalisations considérées.

```
def copule_gaussienne(size, n, rho):
    G = rng.standard_normal(size = (n+1, size))
    G_correl = np.sqrt(rho) * G[0] + np.sqrt(1-rho) * G[1:]
    U = stats.norm.cdf(G_correl)
    return U
```

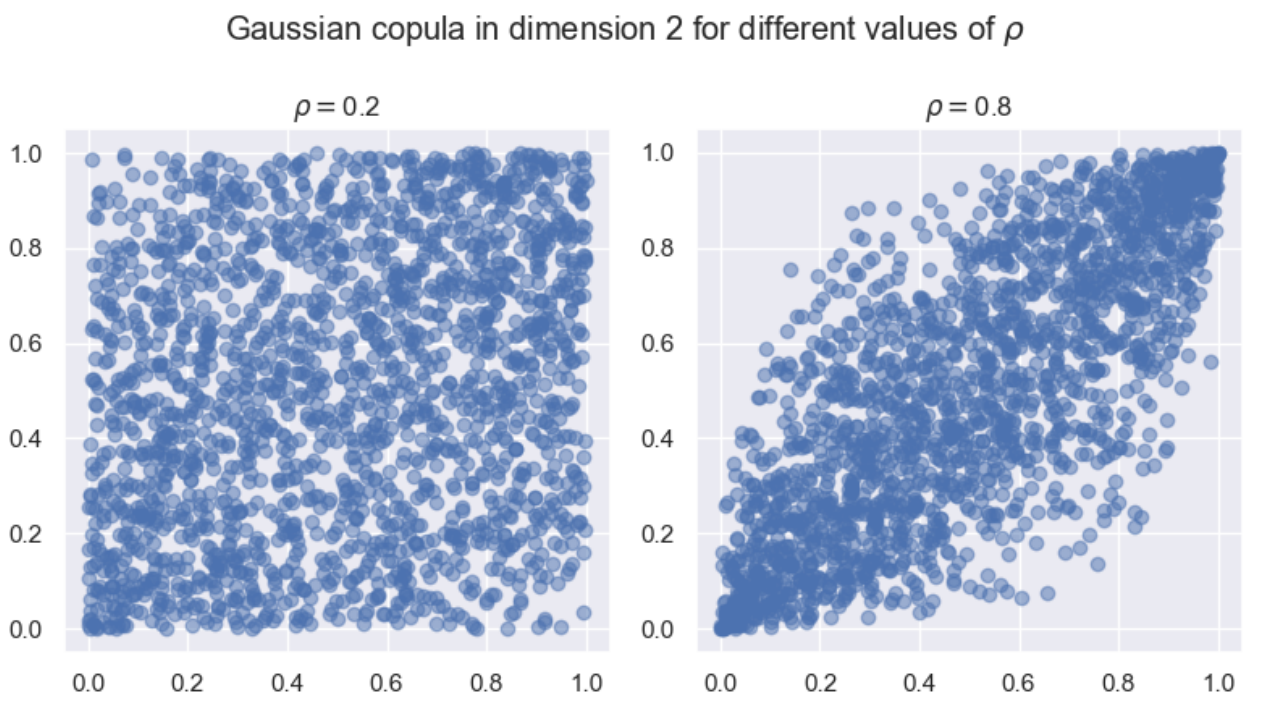
```
size, n = 10000, 2
rho = 0.5
U = copule_gaussienne(size, n, rho)
U
```

```
array([[0.18176204, 0.20001917, 0.55798772, ..., 0.27996056, 0.08020953,
        0.13098167],
       [0.23338515, 0.31995263, 0.40600634, ..., 0.6634561 , 0.43504759,
        0.06431932]])
```

2.1.2. Question: représentation de la copule gaussienne

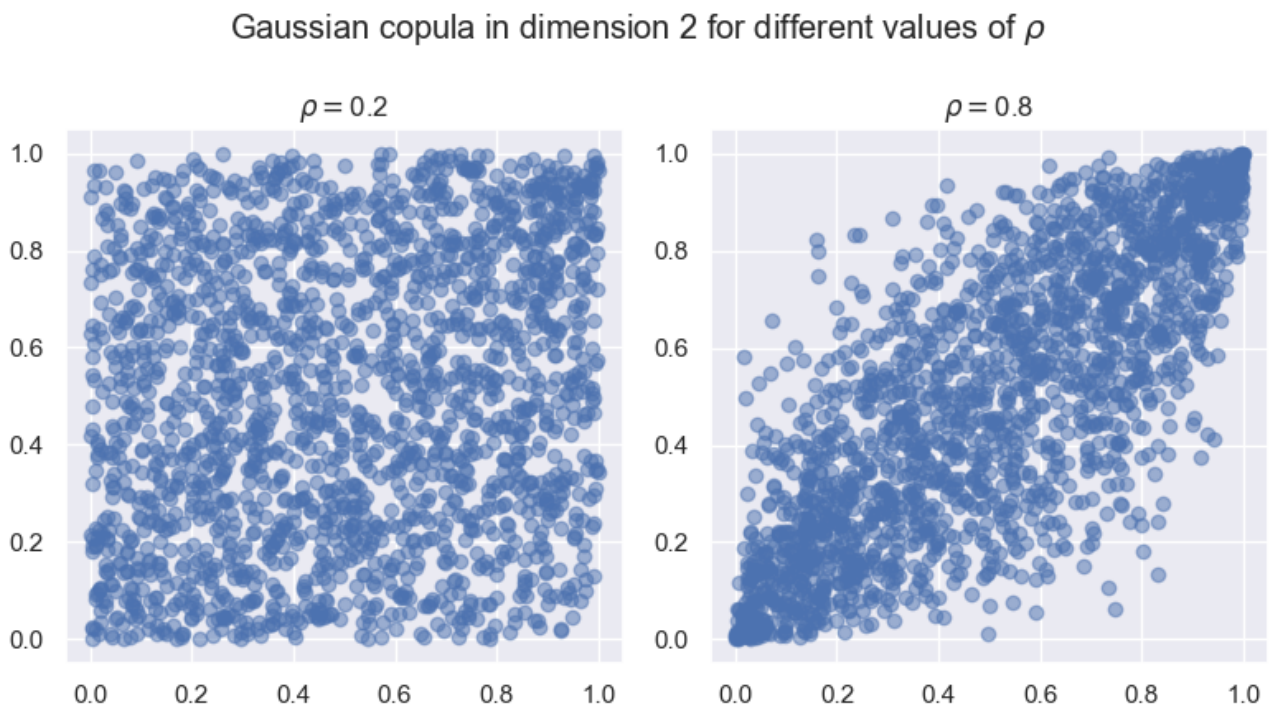
Reproduire le tracé suivant à partir d'échantillons $\left(U_1^{(j)}, U_2^{(j)}\right)_{1\leq j\leq M}$ de taille $M = 2000$ de réalisations de la copule gaussienne en dimension 2 et de corrélation $\rho = 0.2$ puis $\rho = 0.8$. Que remarquez-vous?

