TD n° 6 : Composantes connexes de graphes aléatoires

Toutes les variables aléatoires (v.a.) des exercices suivants sont définies sur un espace $(\Omega, \mathcal{F}, \mathbb{P})$. Comme à chaque séance, nous importons les packages suivants :

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
```

Exercice 1. **(Composantes connexes)** Soit $G = (V, E)$ un graphe fini sur les sommets $V = \{0, 1, \dots, n-1\}$, et soit $M = (\mathbf{1}_{\{i,j\}\in E})_{0\le i,j\le n-1}$ sa matrice d'adjacence. L'objectif de cet exercice est d'implémenter une fonction `ConComps` qui prend en argument la matrice M (pour un n non fixé) et renvoie la liste des composantes connexes du graphe associé. On propose l'algorithme suivant (illustré dans la Figure 1) :

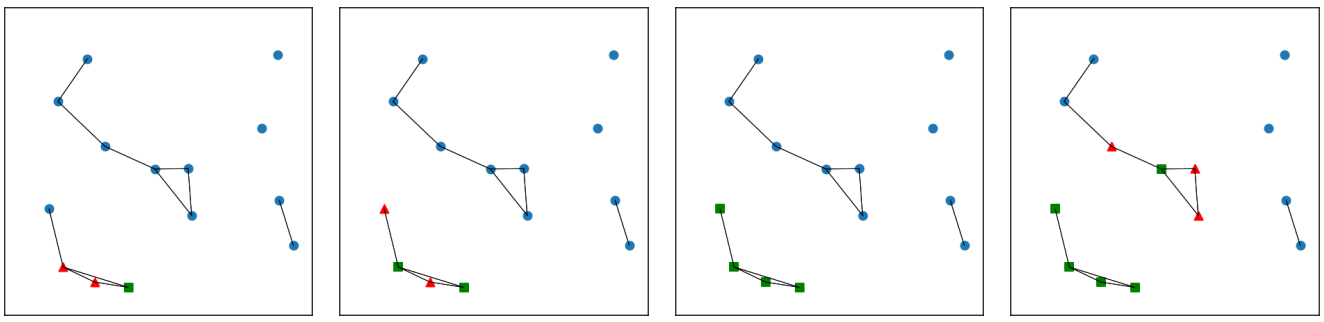
(a) Chaque sommet se voit associer un statut, qui peut être 0 ("inexploré"), -1 ("en attente") ou $+1$ ("exploré"). On stocke ces statuts dans un array `Statut` et on initialise tous les statuts à 0, "inexploré". On initialise également une liste vide `Comps`

(b) On procède récursivement. On choisit un sommet inexploré x. On le marque comme "exploré", on crée une liste `CurrentComp=[x]`, et on marque tous les voisins de x comme "en attente".

(c) S'il y a au moins un sommet "en attente" y, on le rajoute à la liste `CurrentComp`, on le marque comme "exploré", et pour chacun de ses voisins z, s'il était marqué comme "inexploré", alors on change son statut en "en attente".

(d) On itère l'étape (c) tant qu'il y a des sommets "en attente". S'il n'y a plus aucun sommet "en attente", alors on a fini de remplir `CurrentComp`, et on le rajoute à la liste `Comps`.

(e) S'il reste des sommets "inexplorés", alors on revient à l'étape (b), en créant une nouvelle composante connexe à partir d'un sommet "inexploré". Si tous les sommets sont "explorés", alors l'algorithme est terminé et on renvoie `Comps`.



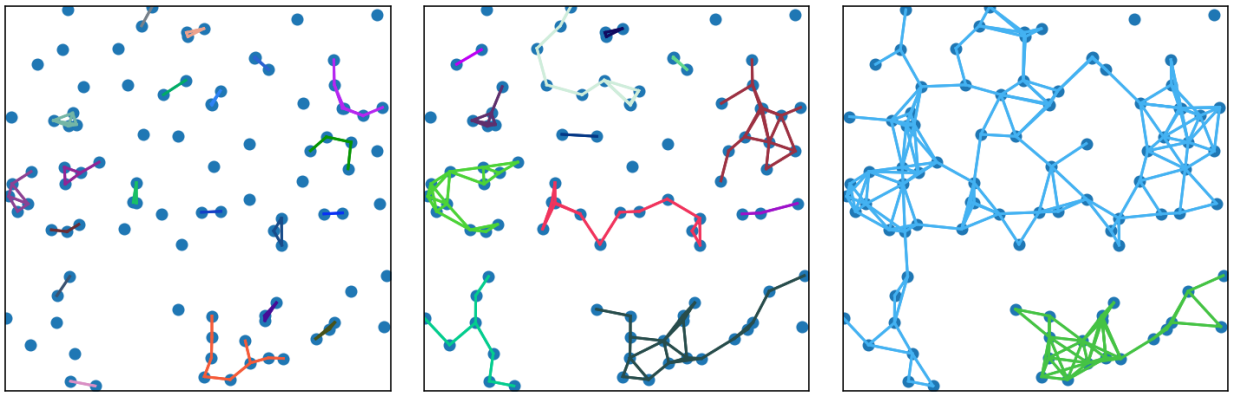
(1) (2) (3) (4)

Figure 1 – Illustration de l'algorithme. Les sommets "inexplorés" sont les points bleus, ceux "en attente" les triangles rouges, et ceux "explorés" les carrés verts. On commence par explorer un sommet, et marquer tous ses voisins "en attente", voir l'étape (1). Puis on explore les sommets "en attente" un à un, et dès qu'un tel sommet a un voisin précédemment "inexploré", il est aussi marqué "en attente", voir l'étape (2). On continue jusqu'à ce qu'il n'y ait plus aucun sommet en attente, voir l'étape (3). Les sommets explorés jusqu'alors forment une des composantes connexes du graphe. Pour déterminer la composante connexe suivante, on recommence cette procédure en explorant un nouveau sommet qui était marqué "inexploré", et en marquant ses voisins "en attente" à nouveau, voir l'étape (4).

1. Implémenter la fonction `ConComps` en utilisant l'algorithme ci-dessus.
2. En s'inspirant de la fonction `Plot2DEdges` du TD 4, implémenter une fonction `Plot2DComps` qui prend en argument un nombre $T > 0$, une mesure ponctuelle $\pi = \sum_{i=0}^{n-1} \delta_{x_i}$ sur $[-T, T]^2$, et une matrice d'adjacence $M \in \{0,1\}^{n \times n}$, et qui réalise une représentation graphique de π sur laquelle on a tracé les arêtes $\{x_i, x_j\}$ données par la matrice M, en utilisant une **couleur différente** pour chacune des composantes connexes.
Indication : une couleur en Python peut être définie au format rgb par un triplet de flottants entre 0 et 1. Par exemple, une couleur aléatoire peut être générée par la ligne suivante :

