consensus_M2

March 17, 2025

1 Sujet de TP M2 Data Mining 2 - Graph Mining

L'objectif de cette séance est de mettre en oeuvre des méthodes de détection de communautés (clustering des sommets d'un graphe) en appliquant différentes techniques vues en cours. Le TP est :

- À faire en solo ou en binôme
- À rendre sous forme de notebook Jupyter au plus tard le dimanche 19 novembre à 23h59

Ce sujet a été testé sur la machine virtuelle "ULR Ubuntu 20.04". Sur cette machine, avant de lancer Jupyter, vous devez avoir téléchargé le fichier Jupyter, puis dans un terminal allez dans le dossier où se trouve le fichier Jupyter, et faites :

- python -m venv env
- source env/bin/activate
- pip install ipykernel
- python -m ipykernel install –name=env –user
- jupyter-notebook
- enfin, dans le menu Noyau -> Changer de noyau -> env

Ce sujet devrait aussi marcher sur les installations Linux récentes, et sur Google Collab.

2 Test de l'installation

Les deux cellules ci-dessous permettent de vérifier que tout fonctionne correctement en générant un graphe aléatoire à 10 sommets et 20 arêtes, puis en l'affichant.

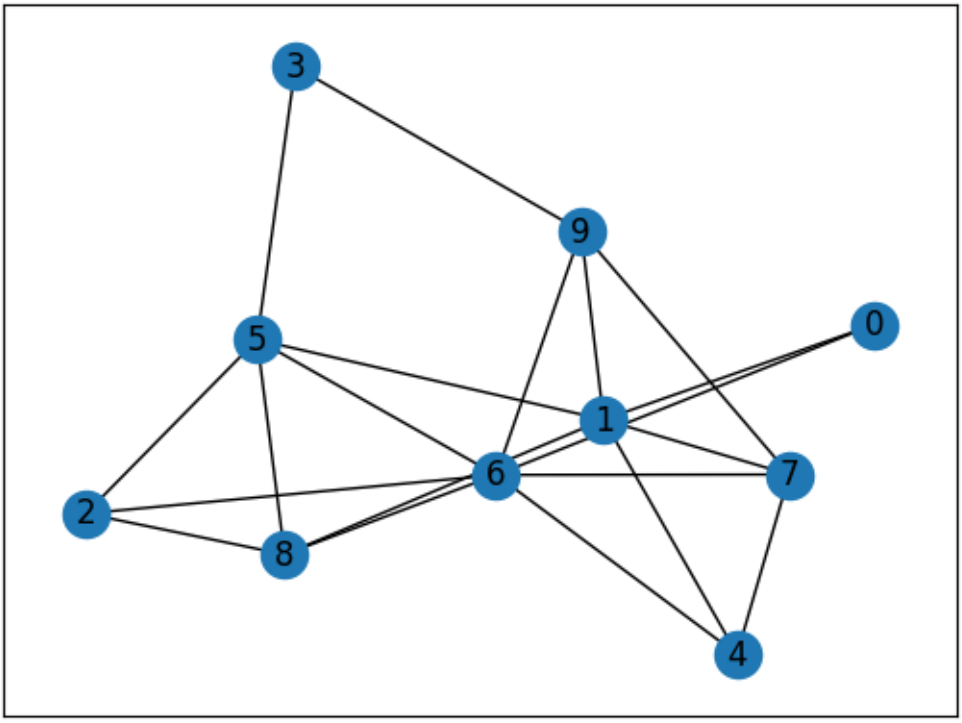
[4]:
```
!pip install partition-networkx
!pip install python-louvain
!pip install matplotlib

import partition_networkx
from community import community_louvain

%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx
import networkx.algorithms.community as nx_comm
import numpy as np
```

```
Collecting partition-networkx
  Obtaining dependency information for partition-networkx from https://files.pyt
honhosted.org/packages/71/1e/5f808a735b5c19a38d334ae07c482268e68e02e546523813a56
411334c80/partition_networkx-0.0.2-py3-none-any.whl.metadata
  Using cached partition_networkx-0.0.2-py3-none-any.whl.metadata (3.8 kB)
Collecting networkx (from partition-networkx)
  Obtaining dependency information for networkx from https://files.pythonhosted.
org/packages/d5/f0/8fbc882ca80cf077f1b246c0e3c3465f7f415439bdea6b899f6b19f61f70/
networkx-3.2.1-py3-none-any.whl.metadata
  Using cached networkx-3.2.1-py3-none-any.whl.metadata (5.2 kB)
Collecting python-louvain (from partition-networkx)
  Using cached python_louvain-0.16-py3-none-any.whl
Collecting numpy (from partition-networkx)
  Obtaining dependency information for numpy from https://files.pythonhosted.org
/packages/8a/08/a7e5dadc21fe193baea5f257e11b7b70cc27a89692fc9e3ed690e55cc4b6/num
py-1.26.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Using cached
numpy-1.26.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(61 kB)
Using cached partition_networkx-0.0.2-py3-none-any.whl (6.1 kB)
Using cached networkx-3.2.1-py3-none-any.whl (1.6 MB)
Using cached
numpy-1.26.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.2
MB)
Installing collected packages: numpy, networkx, python-louvain, partition-
networkx
Successfully installed networkx-3.2.1 numpy-1.26.1 partition-networkx-0.0.2
python-louvain-0.16
Requirement already satisfied: python-louvain in ./env/lib/python3.11/site-
packages (0.16)
Requirement already satisfied: networkx in ./env/lib/python3.11/site-packages
(from python-louvain) (3.2.1)
Requirement already satisfied: numpy in ./env/lib/python3.11/site-packages (from
python-louvain) (1.26.1)
Collecting matplotlib
  Obtaining dependency information for matplotlib from https://files.pythonhoste
d.org/packages/b0/06/1cafe3b9ab069e5fef70aada238a51a23fbdfb4b5e105a45358a01a3823
b/matplotlib-3.8.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Using cached matplotlib-3.8.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.8 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Obtaining dependency information for contourpy>=1.0.1 from https://files.pytho