Compléter ce tracé en vérifiant graphiquement que la loi empirique de chaque composante $\left(U_i^{(j)}\right)_{1\leq j\leq M}$ pour $i = 1, 2$ est proche de la loi uniforme sur $[0, 1]$. Vous pouvez augmenter la taille de M.




```
size = 2000
n = 2

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(8, 4.5),
layout='tight')
for rho, ax in zip((0.2, 0.8), axs):
    sample = copule_gaussienne(size, n, rho)
    ax.scatter(sample[0], sample[1], alpha=0.5)
    ax.set_title(fr"$\rho=${rho}")
fig.suptitle(fr"Gaussian copula in dimension {n} for different values of
$\rho$")
plt.show()
#plt.savefig('img/copula.png')
```




```
size = 200000
n = 2

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 4.5),
layout='tight')
for rho, ax in zip((0.2, 0.8), axs):
    sample = copule_gaussienne(size, n, rho)
    ax[0].hist(sample[0], density=True)
    ax[1].hist(sample[1], density=True)
    ax[0].set_title(fr"$\rho=${rho}, histogram of the first component")
    ax[1].set_title(fr"$\rho=${rho}, histogram of the second component")
fig.suptitle(fr"Gaussian copula in dimension {n} for different values of
$\rho$")
plt.show()
```


On revient maintenant au problème des instants de défaut dont on rappelle que $\tau_i \sim \mathcal{E}(\lambda_i)$. On s'intéresse à la variable aléatoire

$$X = \sum_{i=1}^{n} \mathbf{1}_{\tau_i \leq 1}.$$

C'est donc une **variable aléatoire discrète** à valeurs dans $\{0, \dots, n\}$.

2.1.3. Question: simulation des instants de défaut

Ecrire une fonction (avec les arguments adhoc) pour simuler un échantillon de X.
On utilise dans la suite $n = 20$ instants de défaut et $\lambda_i = 0.4$ pour tout $i \in \{1, \dots, n\}$.

```
def instants_defaut(size, n, rho, lambd):
    U = copule_gaussienne(size, n, rho)
    tau = -np.log(U) / lambd    # ou E.ppf(U) si E =
stats.expon(scale=1/lambd)
    return np.sum(tau <= 1, axis=0)
# il est important d'apprendre à utiliser les vecteurs de booléens: une
valeur True vaut 1 et une valeur False vaut 0
```

```
instants_defaut(size=100, n=20, rho=0.4, lambd=0.4)
```

```
array([ 0,  2,  2, 13,  0,  0,  3,  2, 13, 11, 13,  3, 11, 14,  3,  5,  0,
        9,  4,  3,  2,  7, 12, 10,  6,  5,  6,  4,  0,  7, 10,  4, 18, 17,
        4,  3, 13,  4,  9,  5, 16, 10,  2,  1,  7,  0, 11, 13,  8, 12,  0,
        4,  3, 14,  2,  4,  1,  5,  1,  2,  5, 10,  8,  3,  5,  5,  5,  1,
        0,  2,  0, 12, 13,  6, 10,  5,  3,  3,  4,  5, 10,  7,  6,  1,  5,
        1, 13,  2,  1,  9,  2,  8,  6,  1, 12,  7,  4,  2,  3,  3])
```

2.1.4. Question: création d'un ensemble de scénarios `samples`

Créer un vecteur `rhos` de taille 4 avec les valeurs $(0, 0.2, 0.6, 0.8)$ et créer une matrice `samples` de 4 lignes et 20000 colonnes, dont chaque ligne sera remplie par des simulations de X pour une valeur de `rho` donnée. Remplir cette matrice avec réalisations de X. Il faut que les éléments de la matrice `sample` soit de type entier `np.int64`.

Cette matrice sera utilisée dans la suite pour le calcul de la moyenne et le tracé des histogrammes.

```
size = 100000
n = 20
lambd = 0.4
rhos = [0, 0.2, 0.6, 0.8]
samples = np.empty((len(rhos), size), dtype=np.int64)
for k, rho in enumerate(rhos):
    samples[k] = instants_defaut(size, n, rho, lambd)
```

2.1.5. Question: calcul de la moyenne

Calculer (mathématiquement) $\mathbf{E}[X]$ en fonction de ρ et vérifier ce résultat en calculant la moyenne empirique de chaque ligne de la matrice `samples` des réalisations de X.