```
randcolor = st.uniform.rvs(size=3)
```

3. *(Application)* Simuler une PPM et représenter graphiquement son graphe booléen à distance R associé et ses composantes connexes. On pourra réutiliser les fonctions `Pi` et `BooleanEdges` du TD 4, et on pourra réaliser plusieurs simulations en faisant varier le paramètre R.



Exercice 2. **(Introduction à la percolation continue)**

1. Implémenter une fonction `LargestCompSize` qui prend en argument la matrice d'adjacence d'un graphe et renvoie la proportion de sommets qui sont dans la plus grande composante connexe. On appellera cette quantité la "taille renormalisée de la composante connexe principale" (Par exemple si un graphe contient 20 sommets, dont 8 dans la plus grande composante connexe, alors la valeur renvoyée par `LargestCompSize` est 0.4).

On considère des réalisations Π de PPM homogène sur $[-T, T]^2$ d'intensité λ et leur graphe booléen à distance R associés, et on souhaite calculer la taille renormalisée de leur composante connexe principale pour différentes valeurs de R. On pose `Rvals = np.linspace(0,2,20)` et on prendra $T = 2$, $\lambda = 1$ dans les questions ci-dessous.

2. Générer une matrice de taille $(30, 20)$ telle que chaque entrée (k, i) contient la taille renormalisée de la composante connexe principale du graphe booléen à distance `Rvals[i]` d'une PPM de loi Π. On s'assurera que les PPM sont indépendantes pour différentes valeurs de k, mais elles peuvent être les mêmes pour un k fixé et différentes valeurs de i.
3. Générer `Means` les moyennes empirique des colonnes de la matrice ci-dessus, puis représenter graphiquement `Means` en fonction de R dans `Rvals`.
4. Refaire les questions 2 et 3 en augmentant progressivement la valeur de T, jusqu'à un maximum de $T = 10$ (si votre ordinateur le permet). Quel phénomène observez-vous ?
5. Générer une PPM et représenter graphiquement son graphe booléen associé pour différentes valeurs de R (on fera attention à toujours utiliser la même réalisation de la PPM, et on pourra ignorer le calcul des composantes connexes et utiliser `Plot2DEdges` directement). Est-ce que vous observez le même phénomène qu'à la question précédente ?