nhosted.org/packages/e2/83/29a63bbc72839cc6b24b5a0e3d004d4ed4e8439f26460ad9a34e3
9251904/contourpy-1.2.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Using cached contourpy-1.2.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.8 kB)
```

```
Collecting cycler>=0.10 (from matplotlib)
  Obtaining dependency information for cycler>=0.10 from https://files.pythonhos
ted.org/packages/e7/05/c19819d5e3d95294a6f5947fb9b9629efb316b96de511b418c53d245a
ae6/cycler-0.12.1-py3-none-any.whl.metadata
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Obtaining dependency information for fonttools>=4.22.0 from https://files.pyth
onhosted.org/packages/78/3a/aa9fefad66d11f355bb0ff93e83f7d2450c0eac58bf81e545a7f
aa762442/fonttools-4.44.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Using cached fonttools-4.44.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (153 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Obtaining dependency information for kiwisolver>=1.3.1 from https://files.pyth
onhosted.org/packages/17/ba/17a706b232308e65f57deeccae503c268292e6a091313f6ce833
a23093ea/kiwisolver-1.4.5-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Using cached kiwisolver-1.4.5-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.4 kB)
Requirement already satisfied: numpy<2,>=1.21 in ./env/lib/python3.11/site-
packages (from matplotlib) (1.26.1)
Collecting packaging>=20.0 (from matplotlib)
  Obtaining dependency information for packaging>=20.0 from https://files.python
hosted.org/packages/ec/1a/610693ac4ee14fcdf2d9bf3c493370e4f2ef7ae2e19217d7a237ff
42367d/packaging-23.2-py3-none-any.whl.metadata
  Using cached packaging-23.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pillow>=8 (from matplotlib)
  Obtaining dependency information for pillow>=8 from https://files.pythonhosted
.org/packages/6f/d8/f31dd84b4363b5f24c71b25a13ec3855f5ff233e07e1c3f1f8e979e12be2
/Pillow-10.1.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata
  Using cached Pillow-10.1.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (9.5
kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Obtaining dependency information for pyparsing>=2.3.1 from https://files.pytho
nhosted.org/packages/39/92/8486ede85fcc088f1b3dba4ce92dd29d126fd96b0008ea2131679
40a2475/pyparsing-3.1.1-py3-none-any.whl.metadata
  Using cached pyparsing-3.1.1-py3-none-any.whl.metadata (5.1 kB)
Collecting python-dateutil>=2.7 (from matplotlib)
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting six>=1.5 (from python-dateutil>=2.7->matplotlib)
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Using cached
matplotlib-3.8.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(11.6 MB)
Using cached
contourpy-1.2.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (313
kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
```

```
Using cached
fonttools-4.44.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.8
MB)
Using cached
kiwisolver-1.4.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4
MB)
Using cached packaging-23.2-py3-none-any.whl (53 kB)
Using cached Pillow-10.1.0-cp311-cp311-manylinux_2_28_x86_64.whl (3.6 MB)
Using cached pyparsing-3.1.1-py3-none-any.whl (103 kB)
Installing collected packages: six, pyparsing, pillow, packaging, kiwisolver,
fonttools, cycler, contourpy, python-dateutil, matplotlib
Successfully installed contourpy-1.2.0 cycler-0.12.1 fonttools-4.44.0
kiwisolver-1.4.5 matplotlib-3.8.1 packaging-23.2 pillow-10.1.0 pyparsing-3.1.1
python-dateutil-2.8.2 six-1.16.0
```