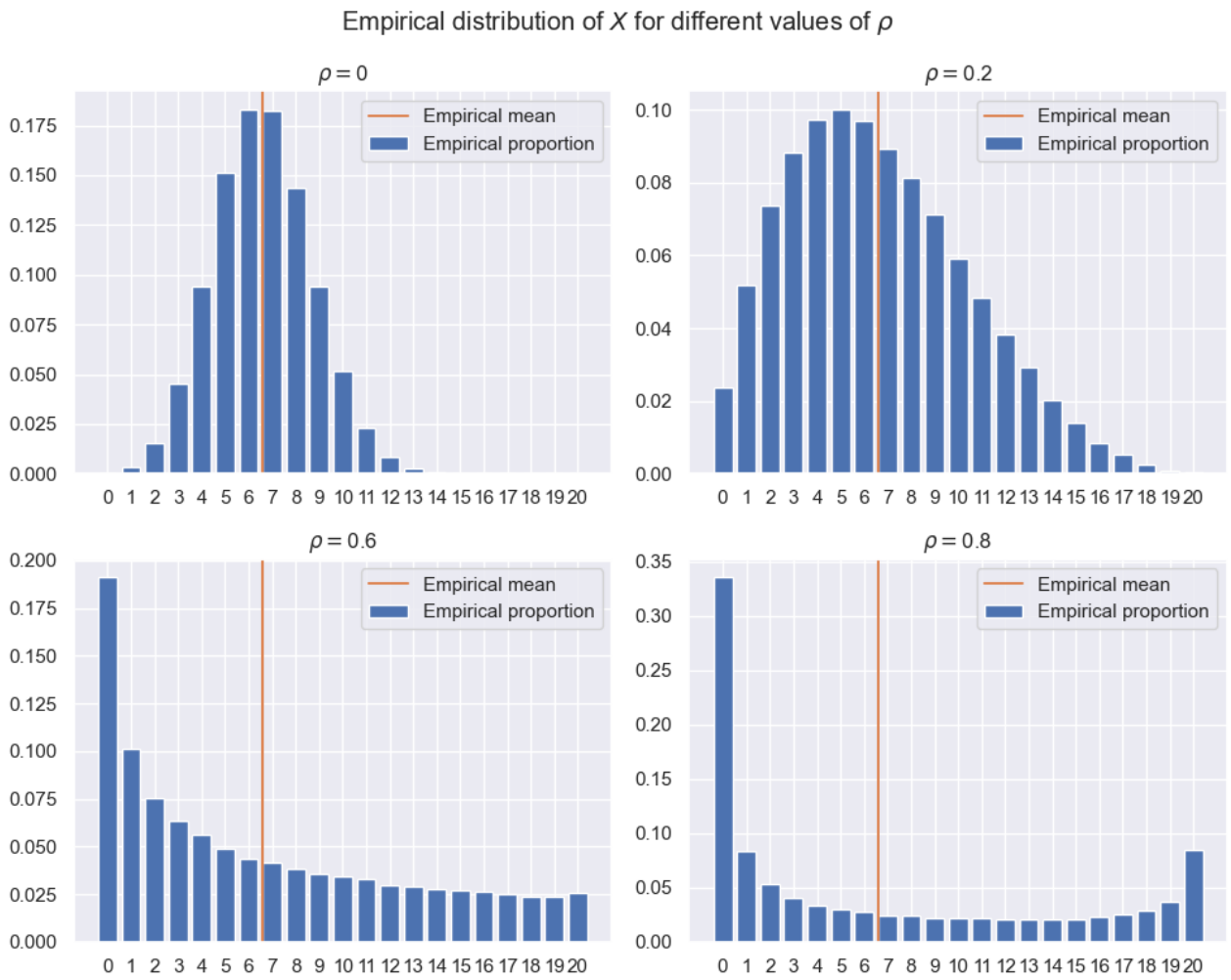
```
print("theoritical mean:", n*(1-np.exp(-lambd)))
```

```
theoritical mean: 6.5935990792872134
```

```
print("empirical means for different values of rho:", "\n rhos = ", rhos,
"\n means", samples.mean(axis=1))
```

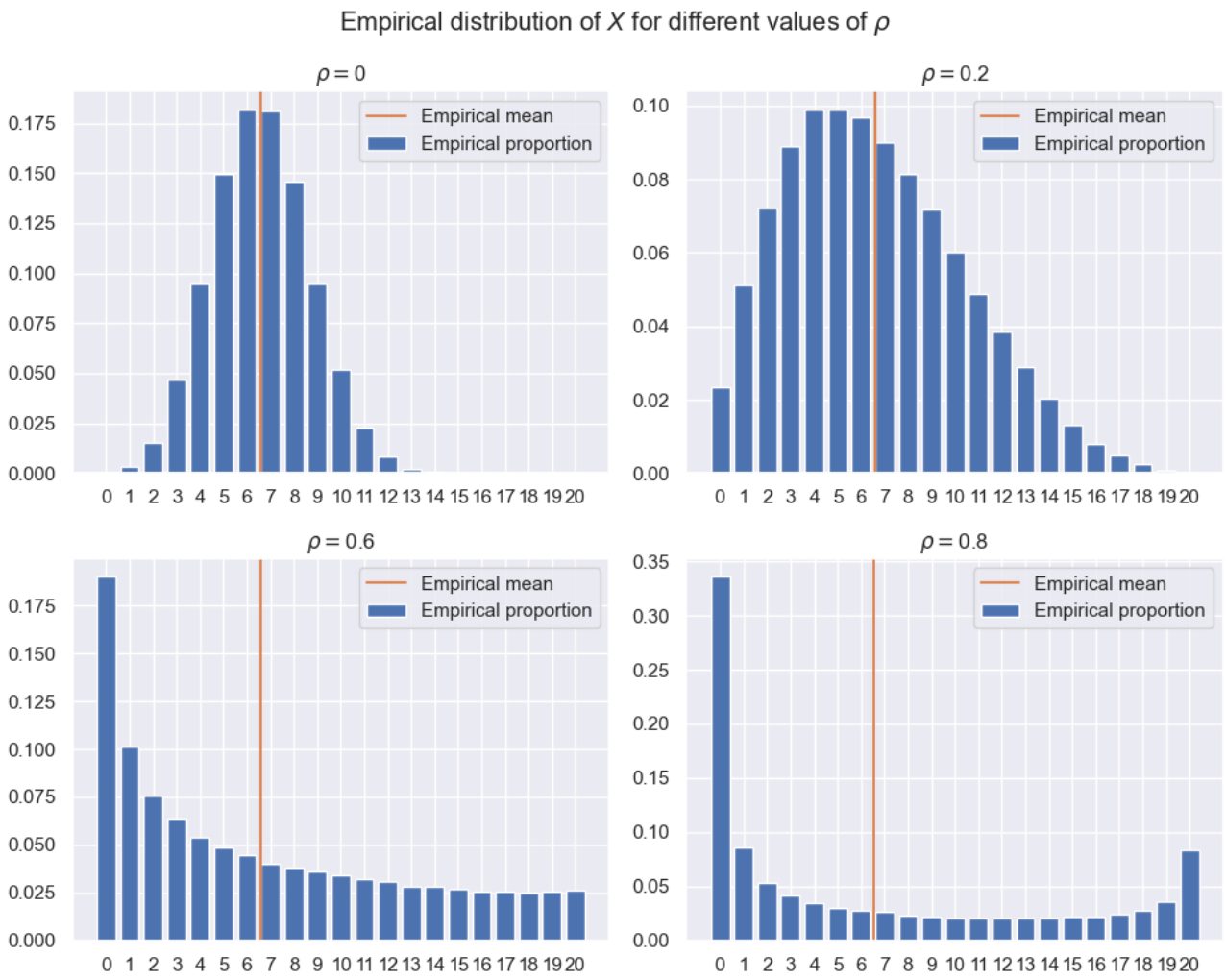
```
empirical means for different values of rho:
 rhos =  [0, 0.2, 0.6, 0.8]
 means [6.59299 6.5938  6.61265 6.54002]
```

2.1.6. Question: représentation graphique de `samples`




```
support = np.arange(n+1)

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 8), layout='tight')
for rho, ax, sample in zip(rhos, axs.flat, samples):
    empirical_prop = np.bincount(sample, minlength=n+1) / size
    ax.bar(support, empirical_prop, label='Empirical proportion')
    ax.axvline(sample.mean(), ymin=0, ymax=1.0, color='C1',
label='Empirical mean')
    ax.set_title(fr"$\rho=${rho}")
    ax.set_xticks(support)
    ax.legend()
fig.suptitle(fr"Empirical distribution of $X$ for different values of
$\rho$")
plt.show()
#plt.savefig('img/empirical_X.png')
```