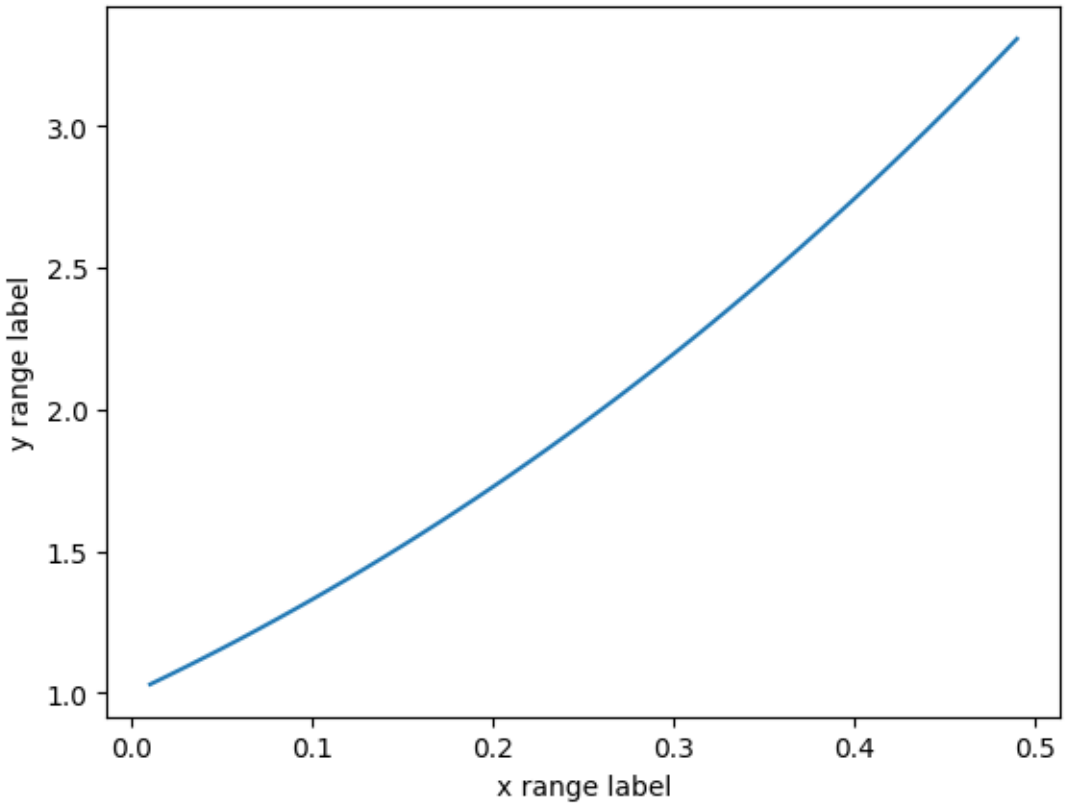
[58]:
```
G = nx.gnm_random_graph(10,20) # Generate a random graph with 10 nodes and 20␣
↪edges
nx.draw_networkx(G)            # Draw G using all default values
```



Voici un exemple de code pour calculer et afficher une courbe simplement. Libre à vous d'utiliser des bouts de ce code dans la suite.

[3]:
```
start_value = 0.01
end_value = 0.5
```

```
step = 0.01

def my_fun(i):
  return (i+1)**3

# Plotting the graph
rang = np.arange(start_value, end_value, step)
plt.plot(rang, [my_fun(i) for i in rang])
plt.xlabel('x range label')
plt.ylabel('y range label')
plt.show()
plt.close()
```



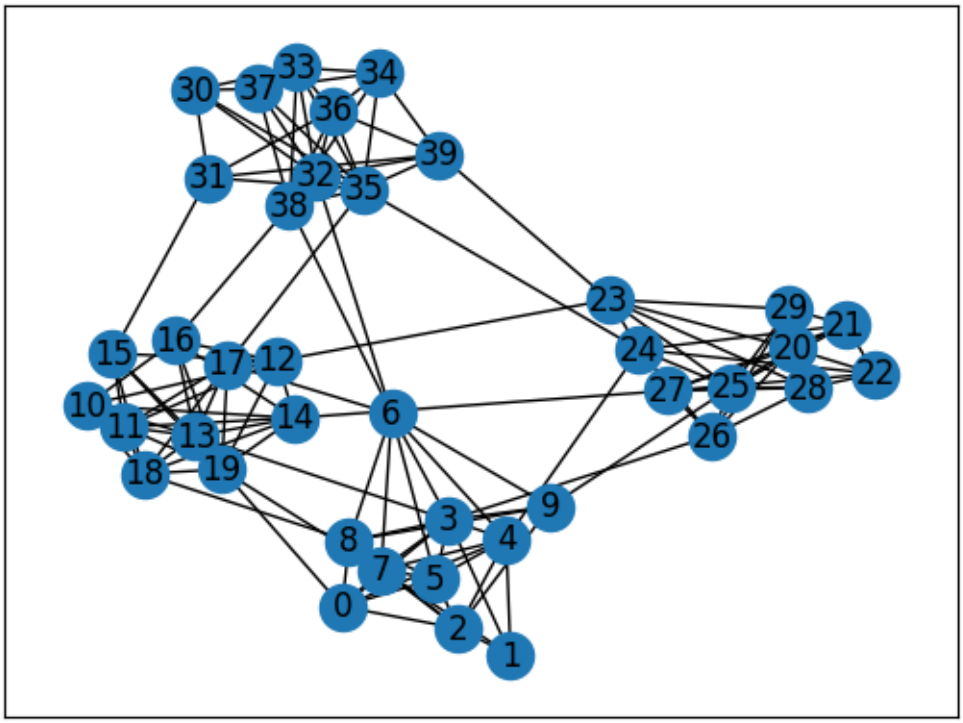
3 Partie 1 - Génération de graphes aléatoires