2.2. Simulation d’une chaîne de Markov: Urnes d’Ehrenfest

On considère d balles ($d > 1$) numérotées de 1 à d et réparties initialement dans deux urnes A et B. On note $E = \{1, \dots, d\}$ l'ensemble des balles et on s'intéresse à l'évolution du contenu des urnes après un nombre $n \geq 1$ de changements d'états. Un changement d'état est modélisé de la façon suivante: "*on tire un numéro de balle selon la loi uniforme sur E et à un tirage i on déplace la balle numéro i d'une urne à l'autre*". Le contenu des urnes change au cours du temps et on note A_n le contenu de A à l'itération n (de même B_n est le contenu de B au temps n). Ce modèle porte le nom d'Urnes d'Ehrenfest.

Dans le programme de test on prendra $d = 20$ c'est à dire $E = \{1, \dots, 20\}$ et le contenu initial $A_0 = \{1, \dots, 10\}$.

2.2.1. Approche naïve

On commence par une approche naïve qui consiste à programmer l'évolution du contenu de l'urne. Si on s'intéresse uniquement au nombre de balles dans chaque urne, il est plus commode d'utiliser la propriété de Markov du système, aussi bien pour l'étude mathématique que pour la programmation.

2.2.1.1. Question: pour manipuler le type `list`

En utilisant le type `list` initaliser les listes `E` et `A_0`, puis programmer l'évolution du contenu de l'urne pas à pas (par exemple en utilisant les méthodes `remove` et `append`). Afficher le résultat `A_n` et `B_n` après `n = 100` itérations.

```
d = 20
E = list(range(1,d+1))
A = list(range(1,11))
B = [i for i in E if not(i in A)]

n = 100
A_0 = A.copy()
for k in range(n):
    i = rng.integers(1, d+1)
    if i in A:
        A.remove(i)
    else:
        A.append(i)
A_n = A.copy()
B_n = [i for i in E if not(i in A_n)]
```

```
print(f"A_0 = {A_0}")
print(f"A_n = {A_n}")
print(f"B_n = {B_n}")
```

```
A_0 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
A_n = [10, 19, 6, 15, 11, 9]
B_n = [1, 2, 3, 4, 5, 7, 8, 12, 13, 14, 16, 17, 18, 20]
```

2.2.1.2. Question: pour manipuler le type `set`

Reprendre la question précédente avec le type `set` et les méthodes associées: `add` et `remove`.

```
d = 20
E = set(range(1,d+1))
A = set(range(1,11))
B = E - A

n = 100
A_0 = A.copy()
for k in range(n):
    i = rng.integers(1, d+1)
    if i in A:
        A.remove(i)
    else:
        A.add(i)
A_n = A.copy()
B_n = E - A_n
```

```
print(f"A_0 = {A_0}")
print(f"A_n = {A_n}")
print(f"B_n = {B_n}")
```

```
A_0 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
A_n = {1, 4, 6, 8, 11, 13, 14, 16, 18, 20}
B_n = {2, 3, 5, 7, 9, 10, 12, 15, 17, 19}
```

2.2.1.3. Question facultative

Cette construction itérative nécessaire dans des dynamiques plus complexes peut être améliorée de la façon suivante. Si on construit un vecteur `sample` des `n` réalisations de la loi uniforme sur E, on peut en déduire directement la composition des urnes à l'itération n. En effet la balle i n'a pas changé d'urne entre 0 et n si et seulement si le numéro i apparait un nombre pair de fois dans `sample`.

Construire directement `A_n` et `B_n` à partir d'un vecteur `sample` de taille `n` qui contient les réalisations de la loi uniforme sur E.

```
d = 20
E = set(range(1,d+1))
A = set(range(1,11))
B = E - A

n = 100
sample = rng.integers(size = n, low=1, high=d+1)
A_n = set()
for k in A:
    if np.sum(sample == k) % 2 == 0:
        A_n.update({k})
for k in E - A:
    if np.sum(sample == k) % 2 == 1:
        A_n.update({k})

B_n = E - A_n
```

```
print(f"A_0 = {A_0}")
print(f"A_n = {A_n}")
print(f"B_n = {B_n}")
```

```
A_0 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
A_n = {16, 19, 8, 11, 12, 15}
B_n = {1, 2, 3, 4, 5, 6, 7, 9, 10, 13, 14, 17, 18, 20}
```

2.2.2. Modélisation en tant que chaîne de Markov

On s'intéresse maintenant non plus à la composition de l'urne A_n mais uniquement à sa taille. On note $X_n = \mathrm{Card}(A_n)$. L'évolution de X_n se fait de la façon suivante: si l'urne A_n contient X_n balles alors la probabilité de tirer une balle présente dans A_n est $\frac{X_n}{d}$. Ainsi avec probabilité $\frac{X_n}{d}$, $X_{n+1} = X_n - 1$ (car on déplace la balle dans l'urne B), et avec probabilité $\frac{d-X_n}{d}$ on a $X_{n+1} = X_n + 1$. La récurrence aléatoire suivante permet de construire une trajectoire $(X_0, \dots, X_n)$

$$\forall k \geq 0, \quad X_{k+1} = X_k + 1 - 2\mathbf{1}_{\left\{U_{k+1} < \frac{X_k}{d}\right\}}, \qquad X_0 \in \{0, \dots, d\},$$

avec $(U_k)_{k\geq 1}$ une suite de variables aléatoires indépendantes de loi uniforme sur $[0, 1]$.

2.2.2.1. Question: simulation de N trajectoires

Ecrire une fonction qui permet de simuler N trajectoires indépendantes $(X_0^{(j)}, \dots, X_n^{(j)})_{1\leq j\leq N}$ de la dynamique précédente, et qui renvoie donc un `np.array` de dimension `(N, n+1)` qui contient ces trajectoires. A vous déterminer les arguments de cette fonction.