Nous allons utiliser la fonction `planted_partition_graph` qui permet de générer un graphe aléatoire comportant l groupes, chacun composé de k sommets. Deux sommets du même groupe sont connectés avec probabilité `p_in`, deux sommets de groupes différents sont connectés avec probabilité `p_out`.

Le code ci-dessous génère un graphe de 40 sommets répartis en 4 groupes avec des probabilités de

0.7 et 0.03. Les choix de probabilités devraient rendre les 4 groupes visibles.

[5]:
```
l = 4 # number of groups
k = 10 # number of nodes per group
p_in = 0.7 # intra-group probability of connection
p_out = 0.03 # between-group probability of connection

G = nx.generators.planted_partition_graph(l, k, p_in, p_out)
nx.draw_networkx(G)
```



Dans l'exemple précédent, nous avons généré un graphe contenant 4 groupes de 10 sommets, chacun des groupes ayant une probabilité de connexion interne (`p_in = 0.7`) bien supérieure à la probabilité de connexion entre groupes (`p_out = 0.03`). Nous pouvons donc considérer que ces 4 groupes sont des communautés du graphe.

Voici un exemple de code permettant de construire la vérité terrain pour les 4 groupes générés précédemment, ce qui sera utile pour la suite :

[6]:
```
l = 4 # number of groups
k = 10 # number of nodes per group

gt_part = [set(range(k*i, k*(i+1))) for i in range(l)]
print(gt_part)
```

```
[{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}, {20,
```

```
21, 22, 23, 24, 25, 26, 27, 28, 29}, {32, 33, 34, 35, 36, 37, 38, 39, 30, 31}]
```

Question 1.1 : Expliquer le lien entre les deux probabilités et la structure communautaire du graphe.

Réponse :

Question 1.2 : Implémentez le calcul de la modularité. Vous pourrez ensuite générer un graphe comme dans l'exemple ci-dessus, puis calculer la modularité de la vérité terrain. Vous pourrez vérifier que votre implémentation est correcte en comparant son résultat à la fonction de modularité implémentée dans Networkx.

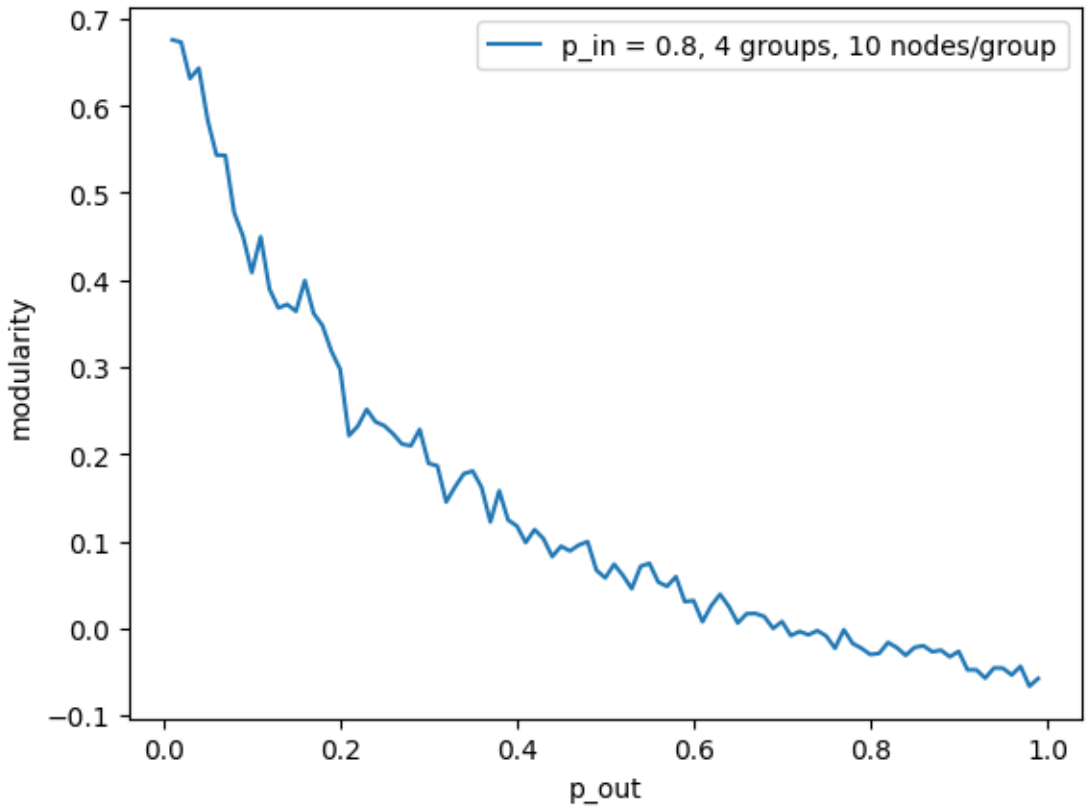
[77] :

```
0.6175491898148149
0.6175491898148149
```