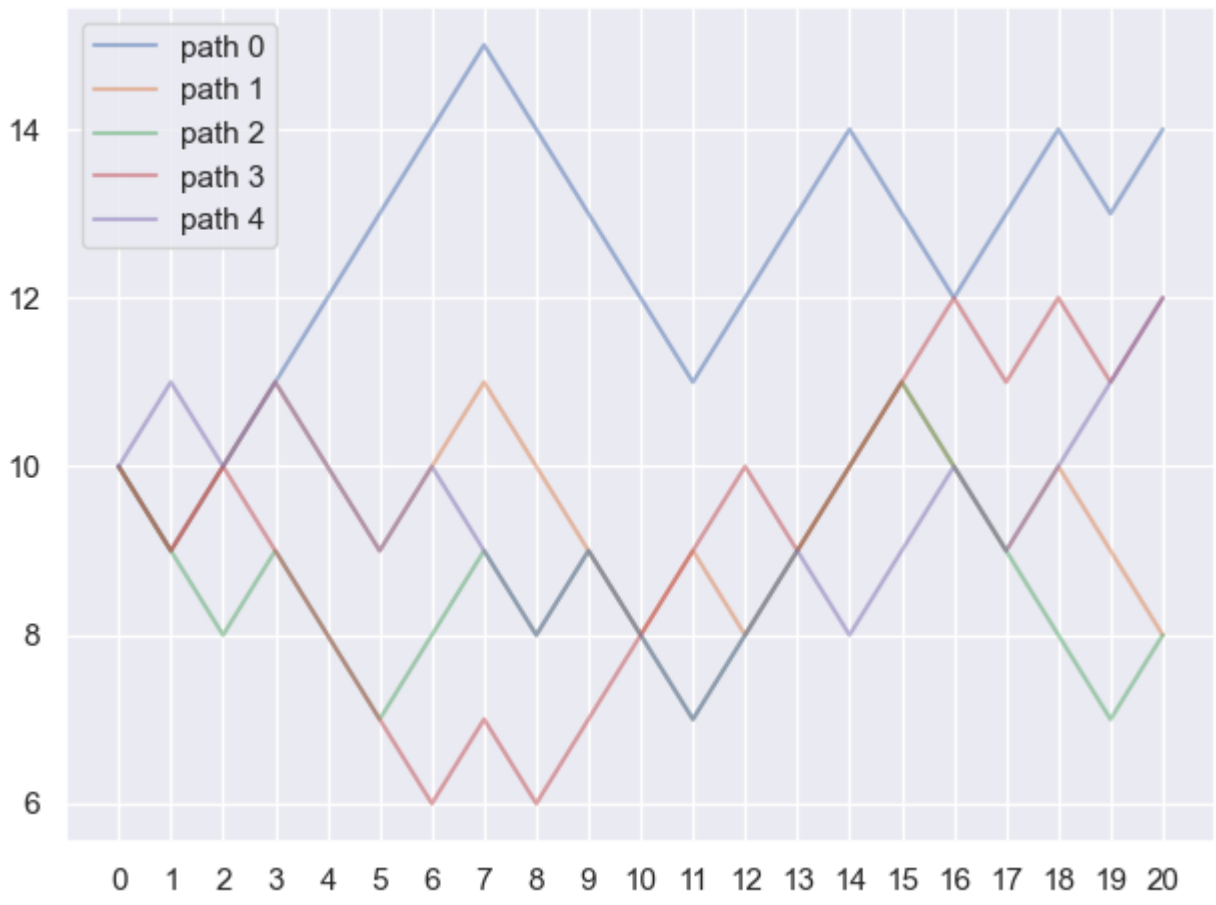
```
def ehrenfest(x0, n, N):
    uniforms = rng.uniform(size=(n, N))
    sample = np.zeros(shape=(n+1, N), dtype=np.int64)
    sample[0] = x0
    for k in range(1,n+1):
        sample[k] = sample[k-1] + 1 - 2 * (uniforms[k-1] < sample[k-1] /
d)
    return sample.T
```

```
ehrenfest(10, 10, 5)
```

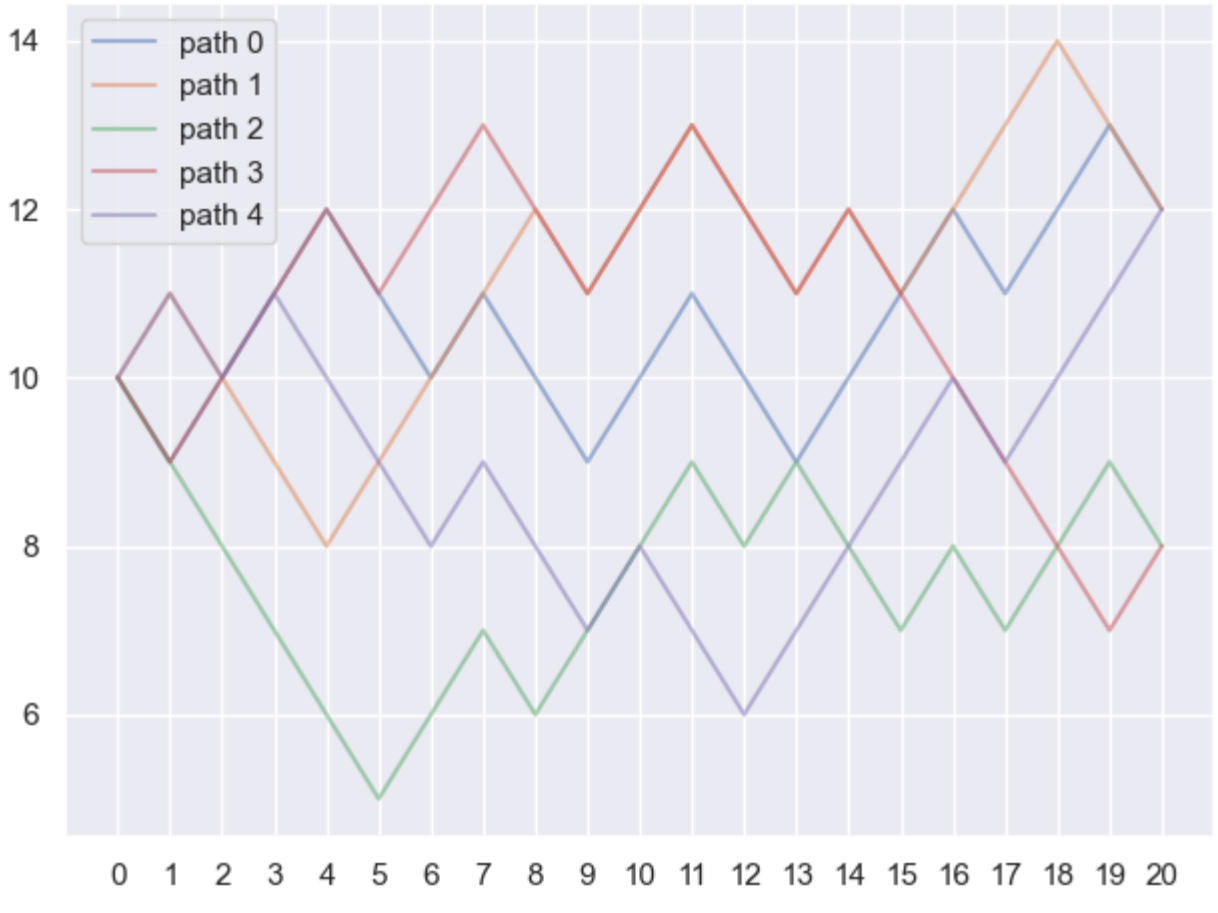
```
array([[10, 11, 12, 13, 12, 11, 12, 13, 14, 13, 12],
       [10,  9,  8,  9, 10, 11, 12, 13, 12, 13, 14],
       [10,  9,  8,  7,  8,  7,  6,  7,  8,  9,  8],
       [10,  9,  8,  9, 10,  9,  8,  9, 10, 11, 12],
       [10, 11, 10, 11, 10,  9,  8,  9, 10, 11, 10]])
```