Question 1.3 : Ecrire du code pour tracer une courbe qui va calculer la modularité de cette vérité terrain en fonction de `p_out`, en utilisant 4 groupes de 10 sommets avec une probabilité de connection intra-groupe à 0,8.

Vous pourriez avoir besoin de la fonction suivante pour calculer la modularité https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.qua

[59] :



Question 1.4 : Calculer (mathématiquement) la modularité attendue si `p_out = 0`, et si `p_out = p_in`, pour un graphe dans lequel il y a 4 communautés de 10 sommets chacune.

Réponse :

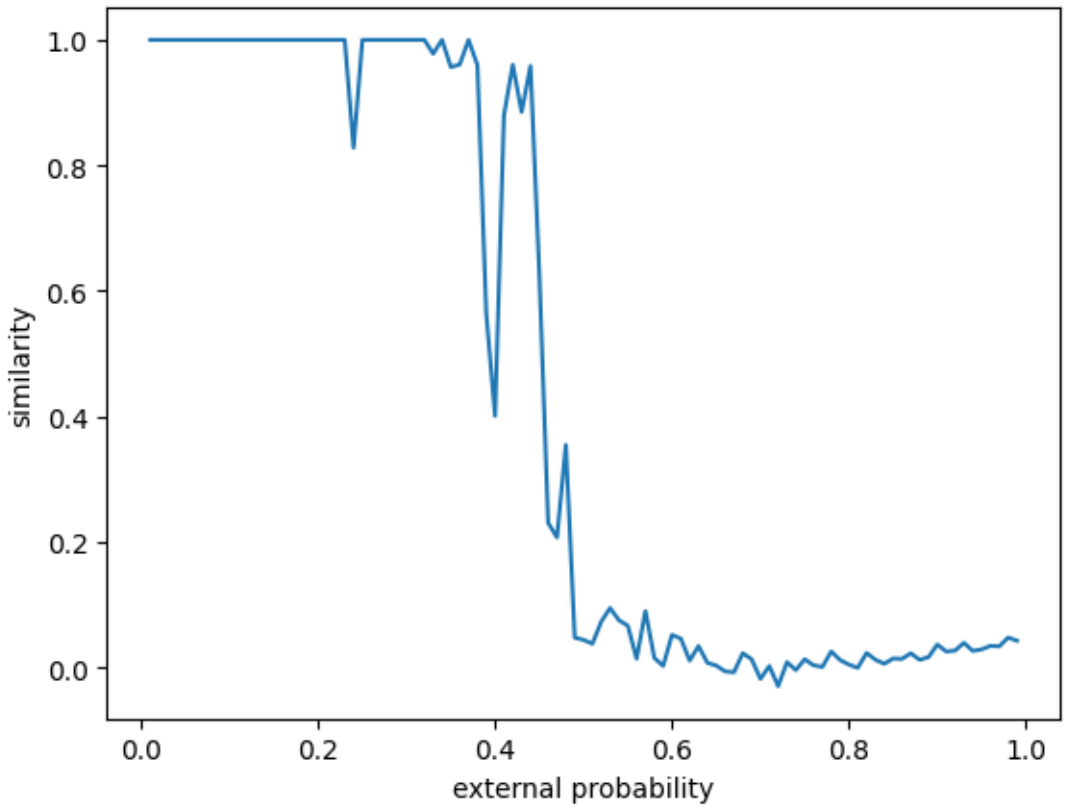
4 Partie 2 - détection de communautés

Nous allons maintenant détecter les communautés du graphe précédent et comparer ces communautés à la vérité terrain. Nous allons écrire du code pour calculer la similarité entre la vérité terrain et le calcul de communautés sur le graphe précédent. A partir de maintenant et pour toute la suite nous considérons 4 groupes de 32 sommets chacun et une probabilité interne (`p_in`) de 0,7. Seule la probabilité externe (`p_out`) va varier.