2.2.2.2. Question: affichage de trajectoires

Reproduire le graphe suivant qui représente l'évolution de 5 trajectoires de $(X_k)_{0\leq k\leq n}$ avec $n = 20$.




```
x0 = 10
n = 20
sample = ehrenfest(x0=x0, n=n, N=5)
fig, ax = plt.subplots(layout='tight')
for j, path in enumerate(sample):
    ax.plot(path, alpha=0.5, label=f"path {j}")
    ax.set_xticks(np.arange(n+1))
    ax.legend()
plt.show()
#plt.savefig('img/paths.png')
```

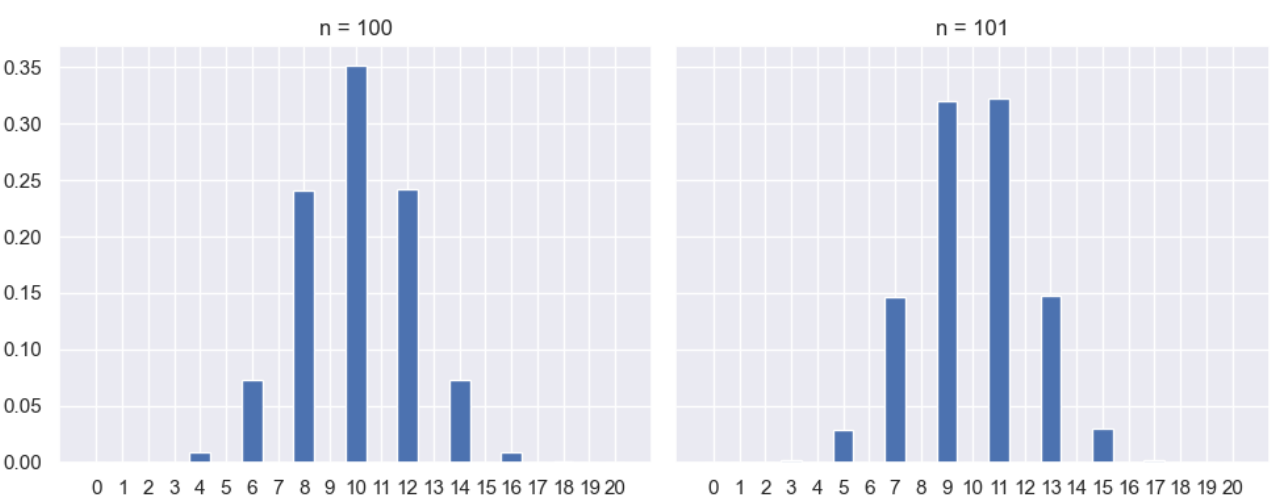


2.2.2.3. Question: représentation de la loi après n itérations

Représenter la distribution empirique de $(X_n^{(j)})_{1\leq j\leq N}$ pour $N = 100\,000$ et deux valeurs de n, $n = 100$ puis $n = 101$. Que remarque-t-on? Etait-ce prévisible?

```
N = 100000

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,4), layout='tight',
sharey=True)
for n, ax in zip((100, 101), axs):
    sample = ehrenfest(x0, n, N)
    support = np.arange(d+1)
    # une façon via bincount
    empirical_prop = np.bincount(sample[:,-1], minlength=d+1) / N
    ax.bar(x=support, height=empirical_prop, label='Empirical proportion')
    # une autre façon via histogram
    #histo = np.histogram(sample[:,-1], bins=np.arange(d+2), density=True)
    #ax.bar(x=support, height=histo[0])
    ax.set_xticks(support)
    ax.set_title(f"n = {n}")
plt.show()
# la chaine de Markov est de période 2
# les lois de X_{2n} et X_{2n+1} sont portées par des ensembles disjoints
```



2.2.2.4. Question: modification de la dynamique

On modifie un peu la modélisation précédente en considérant la règle suivante: *“on tire un numéro de balle selon la loi uniforme sur E et à un tirage i on déplace la balle numéro i d’une urne à l’autre **avec probabilité 1/2**”*.

Refaire les questions précédentes avec ce nouveau modèle. On peut montrer que si $X_n \sim \mathcal{B}\left(d, \frac{1}{2}\right)$ (loi binomiale) alors $X_{n+1} \sim \mathcal{B}\left(d, \frac{1}{2}\right)$. On dit que la loi binomiale $\mathcal{B}\left(d, \frac{1}{2}\right)$ est invariante pour la chaine de Markov. De plus pour toute configuration initiale X_0 la chaine $(X_n)_{n\geq 0}$ converge en loi vers cette mesure invariante $\mathcal{B}\left(d, \frac{1}{2}\right)$. On veut

illustrer cette convergence.