[43]:
```
k = 32
l = 4
p_in = 0.7
```

Question 2.1 : Écrire une fonction qui détecte les communautés avec la méthode de Louvain et qui compare ces communautés à la vérité terrain en utilisant la fonction `G.gam()` comme décrite dans la documentation. Tracer la courbe de similarité en fonction de `p_out`.

[48]:
```
```

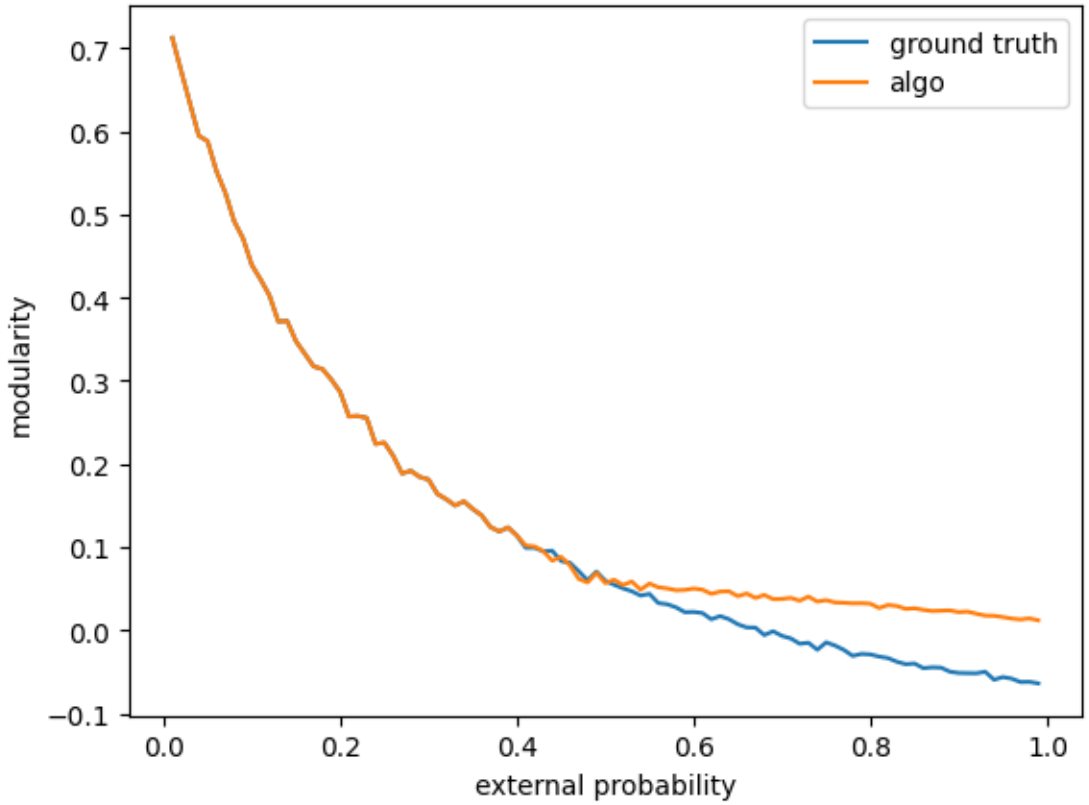


Question 2.2 : Que pouvez-vous conclure de la courbe obtenue à la question 2.1 ? Pensez-vous que l'algorithme utilisé est efficace ?

Réponse :

Question 2.3 : Écrire une fonction qui détecle les communautés avec la méthode de Louvain et qui compare la modularité de ces communautés à celle de la vérité terrain (cf question 1.2). Tracer les deux courbes de modularité.

[47] :



Question 2.4 : A partir des courbes obtenues aux questions 2.1 et 2.3, que pouvez-vous conclure sur la qualité de l'algorithme ?