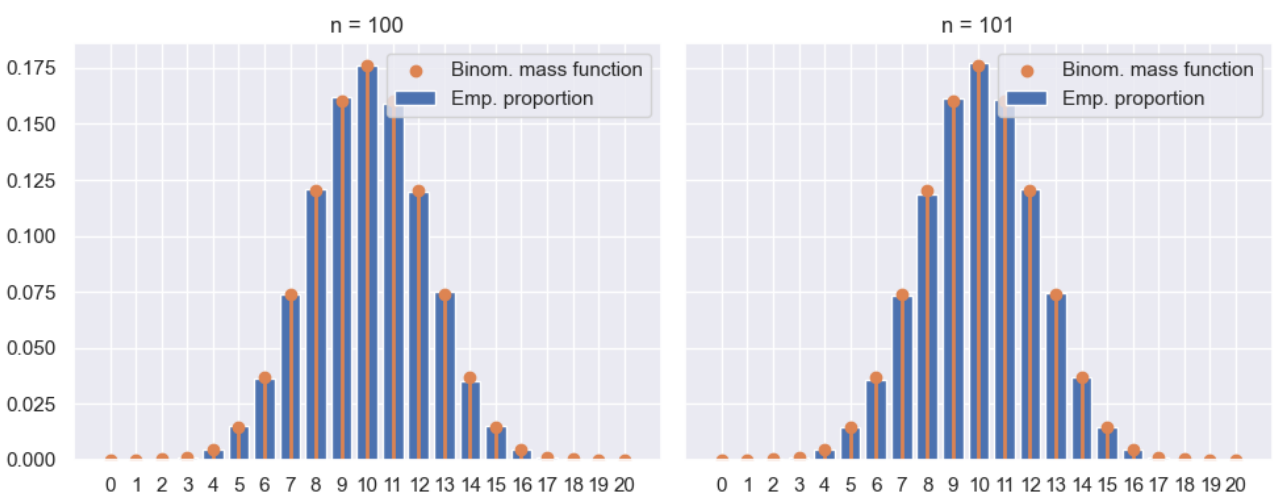
Comparer cette loi binomiale avec l'histogramme empirique de la loi X_n pour n grand (par exemple $n = 100$, vous pouvez choisir X_0 fixé à 10).

```
def ehrenfest_modif(x0, n, N):
    uniforms = rng.uniform(size=(n, 2, N))
    sample = np.zeros(shape=(n+1, N), dtype=np.int64)
    sample[0] = x0
    for k in range(1,n+1):
        sample[k] = sample[k-1] + (1 - 2 * (uniforms[k-1, 0] < sample[k-1]
/ d)) * (uniforms[k-1, 1] < 0.5)
    return sample.T
```

```
N = 100000
binom = stats.binom(n = d, p = 0.5)

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10,4), layout='tight',
sharey=True)
for n, ax in zip((100, 101), axs):
    sample = ehrenfest_modif(x0, n, N)
    support = np.arange(d+1)
    # une façon via bincount
    empirical_prop = np.bincount(sample[:,-1], minlength=d+1) / N
    ax.bar(x=support, height=empirical_prop, label='Emp. proportion')
    # une autre façon via histogram
    #histo = np.histogram(sample[:,-1], bins=np.arange(d+2), density=True)
    #ax.bar(x=support, height=histo[0])

    ax.scatter(support, binom.pmf(support), label='Binom. mass function')
    ax.vlines(support, 0, binom.pmf(support), color='C1', lw=2, alpha=1)
    ax.legend()
    ax.set_xticks(support)
    ax.set_title(f"n = {n}")
plt.show()
```



2.3. Processus de Poisson

On considère un processus de Poisson de paramètre (ou intensité) $\lambda > 0$, c'est à dire un processus de comptage associé à un processus ponctuel $(T_n)_{n\geq 1}$ où les variables aléatoires T_n (appelées instants de sauts) sont définies par

$$\forall n \geq 1, \quad T_n - T_{n-1} = S_n, \qquad \text{en posant } T_0 = 0$$

avec $(S_n)_{n\geq 1}$ suite *i.i.d.* de loi exponentielle de paramètre $\lambda > 0$.

Pour tout $t \geq 0$, on définit

$$N_t = \sum_{n\geq 0} \mathbf{1}_{T_n \leq t},$$

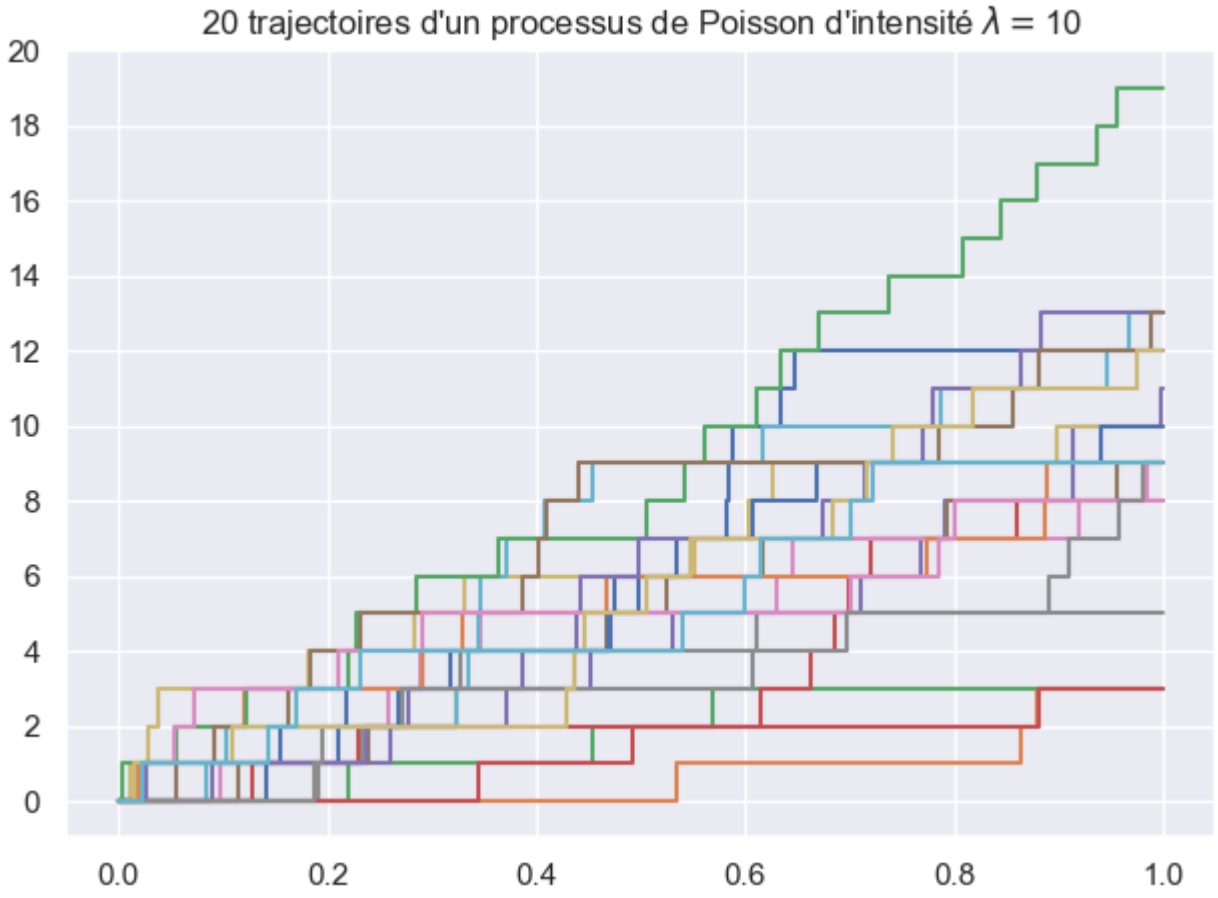
et on veut simuler une trajectoire de $(N_t)_{t\in[0,T]}$ pour un horizon $T > 0$ fixé.

2.3.1. Question: simulation de trajectoires

Ecrire une fonction `one_poisson_path(lambd, T)` qui renvoie les instants $(T_0, \dots, T_n)$ d'un processus de Poisson d'intensité `lambd` restreint à $[0, T]$. Par convention, on veut que

le dernier point du tableau renvoyé soit T.

Reproduire le tracé suivant.

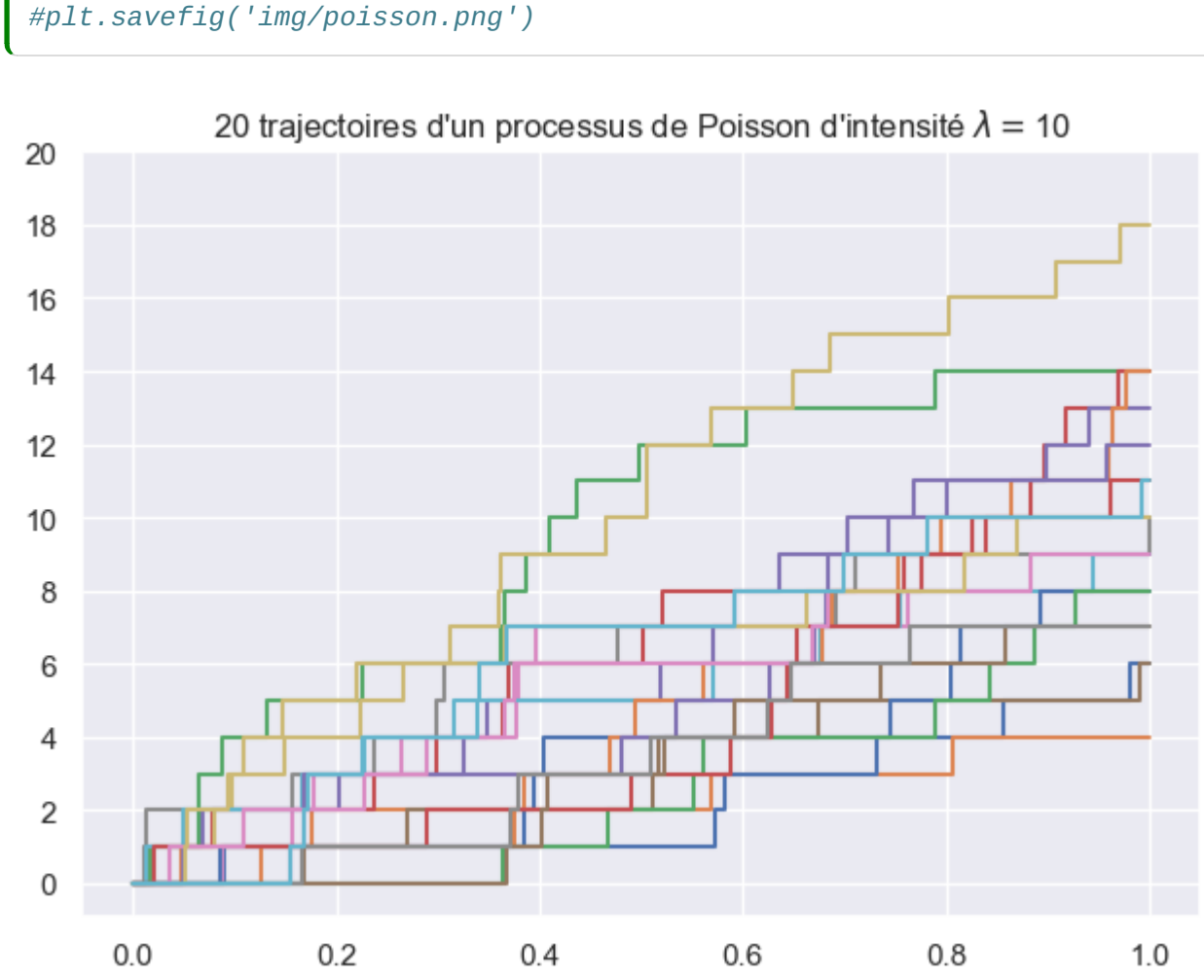



```
def one_poisson_path(lambd, T):
    times = np.zeros(1)
    Tk = 0
    while Tk < T:
        Tk = Tk + rng.standard_exponential() / lambd
        times = np.append(times, Tk)
    times[-1] = T
    return times
```

```
n = 20
lambd, T = 10, 1
paths = [ one_poisson_path(lambd, T) for _ in range(n) ]

def count(size):
    return np.append(np.arange(size), size-1)

fig, ax = plt.subplots(layout='tight')
for path in paths:
    ax.step(path, count(len(path)-1), where='post')
    ax.set_title(fr"{n} trajectoires d'un processus de Poisson d'intensité
$\lambda=${lambd}")
    ax.set_yticks(2*np.arange(0, 11))
plt.show()
#plt.savefig('img/poisson.png')
```



2.3.2. Question: simulation alternative

On rappelle que si $(N_t)_{t\geq 0}$ est un processus de Poisson d'intensité $\lambda > 0$, alors conditionnellement à l'événement $N_T = n$ les instants de sauts $(T_k)_{k=1,\dots,n}$ (tels que $0 < T_1 < \cdots < T_n \leq T$) ont même loi que le réordonnement croissant d'un vecteur $(U_1, \dots, U_n) \sim \mathcal{U}([0,T]^n)$.

Reprendre la question précédente en utilisant cette propriété.


```
def one_poisson_path_alt(lambd, T):
    size = rng.poisson(lam=lambd*T)
    times = np.empty(size+2)
    times[0] = 0
    times[1:-1] = np.sort(rng.uniform(size=size, low=0, high=T))
    times[-1] = T
    return times
```

```
n = 20
lambd, T = 10, 1
paths = [ one_poisson_path_alt(lambd, T) for _ in range(n) ]

def count(size):
    return np.append(np.arange(size), size-1)

fig, ax = plt.subplots(layout='tight')
for path in paths:
    ax.step(path, count(len(path)-1), where='post')
    ax.set_title(fr"{n} trajectoires d'un processus de Poisson d'intensité $\lambda=${lambd}")
    ax.set_yticks(2*np.arange(0, 11))
plt.show()
#plt.savefig('img/poisson.png')
```