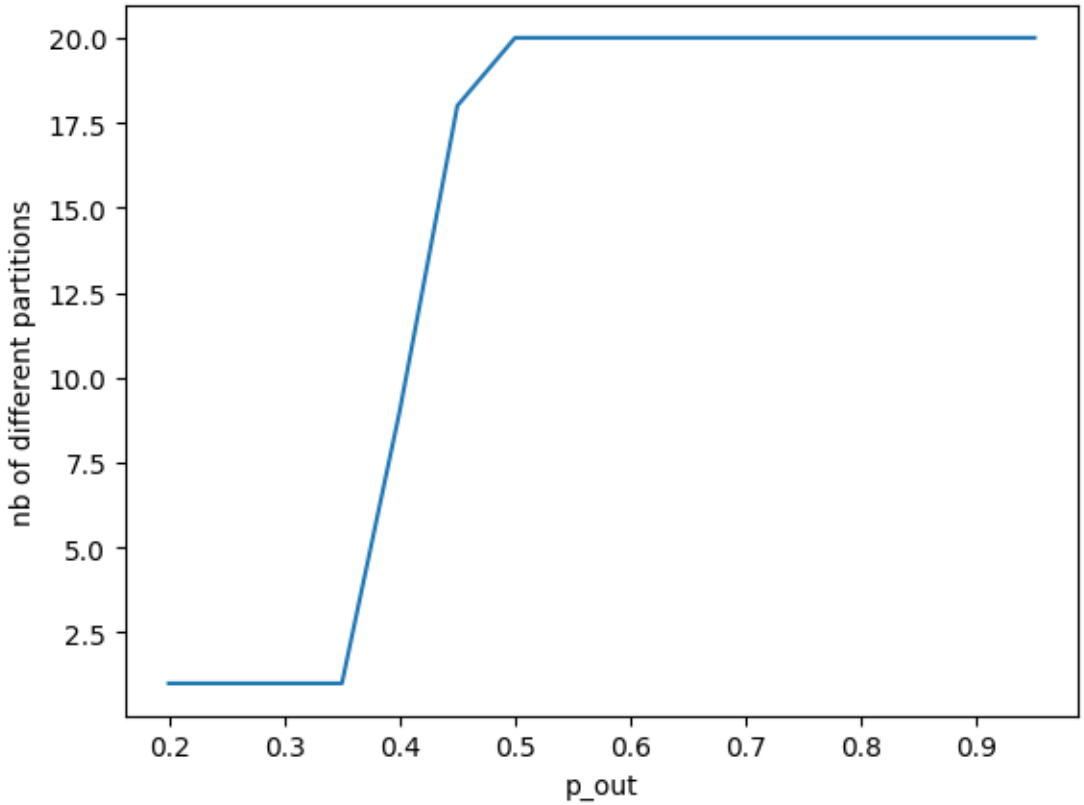
Réponse :

5 Partie 3 - communautés consensuelles

Question 3.1 : Écrire une fonction qui exécute n_p fois la méthode de Louvain sur un graphe, que vous aurez généré comme précédement, de manière à calculer n_p partitions. Calculer le nombre de

partitions différentes que vous avez obtenu parmis vos n_p exécutions. Enfin, tracer le nombre de partitions différentes en fonction de `p_out`.

[46] :

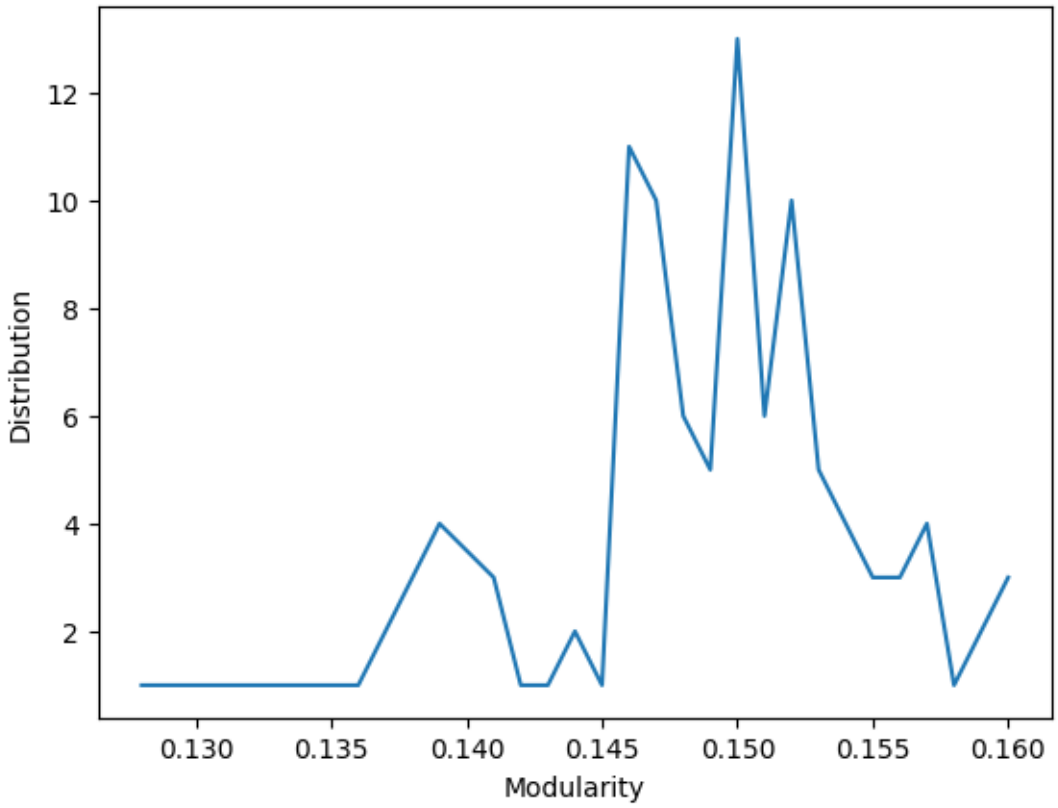


Question 3.2 : Quand `p_out` augmente, le nombre de partitions différentes se rapproche de n_p. Selon vous, pourquoi ?

Réponse :

Question 3.3 : Exécutez n_p fois la méthode de Louvain (avec `randomize=True`) sur un graphe, que vous aurez généré comme précédement, de manière à calculer n_p partitions. Afficher la distribution de la modularité.

[57] :



Question 3.4 : Cette courbe montre que sur un même graphe, plusieurs exécutions de la méthode de Louvain permettent de trouver des partitions de modularité différente. Selon vous, est-ce que la modularité est un bon critère pour départager ces partitions ?

Réponse :

Les deux questions précédentes montrent que les algorithmes de détections de communautés que nous utilisons ne sont pas déterministes. Les partitions qu'ils calculent ne sont pas toujours les meilleures. Les communautés consensuelles permettent de corriger ces problèmes en aggrégeant toutes les partitions dans une matrice de consensus, depuis laquelle on peut calculer des communautés consensuelles.

Question 3.5 : Calcul de communautés consensuelles. Calculez de nouveau n_p partitions grace à la méthode de Louvain (avec `randomize=True`), puis construisez une matrice de consensus P, où P_{ij} est égal au nombre de fois où le sommet i et le sommet j on été mis dans la même communautés au cours des n_p exécutions de la méthode de Louvain. P_{ij} est donc compris entre 0 et n_p. $P_{ij} = 0$ si i et j n'ont jamais été dans la même communauté. $P_{ij} = n_p$ si i et j on été dans la même communauté à chaque exécution. Pour vous faciliter la question suivante, P peut être construit avec `np.array()`.

[13]:

```
[[10  5  3 …  4  3  7]
```

```
 [ 5 10  4 …  3  3  3]
 [ 3  4 10 …  2  1  2]
 …
 [ 4  3  2 … 10  2  6]
 [ 3  3  1 …  2 10  1]
 [ 7  3  2 …  6  1 10]]
```

Question 3.6 : La matrice P peut-être considérée comme une matrice d'adjacence. Construire le graphe pondéré G_p, correspondant à la matrice P, puis exécuter une dernière fois la méthode de Louvain sur ce graphe pour obtenir des communautés consensuelles. Vous pouvez utiliser nx.from_numpy_array() pour importer la matrice P.

[15]:

```
{0: 2, 1: 0, 2: 1, 3: 2, 4: 1, 5: 3, 6: 0, 7: 2, 8: 2, 9: 1, 10: 2, 11: 3, 12:
2, 13: 1, 14: 1, 15: 1, 16: 4, 17: 3, 18: 2, 19: 2, 20: 1, 21: 3, 22: 4, 23: 2,
24: 3, 25: 3, 26: 1, 27: 0, 28: 3, 29: 3, 30: 0, 31: 4, 32: 3, 33: 0, 34: 4, 35:
1, 36: 1, 37: 4, 38: 2, 39: 1, 40: 2, 41: 4, 42: 0, 43: 4, 44: 4, 45: 4, 46: 3,
47: 2, 48: 3, 49: 2, 50: 3, 51: 4, 52: 1, 53: 1, 54: 3, 55: 4, 56: 1, 57: 4, 58:
1, 59: 1, 60: 3, 61: 4, 62: 2, 63: 4, 64: 3, 65: 4, 66: 3, 67: 0, 68: 0, 69: 4,
70: 3, 71: 4, 72: 3, 73: 1, 74: 3, 75: 3, 76: 0, 77: 3, 78: 2, 79: 3, 80: 0, 81:
4, 82: 4, 83: 3, 84: 2, 85: 3, 86: 4, 87: 4, 88: 0, 89: 4, 90: 4, 91: 3, 92: 1,
93: 4, 94: 1, 95: 2, 96: 3, 97: 4, 98: 1, 99: 3, 100: 1, 101: 4, 102: 3, 103: 2,
104: 1, 105: 1, 106: 4, 107: 2, 108: 2, 109: 4, 110: 1, 111: 0, 112: 3, 113: 1,
114: 1, 115: 3, 116: 3, 117: 1, 118: 4, 119: 1, 120: 4, 121: 4, 122: 3, 123: 4,
124: 1, 125: 1, 126: 4, 127: 3}
```

Question 3.7 : Générer un graphe G, puis calculer plusieurs communautés consensuelles sur G et afficher leur modularité.

[ ]:

Question 3.8 : Conclure sur l'intérêt des communautés consensuelles.

[ ]: