

# Etude de la propagation de l'information : le cas des fake-news

Comment modéliser et influencer la propagation d'informations sur un réseau social ?

---

MARTIN Lucas

n°54331

# Modélisation d'un réseau de comptes aléatoire

- On considère un réseau social composé de  $n$  utilisateurs.
- On associe à chaque utilisateur un compte personnel représenté initialement par 5 paramètres:
  - 1) Un identifiant
  - 2) Un nombre d'abonnés
  - 3) Une nature (public ou privé)
  - 4) Une proximité
  - 5) Une fiabilité

# Modélisation d'un réseau de comptes aléatoire

## La proximité :

- Entier compris entre 0 et prox\_max
- Plus 2 utilisateurs ont une proximité proche, plus ils ont de chances de s'abonner entre eux

# Modélisation d'un réseau de comptes aléatoire

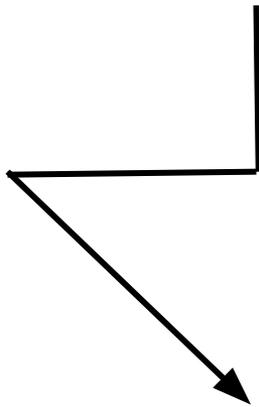
## La fiabilité :

- 1) Sert à déterminer quel genre d'information partage l'utilisateur
  
- 2) Elle peut prendre 3 valeurs :
  - 1 si le compte de l'utilisateur diffuse des fake news
  - 2 si il est douteux
  - 3 si il est fiable

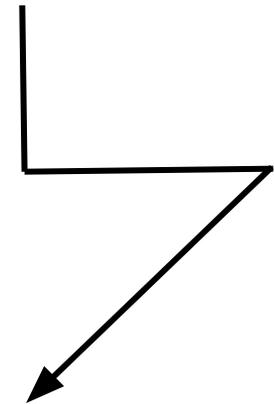
# Modélisation d'un réseau de comptes aléatoire

On note  $n$  le nombre de comptes dans le réseau

[liste\_id, liste\_nombre\_abonnés, liste\_proximité, liste\_public, liste\_fiabilité]



Liste de taille  $n$   
Le  $i$ ème élément donne le nombre  
d'abonnés du  $i$ ème compte

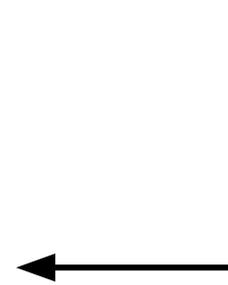


Liste de taille  $n$   
Le  $i$ ème élément donne la nature  
du  $i$ ème compte(public/privé)

# Modélisation d'un réseau de comptes aléatoire

```
def creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3):
```

Nombre de comptes de catégorie de fiabilité 2



Le nombre  $n$  de comptes sur le réseau est égale à la somme des nombres de comptes de fiabilité 1, 2, et 3

# Modélisation d'un réseau de comptes aléatoire

- Identifiant : entier compris 0 et  $n - 1$
- Nature : 0 pour privé, 1 pour public
- Proximité : entier aléatoire compris entre 1 et  $\text{prox\_max}$
- Fiabilité : 1, 2, ou 3
- Nombre d'abonnés : entier aléatoire inférieur à  $n$

# Modélisation d'un réseau de comptes aléatoire

```
pop = len(comptes[0])
for i in range(pop):
    abannes_compte = []
    while len(abannes_compte) < comptes[1][i]:
        j = random.randint(0, pop - 1)
```

-

-

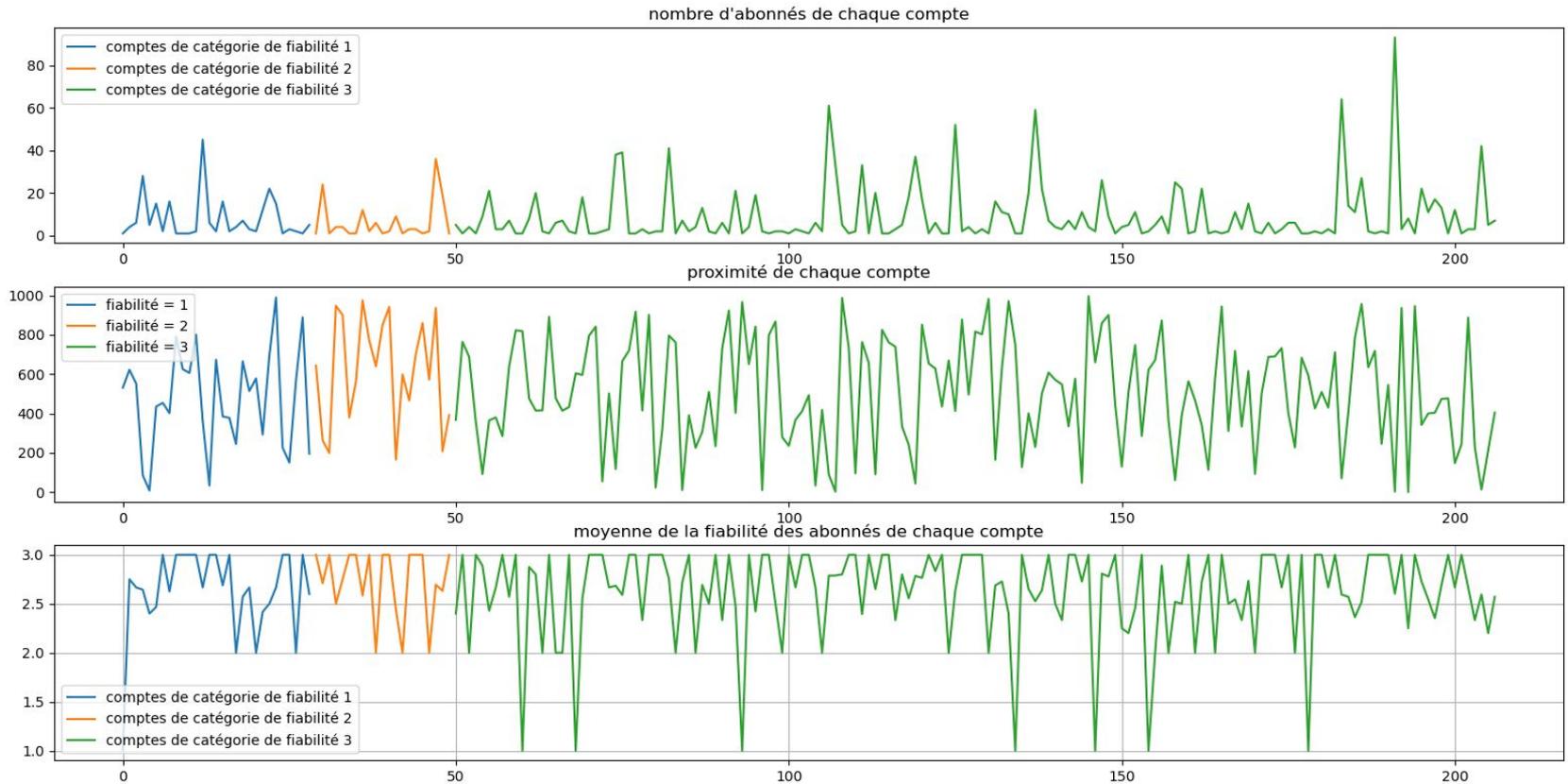
On vérifie que le compte d'identifiant j n'est pas dans la liste des abonnés

-

Plus les 2 comptes ont une proximité proche, plus ils ont de chances de s'abonner entre eux

```
a = comptes[2][i] - comptes[2][j]
c = []
for _ in range(abs(a)):
    c.append(0)
while len(c) < prox_max:
    c.append(1)
id = random.choice(c)
if id > 0:
    abannes_compte.append(j)
```

# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes aléatoire

```
# Le compte retweet-t-il? La probabilité qu'il retweet fluctue en fonction de sa fiabilité
# si la fonction retourne 1, le compte partage l'information
# Dans la simulation 1 cette fonction sert pour le gain d'abonnés car la propagation de l'information se fait uniquement par l'intermédiaire des abonnés et non du partage de l'information
def retweet_fiabilite(fiabilite):
    retweet = 0
    valeur_possible = []
    for i in range(0, 10001, 1):
        valeur_possible.append(i)
    # si la valeur choisie est compris entre 0 et la probabilité de retweet, retweet = 1
    valeur_choisie = random.choice(valeur_possible)
    if fiabilite == 1:
        if valeur_choisie < 151:
            retweet = 1
    elif fiabilite == 2:
        if valeur_choisie < 46:
            retweet = 1
    elif fiabilite == 3:
        if valeur_choisie < 15:
            retweet = 1
    return retweet
```

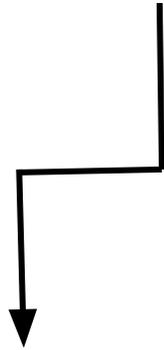
- Gain d'abonnés  
potentiel



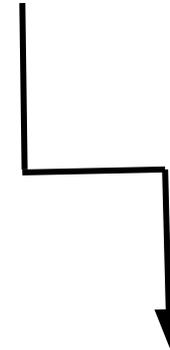
```
if retweet_fiabilite(comptes[4][i]) == 1:
    a = comptes[2][i] - comptes[2][j]
    c = []
    for _ in range(abs(a)):
        c.append(0)
    while len(c) < prox_max:
        c.append(1)
    id = random.choice(c)
    if id > 0:
        comptes[5][i].append(j)
```

# Modélisation d'un réseau de comptes aléatoire

$[0, 1, 0, 0, \dots, 0, 1, 1]$



L'utilisateur d'identifiant 0  
n' a pas l'information partagée



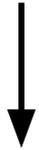
L'utilisateur d'identifiant n  
a l'information partagée

# Modélisation d'un réseau de comptes aléatoire

Initialement, un utilisateur a l'information



$[0, 0, \dots, 0, 1, 0, \dots, 0, 0]$



Il partage l'information



Evolution du nombre d'utilisateurs qui ont l'information

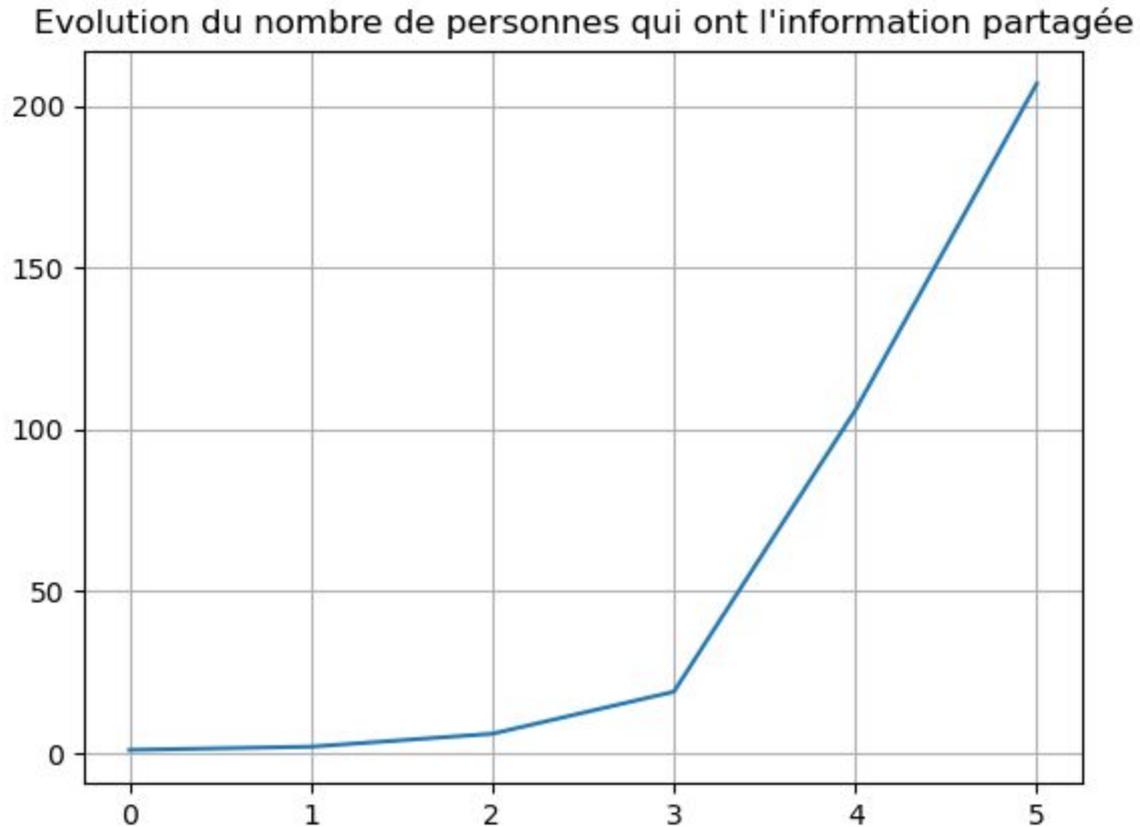


A la fin, tous les utilisateurs ont l'information partagée



$[1, 1, \dots, 1, 1]$

# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes aléatoire

On fait partir l'information de chaque compte du réseau social

fiabilité	1	2	3
simulation 1	6,25	6,5	6,45
simulation 2	7,6	7	7,2
simulation 3	5,5	5,75	5,5

Moyenne des temps de propagation de l'information de chaque catégorie de fiabilité

# Modélisation d'un réseau de comptes

## Problèmes

- La fiabilité du compte de l'utilisateur n'influence pas son nombre d'abonnés
- La fiabilité du compte de l'utilisateur n'influence pas sa proximité, ses abonnés, ses abonnements
- Si un compte est abonné à un autre compte qui a l'information, alors il a lui aussi l'information
- On ne connaît pas la nature de l'information partagée (vraie/fausse).
- Les utilisateurs n'ont pas d'avis sur l'information partagée

# Modélisation d'un réseau de comptes aléatoire

Le nombre d'abonnés des comptes a une distribution exponentielle

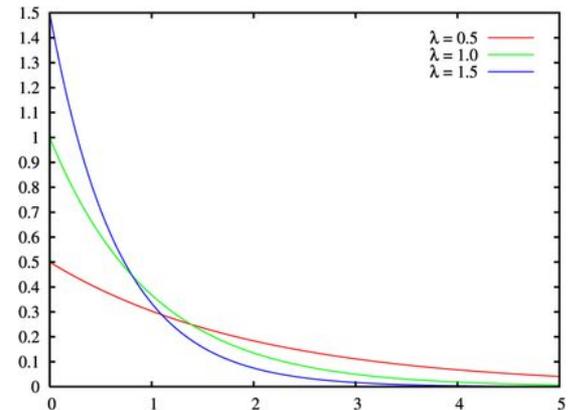


On choisit une moyenne de la distribution exponentielle différente en fonction de la catégorie de fiabilité du compte



# Le / 200 est choisi pour avoir des nombres moyens d'abonnés acceptables

```
moy_1 = pop / 200
moy_2 = moy_1 * 6
moy_3 = moy_1 * 36
for i in range(pop):
    if liste_fiabilite[i] == 1:
        abo = int(random.expovariate(1 / moy_1))
        while abo > pop * 8 / 10 or 3 > abo:
            abo = int(random.expovariate(1 / moy_1))
        liste_nb_abonnes.append(abo)
    elif liste_fiabilite[i] == 2:
        abo = int(random.expovariate(1 / moy_2))
        while abo > pop * 8 / 10 or 3 > abo:
            abo = int(random.expovariate(1 / moy_2))
        liste_nb_abonnes.append(abo)
    else:
        abo = int(random.expovariate(1 / moy_3))
        while abo > pop * 8 / 10 or 3 > abo:
            abo = int(random.expovariate(1 / moy_3))
        liste_nb_abonnes.append(abo)
```



[https://fr.wikipedia.org/wiki/Loi\\_exponentielle](https://fr.wikipedia.org/wiki/Loi_exponentielle)

# Modélisation d'un réseau de comptes aléatoire

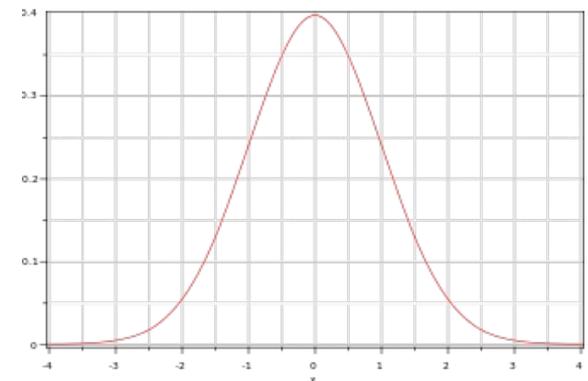
Les comptes de même catégorie de fiabilité, ont plus de chances de s'abonner entre eux



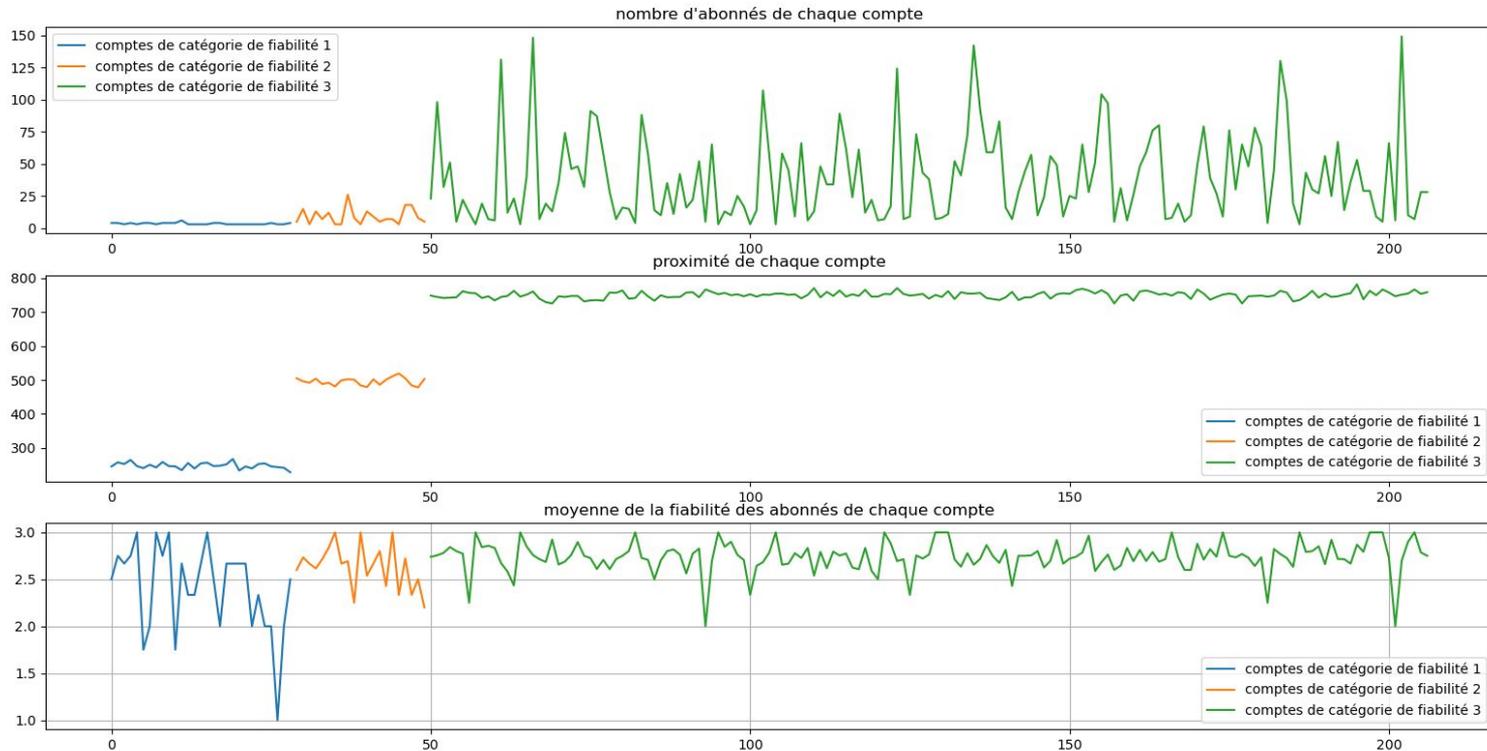
On choisit une distribution gaussienne, avec une moyenne différente en fonction de la catégorie de fiabilité



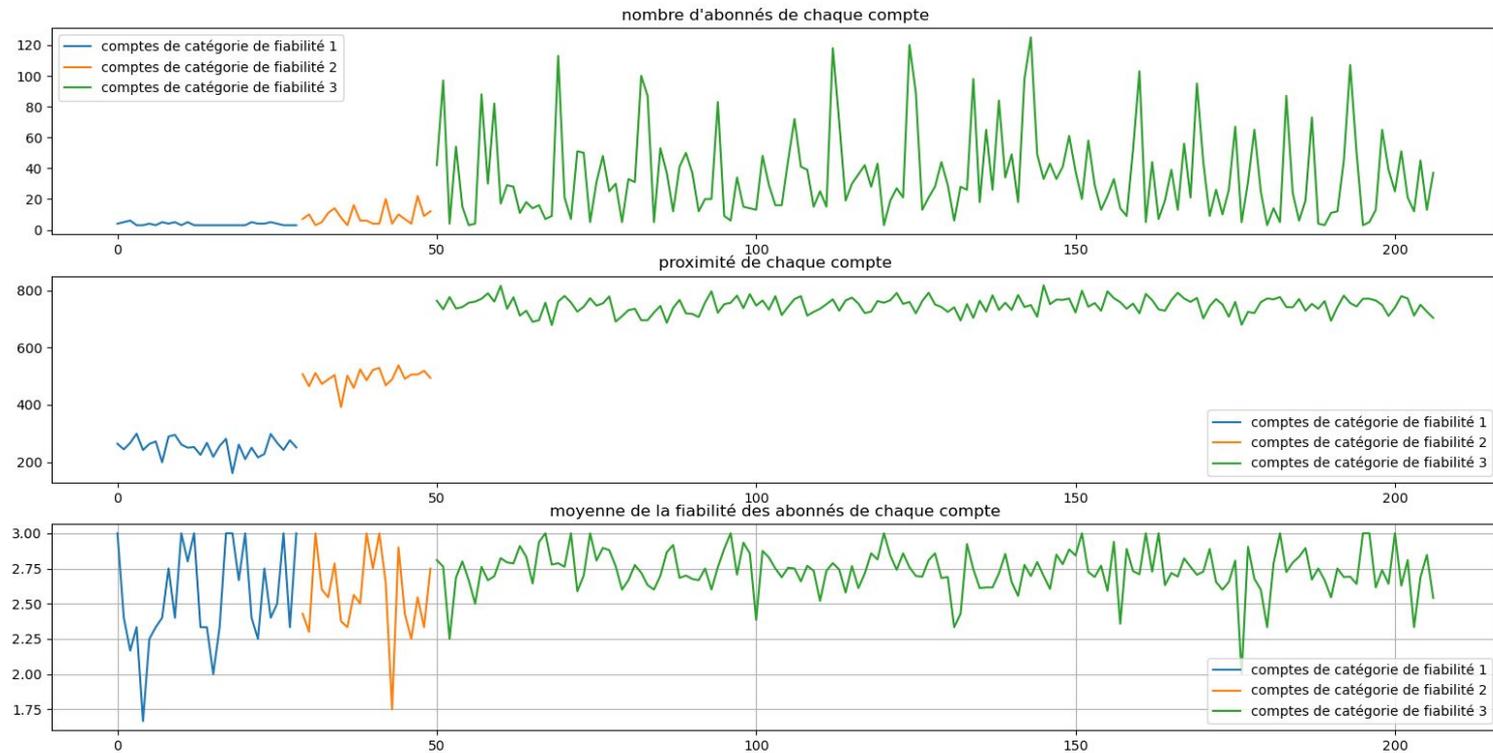
```
# moyenne des proximités des comptes de la catégorie de fiabilité 1/2/3
moyenne_1 = prox_max / 4
moyenne_2 = (2 * prox_max) / 4
moyenne_3 = (3 * prox_max) / 4
for i in range(pop):
    if liste_fiabilite[i] == 1:
        liste_proximite.append(int(random.gauss(moyenne_1, et1)))
    elif liste_fiabilite[i] == 2:
        liste_proximite.append(int(random.gauss(moyenne_2, et2)))
    else:
        liste_proximite.append(int(random.gauss(moyenne_3, et3)))
```



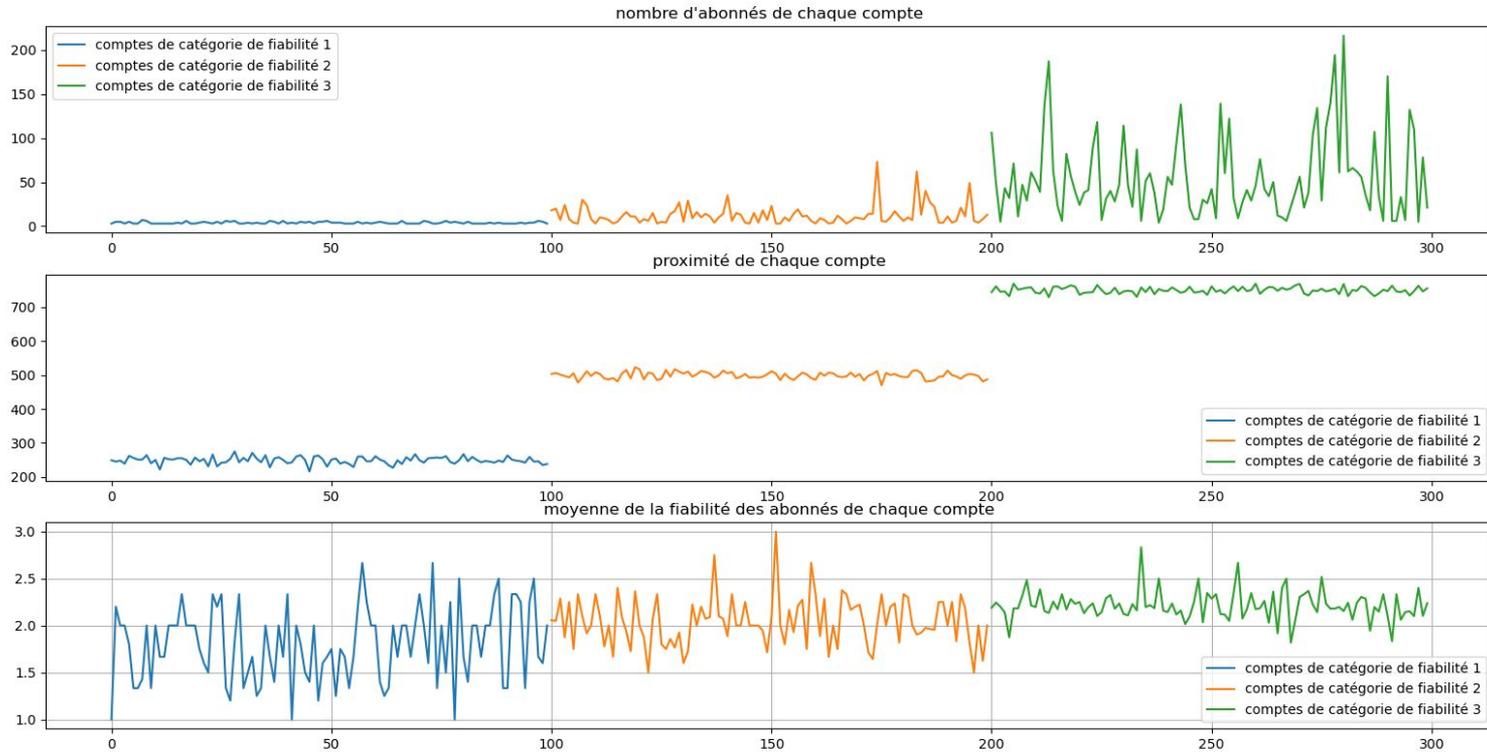
# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes aléatoire



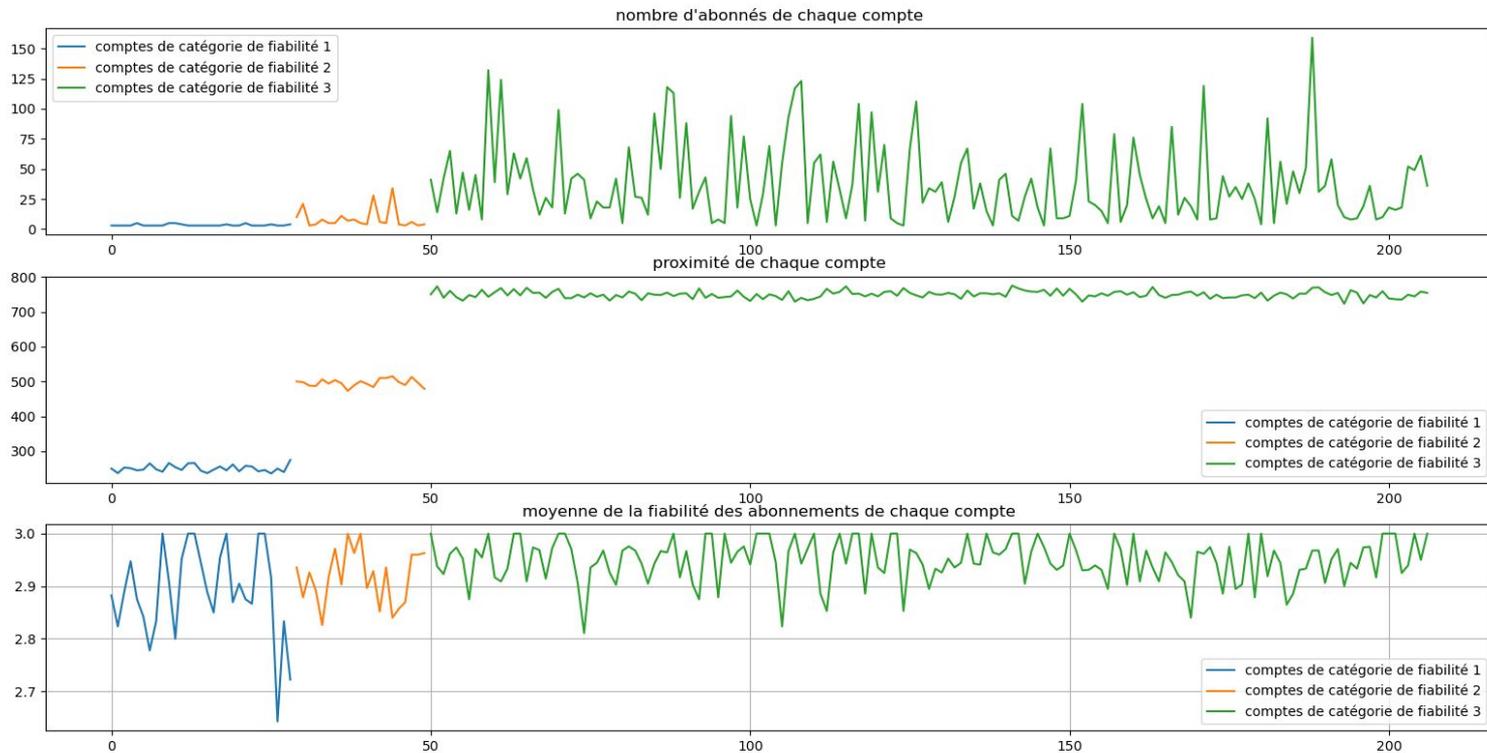
# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes aléatoire

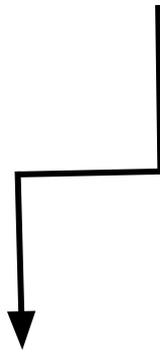


# Modélisation d'un réseau de comptes aléatoire

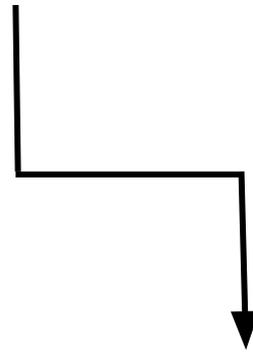


# Modélisation d'un réseau de comptes aléatoire

[[[0, -1, -1], -1, 0], ..., [[0, 0, 1], 1, 1]]

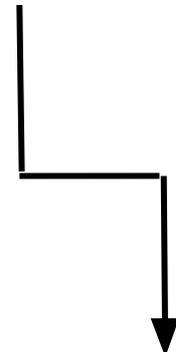


Liste permettant de garder en mémoire les avis de l'utilisateur au cours du temps



Avis de l'utilisateur:

- 1 s'il croit en l'information
- -1 s'i n'y croit pas
- 0 si il est neutre



Partage de l'information:

- 1 s'il l'a partagé
- 0 sinon

# Modélisation d'un réseau de comptes aléatoire

- Tous les comptes ne partagent pas l'information
- Le pourcentage de partage varie en fonction de la catégorie de fiabilité du compte

```
# Le compte retweet-t-il? La probabilité qu'il retweet fluctue en fonction de sa fiabilité
# si la fonction retourne 1, le compte partage l'information
# Dans la simulation 1 cette fonction sert pour le gain d'abonnés car la propagation de l'information se fait uniquement par l'intermédiaire des
abonnés et non du partage de l'information
def retweet_fiabilite(fiabilite):
    retweet = 0
    valeur_possible = []
    for i in range(0, 10001, 1):
        valeur_possible.append(i)
    # si la valeur choisie est compris entre 0 et la probabilité de retweet, retweet = 1
    valeur_choisie = random.choice(valeur_possible)
    if fiabilite == 1:
        if valeur_choisie < 151:
            retweet = 1
    elif fiabilite == 2:
        if valeur_choisie < 46:
            retweet = 1
    elif fiabilite == 3:
        if valeur_choisie < 15:
            retweet = 1
    return retweet
```

# Modélisation d'un réseau de comptes aléatoire

## Partage d'une vraie information:

```
if partage == 1:  
    # 33 % confrontés à une vraie information la croient, 44 % ne pensent pas avoir été influencés, 23 % n'y croient pas  
    # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné  
    influence = ([-1] * 23) + ([0] * 44) + ([1] * 33)  
    influence_ab = random.choice(influence)  
    comptes_[6][i][0].append(influence_ab)
```

Les comptes de catégorie de fiabilité 1 ne partagent pas de vraies informations

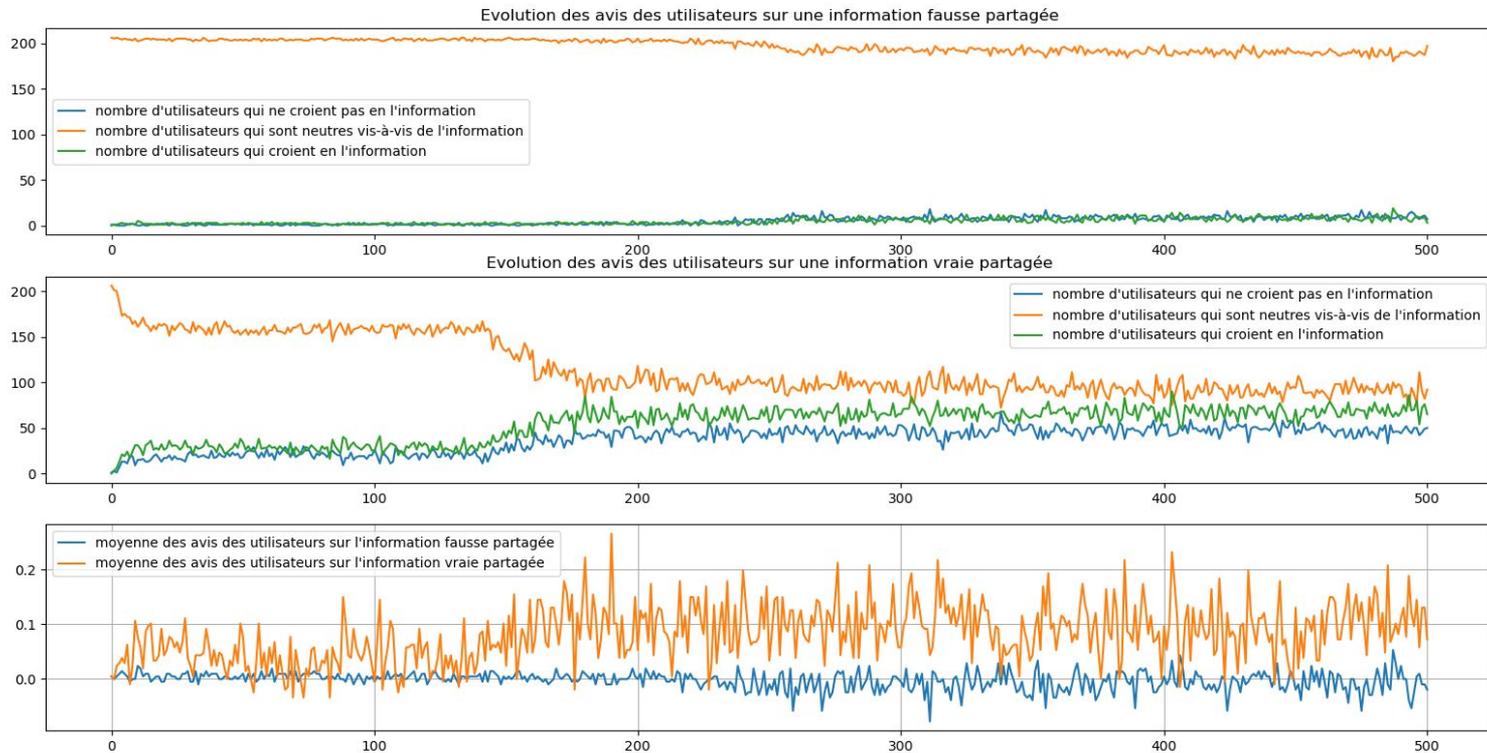
# Modélisation d'un réseau de comptes aléatoire

## Partage d'une fausse information:

```
if partage == 1:  
    # 23 % confrontés à une fake news la croient, 50 % ne pensent pas avoir été influencés, 27 % n'y croient pas  
    # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné  
    influence = ([-1] * 27) + ([0] * 50) + ([1] * 23)  
    influence_ab = random.choice(influence)  
    comptes_[6][i][0].append(influence_ab)
```

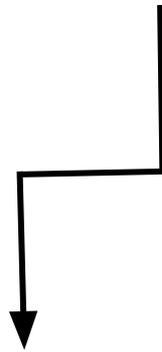
Les comptes de catégorie de fiabilité 3 ne partagent pas de fausses informations

# Modélisation d'un réseau de comptes aléatoire

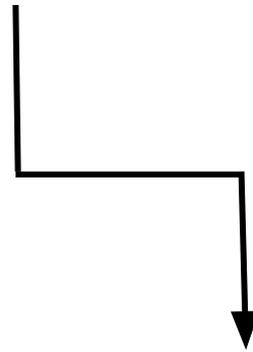


# Modélisation d'un réseau de comptes aléatoire

[[[0, -1, -1], -1, 0], ..., [[0, 0, 1], 1, 1]]

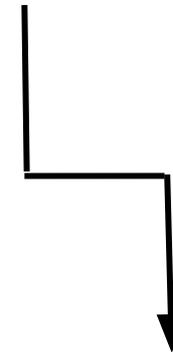


Liste permettant de garder en mémoire les avis de l'utilisateur au cours du temps



Avis de l'utilisateur:

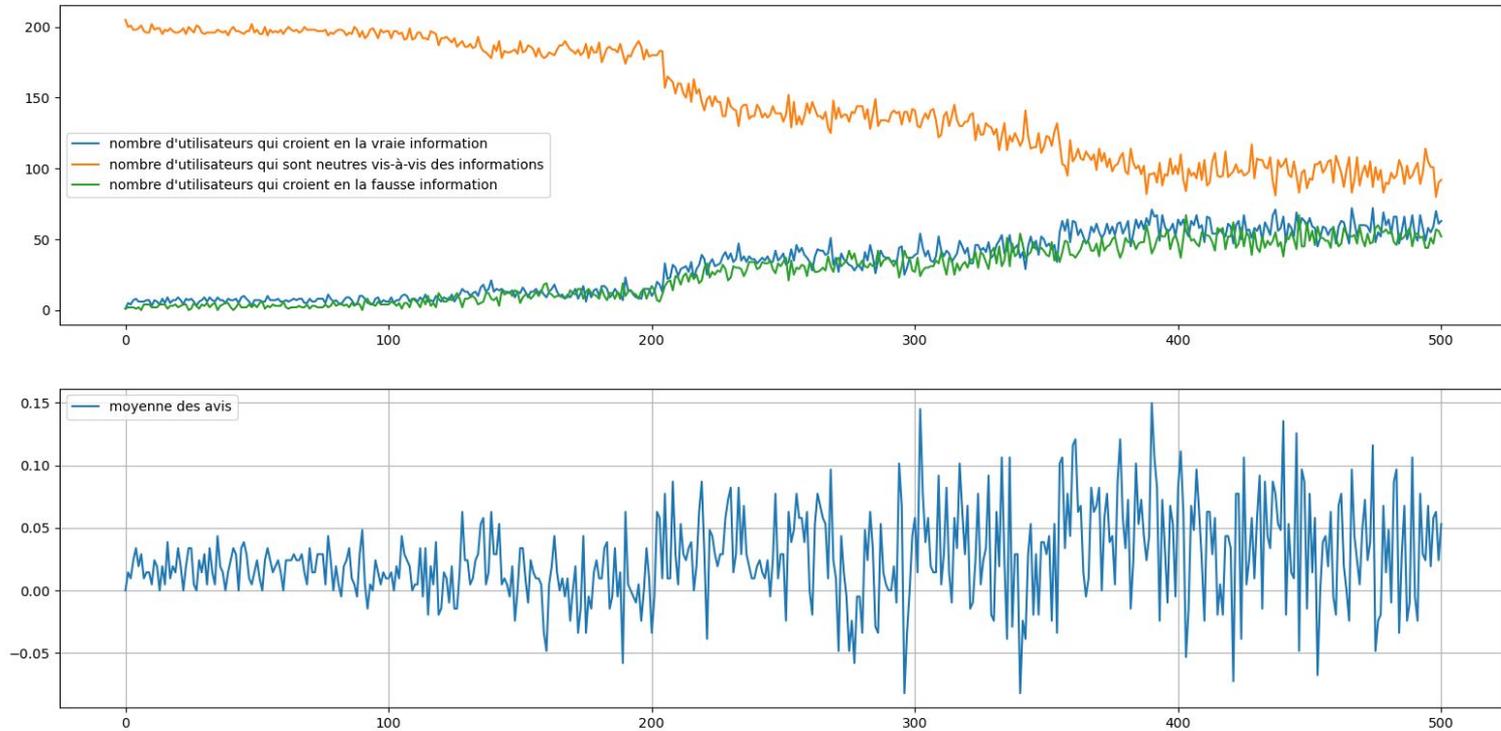
- 1 s'il croit en la vraie information
- -1 s'il croit en la fausse
- 0 s'il est neutre



Partage de l'information:

- 1 s'il l'a partagé
- 0 sinon

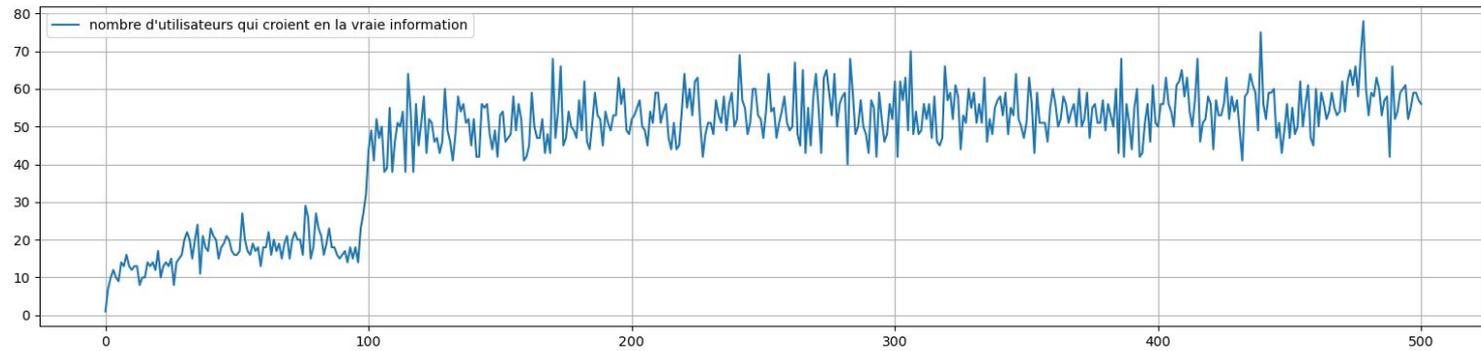
# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes aléatoire

- On simule désormais une campagne de prévention
- Les comptes de catégorie de fiabilité 3 ont maintenant la possibilité de signaler la fausse information
- A partir d'un nombre d'avertissement choisi, on stoppe le partage de la fausse information

# Modélisation d'un réseau de comptes aléatoire



# Modélisation d'un réseau de comptes

## Limites:

- Bien que les caractéristiques des comptes et de la propagation soient inspirées d'études, il est tout de même difficile de proposer un modèle qui correspond à la réalité
- Absence de comparaison possible avec la propagation d'une information réelle
- Le caractère humain est difficile à prendre en compte
- Le modèle donne des résultats incohérents (ex : campagne de prévention)

## ANNEXES

```

import numpy as np
import matplotlib.pyplot as plt
import random

# pop comptes créés avec leur différentes caractéristiques
# comptes créés avec leur différentes caractéristiques
# nb_1 correspond au nombre de comptes de catégorie de fiabilité 1(diffusant des fakes news)
# nb_2 correspond au nombre de comptes de catégorie de fiabilité 2(douteux)
# nb_3 correspond au nombre de comptes de catégorie de fiabilité 3(fiable)
# prox_max désigne la proximité maximale d'un utilisateur. Plus les comptes ont une proximité proche, plus ils ont de chances de s'abonner entre eux. Deux comptes de même catégorie de fiabilité ont plus de chance d'avoir une proximité proche
def creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3):

    liste_nb_abonnes = []
    liste_proximite = []
    liste_public = []
    liste_fiabilite = []
    fiabilite_possible = [1]* nb_1 + [2] * nb_2 + [3] * nb_3
    pop = nb_1 + nb_2 + nb_3
    # On associe à chaque compte une catégorie de fiabilité
    for _ in range(pop):
        liste_fiabilite.append(fiabilite_possible.pop(random.randint(0, len(fiabilite_possible) - 1)))

    liste_num = list(range(pop))

    for _ in range(pop):
        # On associe aléatoirement à chaque compte une nature (public ou privé)
        c = random.randint(0, 1)
        if c > 0:
            e = 10
        else:
            e = 1
        liste_public.append(c)
        # On associe aléatoirement à chaque compte une proximité
        d = random.randint(1, prox_max)
        f = int((d / prox_max) * 10) + 1
        liste_proximite.append(d)
        # On associe à chaque compte un nombre d'abonnés
        liste_nb_abonnes.append(int((1 / random.randint(1, f))*e*random.randint(1, 10)*random.randint(0, int(pop / 2)) / 100) + 1)

```

```

42 # On dissocie le nombre d'abonnés et proximité en fonction des fiabilités pour le tracé des graphiques
43 nb_abo_1 = []
44 nb_abo_2 = []
45 nb_abo_3 = []
46 prox1 = []
47 prox2 = []
48 prox3 = []
49 for i in range(pop):
50     if liste_fiabilite[i] == 1:
51         nb_abo_1.append(liste_nb_abonnes[i])
52         prox1.append(liste_proximite[i])
53     elif liste_fiabilite[i] == 2:
54         nb_abo_2.append(liste_nb_abonnes[i])
55         prox2.append(liste_proximite[i])
56     else:
57         nb_abo_3.append(liste_nb_abonnes[i])
58         prox3.append(liste_proximite[i])
59
60 return [liste_num, liste_nb_abonnes, liste_proximite, liste_public, liste_fiabilite], [nb_abo_1, nb_abo_2, nb_abo_3, prox1, prox2, prox3]

```

```

65 # Le programme d'avant nous a permis de déterminer le nombre d'abonnées de chaque compte. La fonction suivante va désormais déterminer à quels
66 comptes chaque utilisateur est abonné. Plus leurs proximité est proche, plus la probabilité qu'ils s'abonnent entre eux est élevé
67
68 def abonnées_comptes(comptes, prox_max):
69     pop = len(comptes[0])
70     abonnées = []
71     for i in range(pop):
72         abonnées_i = []
73         # Tant que l'utilisateur i n'a pas le bon nombre d'abonnés, on lui cherche des abonnées aléatoirement(en faisant une boucle, les premiers
74         comptes ont plus de chance de s'abonner aux autres)
75         while len(abonnées_i) < comptes[1][i]:
76             j = random.randint(0, pop - 1)
77             # on vérifie que l'utilisateur j n'est pas déjà abonné à l'utilisateur i. S'il est déjà abonné, la variable e prend la valeur 1 et
78             l'utilisateur ne pourra pas s'abonner de nouveau
79             e = 0
80             for abonnée in abonnées_i:
81                 if abonnée == j:
82                     e += 1
83             # Un compte ne peut pas s'abonner à lui-même
84             if j != i:
85                 if e == 0:
86                     # Plus l'utilisateur i a une proximité proche du j, plus le j a de chances de s'abonner au i
87                     a = comptes[2][i] - comptes[2][j]
88                     c = []
89                     for _ in range(abs(a)):
90                         c.append(0)
91                     while len(c) < prox_max:
92                         c.append(1)
93                         id = random.choice(c)
94                         # print(id_abo)
95                     # Si conditions remplies, j est ajouté aux abonnées de i, et i aux abonnements de j
96                     if id > 0:
97                         abonnées_i.append(j)
98     abonnées.append(abonnées_i)
99     return abonnées

```

```

101 def comptes_avec_relation(prox_max, nb_1, nb_2, nb_3):
102     comptes, L2 = creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3)
103     abonnees = abonnees_comptes(comptes, prox_max)
104     comptes.append(abonnees)
105
106     # On cherche désormais à regarder la moyenne de fiabilité des abonnés de chaque compte i en fonction de la fiabilité de i
107     # On utilisera abo pour abonnés
108     moy_fiab_abo_1 = []
109     moy_fiab_abo_2 = []
110     moy_fiab_abo_3 = []
111     pop = len(comptes[0])
112     for i in range(pop):
113         moy_fiab_abo = 0
114         if comptes[4][i] == 1:
115             for abonne in comptes[5][i]:
116                 moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
117             moy_fiab_abo_1.append(moy_fiab_abo)
118         elif comptes[4][i] == 2:
119             for abonne in comptes[5][i]:
120                 moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
121             moy_fiab_abo_2.append(moy_fiab_abo)
122         else:
123             for abonne in comptes[5][i]:
124                 moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
125             moy_fiab_abo_3.append(moy_fiab_abo)
126     L2.append(moy_fiab_abo_1)
127     L2.append(moy_fiab_abo_2)
128     L2.append(moy_fiab_abo_3)
129
130     # Tracé des graphiques
131     plt.subplot(3, 1, 1)
132     plt.plot(list(range(nb_1)), L2[0], label = 'comptes de catégorie de fiabilité 1')
133     plt.legend(loc='upper left')
134     plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[1], label = 'comptes de catégorie de fiabilité 2')
135     plt.legend(loc='upper left')
136     plt.plot(list(range(nb_1 + nb_2, pop)), L2[2], label = 'comptes de catégorie de fiabilité 3')
137     plt.legend(loc='upper left')
138     plt.title('nombre d\'abonnés de chaque compte')
139
140     plt.subplot(3, 1, 2)
141     plt.plot(list(range(nb_1)), L2[3], label = 'fiabilité = 1')
142     plt.legend(loc='upper right')
143     plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[4], label = 'fiabilité = 2')
144     plt.legend(loc='upper right')
145     plt.plot(list(range(nb_1 + nb_2, pop)), L2[5], label = 'fiabilité = 3')
146     plt.legend(loc='upper left')
147     plt.title('proximité de chaque compte')

```

```

149 plt.subplot(3, 1, 3)
150 plt.plot(list(range(nb_1)), L2[6], label = 'comptes de catégorie de fiabilité 1')
151 plt.legend(loc='upper left')
152 plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[7], label = 'comptes de catégorie de fiabilité 2')
153 plt.legend(loc='upper left')
154 plt.plot(list(range(nb_1 + nb_2, pop)), L2[8], label = 'comptes de catégorie de fiabilité 3')
155 plt.legend(loc='lower left')
156 plt.title('moyenne de la fiabilité des abonnés de chaque compte')
157
158 plt.grid()
159 plt.show()
160
161 return comptes, L2
162
163
164 # On initialise la 7ième liste, information, de comptes
165 def creation_comptes_avec_information(comptes):
166     pop = len(comptes[0])
167     # On choisit aléatoirement un compte qui partage une information
168     f = random.randint(0, pop - 1)
169     information = []
170     for i in range(pop):
171         if i != f:
172             information.append(0)
173         else:
174             information.append(1)
175     comptes.append(information)
176     return comptes

```

```

181 # Le compte retweet-t-il? La probabilité qu'il retweet fluctue en fonction de sa fiabilité
182 # si la fonction retourne 1, le compte partage l'information
183 # Dans la simulation 1 cette fonction sert pour le gain d'abonnés car la propagation de l'information se fait uniquement par l'intermédiaire des
abonnés et non du partage de l'information
184 def retweet_fiabilite(fiabilite):
185     retweet = 0
186     valeur_possible = []
187     for i in range(0, 10001, 1):
188         valeur_possible.append(i)
189     # si la valeur choisie est compris entre 0 et la probabilité de retweet, retweet = 1
190     valeur_choisie = random.choice(valeur_possible)
191     if fiabilite == 1:
192         if valeur_choisie < 151:
193             retweet = 1
194     elif fiabilite == 2:
195         if valeur_choisie < 46:
196             retweet = 1
197     elif fiabilite == 3:
198         if valeur_choisie < 15:
199             retweet = 1
200     return retweet
201
202
203 # A chaque tour de boucle dans les simulations, les comptes vont avoir la possibilité de gagner des abonnés
204 # Cette fonction marche de la même manière que la fonction abonnés comptes
205 def gain_abonnees(comptes, prox_max):
206     pop = len(comptes[0])
207     for i in range(pop):
208         for j in range(pop):
209             e = 0
210             for l in comptes[5][i]:
211                 if comptes[0][j] == l:
212                     e += 1
213             if i != j:
214                 if e == 0:
215                     # Les chances que le compte gagne des abonnés varient en fonction de la fiabilité de celui-ci
216                     if retweet_fiabilite(comptes[4][i]) == 1:
217                         a = comptes[2][i] - comptes[2][j]
218                         c = []
219                         for _ in range(abs(a)):
220                             c.append(0)
221                         while len(c) < prox_max:
222                             c.append(1)
223                         id = random.choice(c)
224                         if id > 0:
225                             comptes[5][i].append(j)
226                 comptes[1][i] = len(comptes[5][i])
227     return comptes

```

```

230 # Retourne une liste contenant l'identifiant de tous les comptes ayant l'information
231 def info(comptes):
232     pop = len(comptes[0])
233     inf = []
234     for k in range(pop):
235         if comptes[6][k] == 1:
236             inf.append(k)
237     return inf
238
239
240 # Retourne une liste contenant l'identifiant de tous les comptes qui sont abonnés à un compte ayant l'information
241 def num(comptes, inf):
242     num = []
243     for i in inf:
244         for j in comptes[5][i]:
245             num.append(j)
246     return num
247
248
249
250 # Tous les comptes qui sont abonnés à un compte ayant l'information ont désormais l'information
251 def comptes_rang_5(comptes, num):
252     pop = len(comptes[0])
253     for j in num:
254         comptes[6][j] = 1
255     return comptes
256
257
258
259 # équivalent des 3 dernières fonctions
260 def comptes_info_num(comptes):
261     inf = info(comptes)
262     num_ = num(comptes, inf)
263     return comptes_rang_5(comptes, num_)
264

```

```

266 # Crée un réseau de comptes aléatoire, simule le partage d'une information par un compte, puis permet de suivre l'évolution du nombre de comptes
    qui ont cette information
267 def simulation2(prox_max, nb_1, nb_2, nb_3):
268     comptes, L2 = comptes_avec_relation(prox_max, nb_1, nb_2, nb_3)
269     comptes = creation_comptes_avec_information(comptes)
270     final = []
271     pop = len(comptes[0])
272     for _ in range(pop):
273         final.append(1)
274     a = 1
275     liste_nb_info = [1]
276     while comptes[6] != final:
277         comptes = comptes_info_num(comptes)
278         a += 1
279         inf = 0
280         for i in range(pop):
281             inf += comptes[6][i]
282         liste_nb_info.append(inf)
283         comptes = gain_abonnees(comptes, prox_max)
284
285     plt.plot(list(range(a)), liste_nb_info)
286     plt.title('Evolution du nombre de personnes qui ont l\'information partagée')
287     plt.grid()
288     plt.show()
289
290     return a, liste_nb_info

```

```

294 # Crée un réseau de comptes aléatoire
295 # Simule le partage d'une information par chacun des comptes
296 # Renvoie le nombre moyen d'étapes nécessaires à la propagation de l'information sur le réseau pour chaque catégorie de fiabilité de compte
297 def graph_simulation2(prox_max, nb_1, nb_2, nb_3):
298
299     y_moy = [0, 0, 0]
300     comptes = comptes_avec_relation(prox_max, nb_1, nb_2, nb_3)
301     pop = len(comptes[0])
302     for i in range(pop):
303         comptes_ = list(comptes)
304         information = []
305         for j in range(pop):
306             if j != i:
307                 information.append(0)
308             else:
309                 information.append(1)
310         comptes_.append(information)
311         # temps de propagation
312         e = propagation_information2(comptes_, prox_max)
313         y.append(e)
314         if a == 1:
315             y_moy[0] += e / nb_1
316         elif a == 2:
317             y_moy[1] += e / nb_2
318         else:
319             y_moy[2] += e / nb_3
320     return y_moy

```

```

325 ## Simulation 2
326
327
328 # comptes créés avec leur différentes caractéristiques
329 # nb_1 correspond au nombre de comptes de catégorie de fiabilité 1(diffusant des fakes news)
330 # nb_2 correspond au nombre de comptes de catégorie de fiabilité 2(douteux)
331 # nb_3 correspond au nombre de comptes de catégorie de fiabilité 3(fiable)
332 # prox_max désigne la proximité maximale d'un utilisateur. Plus les comptes ont une proximité proche, plus ils ont de chances de s'abonner entre
    eux. Deux comptes de même catégorie de fiabilité ont plus de chance d'avoir une proximité proche
333
334 def creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3, et1, et2, et3):
335
336     # listes des caractéristiques des comptes
337     liste_nb_abonnes = []
338     liste_proximite = []
339     liste_fiabilite = []
340
341     # on associe à chaque compte une catégorie de fiabilité aléatoire
342     fiabilite_possible = [1] * nb_1 + [2] * nb_2 + [3] * nb_3
343     # La population totale est donc:
344     pop = nb_1 + nb_2 + nb_3
345     for _ in range(pop):
346         liste_fiabilite.append(fiabilite_possible.pop(random.randint(0, len(fiabilite_possible) - 1)))
347
348     # on associe à chaque compte un identifiant
349     liste_id = list(range(pop))
350
351     # on associe à chaque compte une proximité. Les comptes de même catégorie de fiabilité ont plus de chances d'avoir une proximité proche.
352     # On choisit une distribution gaussienne avec une moyenne différente et un écart-type différent en fonction de la catégorie de fiabilité des
    comptes
353     # moyenne des proximités des comptes de la catégorie de fiabilité 1/2/3
354     moyenne_1 = prox_max / 4
355     moyenne_2 = (2 * prox_max) / 4
356     moyenne_3 = (3 * prox_max) / 4
357     for i in range(pop):
358         if liste_fiabilite[i] == 1:
359             liste_proximite.append(int(random.gauss(moyenne_1, et1)))
360         elif liste_fiabilite[i] == 2:
361             liste_proximite.append(int(random.gauss(moyenne_2, et2)))
362         else:
363             liste_proximite.append(int(random.gauss(moyenne_3, et3)))
364

```

```

365 # on associe à chaque compte un nombre d'abonnés initial
366 # On choisit une distribution exponentielle du nombre d'abonnés, avec une moyenne différente en fonction de la catégorie de fiabilité du
compte
367 # les comptes de catégorie 2(resp 3) ont une moyenne d'abonnés 6(resp 36) fois supérieure à ceux de catégorie 1
368 # Le / 200 est choisi pour avoir des nombres moyens d'abonnés acceptables
369 moy_1 = pop / 200
370 moy_2 = moy_1 * 6
371 moy_3 = moy_1 * 36
372 for i in range(pop):
373     if liste_fiabilite[i] == 1:
374         abo = int(random.expovariate(1 / moy_1))
375         while abo > pop * 8 / 10 or 3 > abo:
376             abo = int(random.expovariate(1 / moy_1))
377         liste_nb_abonnes.append(abo)
378     elif liste_fiabilite[i] == 2:
379         abo = int(random.expovariate(1 / moy_2))
380         while abo > pop * 8 / 10 or 3 > abo:
381             abo = int(random.expovariate(1 / moy_2))
382         liste_nb_abonnes.append(abo)
383     else:
384         abo = int(random.expovariate(1 / moy_3))
385         while abo > pop * 8 / 10 or 3 > abo:
386             abo = int(random.expovariate(1 / moy_3))
387         liste_nb_abonnes.append(abo)
388 # On sépare les proximités et nb d'abonnés en fonction de leur fiabilité pour le tracé des courbes
389 nb_abo_1 = []
390 nb_abo_2 = []
391 nb_abo_3 = []
392 prox1 = []
393 prox2 = []
394 prox3 = []
395 for i in range(pop):
396     if liste_fiabilite[i] == 1:
397         nb_abo_1.append(liste_nb_abonnes[i])
398         prox1.append(liste_proximite[i])
399     elif liste_fiabilite[i] == 2:
400         nb_abo_2.append(liste_nb_abonnes[i])
401         prox2.append(liste_proximite[i])
402     else:
403         nb_abo_3.append(liste_nb_abonnes[i])
404         prox3.append(liste_proximite[i])
405 return [liste_id, liste_nb_abonnes, liste_proximite, liste_fiabilite], [nb_abo_1, nb_abo_2, nb_abo_3, prox1, prox2, prox3]
406
407

```

```

409 # Le programme d'avant nous a permis de déterminer le nombre d'abonnés de chaque compte. La fonction suivante va désormais déterminer à quels
comptes chaque utilisateur est abonné. Plus leur proximité est proche, plus la probabilité qu'ils s'abonnent entre eux est élevée
410
411 def abonnees_comptes(comptes, prox_max):
412     pop = len(comptes[0])
413     abonnees = []
414     abonnements = []
415     for _ in range(pop):
416         abonnements.append([])
417     for i in range(pop):
418         abonnees_i = []
419         # Tant que l'utilisateur i n'a pas le bon nombre d'abonnés, on lui cherche des abonnés aléatoirement(en faisant une boucle, les premiers
comptes ont plus de chances de s'abonner aux autres)
420         while len(abonnees_i) < comptes[1][i]:
421             j = random.randint(0, pop - 1)
422             # on vérifie que l'utilisateur j n'est pas déjà abonné à l'utilisateur i. S'il est déjà abonné, la variable e prend la valeur 1 et
l'utilisateur ne pourra pas s'abonner de nouveau
423             e = 0
424             for abonnee in abonnees_i:
425                 if abonnee == j:
426                     e += 1
427             # Un compte ne peut pas s'abonner à lui-même
428             if j != i:
429                 if e == 0:
430                     # Plus l'utilisateur i a une proximité proche du j, plus le j a de chances de s'abonner au i
431                     a = comptes[2][i] - comptes[2][j]
432                     c = []
433                     for _ in range(abs(a)):
434                         c.append(0)
435                     while len(c) < prox_max:
436                         c.append(1)
437                     id = random.choice(c)
438                     # print(id_abo)
439                     # si conditions remplies, j est ajouté aux abonnées de i, et i aux abonnements de j
440                     if id > 0:
441                         abonnees_i.append(j)
442             abonnees.append(abonnees_i)
443     for i in range(pop):
444         for abonnee in abonnees[i]:
445             abonnements[abonnee].append(i)
446
447     return abonnees, abonnements

```

```

451 # Crée un réseau de comptes où les comptes ont des abonnés
452 def comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3):
453     comptes, L2 = creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
454     abonnees, abonnements = abonnes_comptes(comptes, prox_max)
455     comptes.append(abonnees)
456     comptes.append(abonnements)
457
458     # On cherche désormais à regarder la moyenne de fiabilité des abonnés de chaque compte i en fonction de la fiabilité de i
459     # On utilisera abo pour abonnés
460     moy_fiab_abo_1 = []
461     moy_fiab_abo_2 = []
462     moy_fiab_abo_3 = []
463     pop = len(comptes[0])
464     for i in range(pop):
465         moy_fiab_abo = 0
466         if comptes[3][i] == 1:
467             for abonne in comptes[4][i]:
468                 moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
469             moy_fiab_abo_1.append(moy_fiab_abo)
470         elif comptes[3][i] == 2:
471             for abonne in comptes[4][i]:
472                 moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
473             moy_fiab_abo_2.append(moy_fiab_abo)
474         else:
475             for abonne in comptes[4][i]:
476                 moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
477             moy_fiab_abo_3.append(moy_fiab_abo)
478     L2.append(moy_fiab_abo_1)
479     L2.append(moy_fiab_abo_2)
480     L2.append(moy_fiab_abo_3)

```

```

482 # On cherche désormais à regarder la moyenne de fiabilité des abonnements de chaque compte i en fonction de la fiabilité de i
483 # On utilisera ab pour abonnements
484
485 moy_fiab_ab_1 = []
486 moy_fiab_ab_2 = []
487 moy_fiab_ab_3 = []
488 pop = len(comptes[0])
489 for i in range(pop):
490     moy_fiab_ab = 0
491     if comptes[3][i] == 1:
492         for abonnement in comptes[5][i]:
493             moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
494         moy_fiab_ab_1.append(moy_fiab_ab)
495     elif comptes[3][i] == 2:
496         for abonnement in comptes[5][i]:
497             moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
498         moy_fiab_ab_2.append(moy_fiab_ab)
499     else:
500         for abonnement in comptes[5][i]:
501             moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
502         moy_fiab_ab_3.append(moy_fiab_ab)
503 L2.append(moy_fiab_ab_1)
504 L2.append(moy_fiab_ab_2)
505 L2.append(moy_fiab_ab_3)
506
507
508 # plt.subplot(3, 1, 1)
509 # plt.plot(list(range(nb_1)), L2[0], label = 'comptes de catégorie de fiabilité 1')
510 # plt.legend(loc='upper left')
511 # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[1], label = 'comptes de catégorie de fiabilité 2')
512 # plt.legend(loc='upper left')
513 # plt.plot(list(range(nb_1 + nb_2, pop)), L2[2], label = 'comptes de catégorie de fiabilité 3')
514 # plt.legend(loc='upper left')
515 # plt.title('nombre d\'abonnés de chaque compte')
516 #
517 # plt.subplot(3, 1, 2)
518 # plt.plot(list(range(nb_1)), L2[3], label = 'comptes de catégorie de fiabilité 1')
519 # plt.legend(loc='upper right')
520 # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[4], label = 'comptes de catégorie de fiabilité 2')
521 # plt.legend(loc='upper right')
522 # plt.plot(list(range(nb_1 + nb_2, pop)), L2[5], label = 'comptes de catégorie de fiabilité 3')
523 # plt.legend(loc='lower right')
524 # plt.title('proximité de chaque compte')

```

```

526 # plt.subplot(3, 1, 3)
527 # plt.plot(list(range(nb_1)), L2[6], label = 'comptes de catégorie de fiabilité 1')
528 # plt.legend(loc='upper left')
529 # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[7], label = 'comptes de catégorie de fiabilité 2')
530 # plt.legend(loc='upper left')
531 # plt.plot(list(range(nb_1 + nb_2, pop)), L2[8], label = 'comptes de catégorie de fiabilité 3')
532 # plt.legend(loc='lower right')
533 # plt.title('moyenne de la fiabilité des abonnés de chaque compte')
534
535 # plt.subplot(3, 1, 3)
536 # plt.plot(list(range(nb_1)), L2[9], label = 'comptes de catégorie de fiabilité 1')
537 # plt.legend(loc='upper left')
538 # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[10], label = 'comptes de catégorie de fiabilité 2')
539 # plt.legend(loc='upper left')
540 # plt.plot(list(range(nb_1 + nb_2, pop)), L2[11], label = 'comptes de catégorie de fiabilité 3')
541 # plt.legend(loc='lower right')
542 # plt.title('moyenne de la fiabilité des abonnements de chaque compte')
543
544 plt.grid()
545 plt.show()
546
547 return comptes, L2
548
549
550 # A chaque étape les utilisateurs ont la possibilité de gagner des abonnés
551
552 def gain_abonnees(comptes, prox_max):
553     pop = len(comptes[0])
554     for i in range(pop):
555         for j in range(pop):
556             e = 0
557             for l in comptes[4][i]:
558                 if comptes[0][j] == l:
559                     e += 1
560             if i != j:
561                 if e == 0:
562                     a = comptes[2][i] - comptes[2][j]
563                     c = []
564                     for _ in range(abs(a)):
565                         c.append(0)
566                     while len(c) < prox_max:
567                         c.append(1)
568                     id = random.choice(c)
569                     if id > 0:
570                         comptes[4][i].append(id)
571             comptes[1][i] = len(comptes[4][i])
572     return comptes

```

```

576 # On ajoute une liste informations au réseau. Chaque compte a alors un avis sur l'information que partage chaque compte : 0 pour neutre, -1 si
l'utilisateur ne croit pas en l'information, 1 si l'utilisateur croit en l'information. On initialise ici les avis : chaque compte a un avis
neutre par rapport aux informations que partage les autres comptes. Chaque compte a également la possibilité de partager ou non l'information des
autres compte : 0 s'il ne la partage pas, et 1 sinon
577 # On ne peut partager que si avis == 1
578 # On considère que les comptes de catégorie de fiabilité 1, ne partagent que des vraies informations
579 # On considère que les comptes de catégorie de fiabilité 2, partagent tout type d'information
580 # On considère que les comptes de catégorie de fiabilité 3, ne partagent que de fausses informations
581
582 def comptes_avec_information(comptes):
583     pop = len(comptes[0])
584     information = []
585     for _ in range(pop):
586         # la liste de gauche permet d'avoir en mémoire les avis de l'utilisateur au cours du temps. 0 du centre initialise l'avis de
l'utilisateur, et le zéro de droite initialise le partage
587         information.append([[0], 0, 0])
588     comptes.append(information)
589     return comptes

```

```

592     # on effectue une copie élément par élément de comptes car la fonction bug sinon
593
594 def copie_comptes(comptes):
595     pop = len(comptes[0])
596     comptes_ = []
597     for i in range(7):
598         copie_compte = []
599         if i == 4:
600             abonne = []
601             for j in range(len(comptes[4])):
602                 abonne_j = []
603                 for k in range(len(comptes[4][j])):
604                     abonne_j.append(comptes[4][j][k])
605                 abonne.append(abonne_j)
606             comptes_.append(abonne)
607         elif i == 5:
608             abonnement = []
609             for j in range(len(comptes[5])):
610                 abonnement_j = []
611                 for k in range(len(comptes[5][j])):
612                     abonnement_j.append(comptes[5][j][k])
613                 abonnement.append(abonnement_j)
614             comptes_.append(abonnement)
615         elif i == 6:
616             inf = []
617             for j in range(len(comptes[6])):
618                 inf_j = []
619                 for k in range(len(comptes[6][j])):
620                     if k == 0:
621                         liste_avis = []
622                         for l in range(len(comptes[6][j][0])):
623                             liste_avis.append(comptes[6][j][0][l])
624                         inf_j.append(liste_avis)
625                     else:
626                         inf_j.append(comptes[6][j][k])
627                 inf.append(inf_j)
628             comptes_.append(inf)
629         else:
630             for j in range(pop):
631                 copie_compte.append(comptes[i][j])
632             comptes_.append(copie_compte)
633     return comptes_

```

```

638 # Méthode de partage d'une fausse information
639 def partage_fausse_information(comptes):
640
641 #     # On copie le compte pour ne pas influencer la propagation de l'information
642     fiabilite = comptes[3]
643     abonnes = comptes[4]
644     abonnements = comptes[5]
645     informations = comptes[6]
646     pop = len(comptes[0])
647     comptes_ = copie_comptes(comptes)
648
649 # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
650 for i in range(pop):
651     fiab = fiabilite[i]
652     # on regarde si les comptes auxquels l'utilisateur i est abonné ont partagé l'information
653     for ab in abonnements[i]:
654         partage = informations[ab][2]
655         if partage == 1:
656             # 23 % confrontés à une fake news la croient, 50 % ne pensent pas avoir été influencés, 27 % n'y croient pas
657             # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
658             influence = ([-1] * 27) + ([0] * 50) + ([1] * 23)
659             influence_ab = random.choice(influence)
660             comptes_[6][i][0].append(influence_ab)
661     comptes_[6][i][1] = random.choice(comptes_[6][i][0])
662     # si l'utilisateur i croit désormais en l'information, il va avoir la possibilité de la partager
663     if comptes_[6][i][1] == 1:
664         if comptes_[6][i][2] != 1:
665             # les comptes fiables ne partagent pas de fausses informations, les autres peuvent le faire
666             if fiab != 3:
667                 comptes_[6][i][2] = retweet_fiabilite(fiab)
668             else:
669                 comptes_[6][i][2]
670
671
672 return comptes_

```

```

676 # Méthode de partage d'une vraie information
677 def partage_vraie_information(comptes):
678
679     # On copie le compte pour ne pas influencer la propagation de l'information
680     fiabilite = comptes[3]
681     abonnes = comptes[4]
682     abonnements = comptes[5]
683     informations = comptes[6]
684     pop = len(comptes[0])
685     comptes_ = copie_comptes(comptes)
686
687     # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
688     for i in range(pop):
689         fiab = fiabilite[i]
690         # on regarde si les comptes auxquels l'utilisateur i est abonné ont partagé l'information
691         for ab in abonnements[i]:
692             partage = informations[ab][2]
693             if partage == 1:
694                 # 33 % confrontés à une vraie information la croient, 44 % ne pensent pas avoir été influencés, 23 % n'y croient pas
695                 # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
696                 influence = ([-1] * 23) + ([0] * 44) + ([1] * 33)
697                 influence_ab = random.choice(influence)
698                 comptes_[6][i][0].append(influence_ab)
699                 comptes_[6][i][1] = random.choice(comptes_[6][i][0])
700             # si l'utilisateur i croit désormais en l'information, il va avoir la possibilité de la partager
701             if comptes_[6][i][1] == 1:
702                 if comptes_[6][i][2] != 1:
703                     # les comptes diffusant des fake news ne partagent pas les vraies informations, les autres peuvent le faire
704                     if fiab != 1:
705                         comptes_[6][i][2] = retweet_fiabilite(fiab)
706                     else:
707                         comptes_[6][i][2] = 0
708
709     return comptes

```

```

712 # On simule la propagation d'une information vraie et d'une information fausse sur n étapes
713 def propagation_information2(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n):
714     comptes, L2 = comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
715     comptes = comptes_avec_information(comptes)
716     pop = len(comptes[0])
717     comptes_ = copie_comptes(comptes)
718     # Listes pour les tracés de graphiques
719     croit_pas_vraie = [0]
720     neutre_vraie = [pop - 1]
721     croit_vraie = [1]
722     moyenne_vraie = [1 / pop]
723     croit_pas_fausse = [0]
724     neutre_fausse = [pop - 1]
725     croit_fausse = [1]
726     moyenne_fausse = [1 / pop]
727
728     # On effectue simulation de la propagation d'une vraie information
729     # On cherche un compte de fiabilité 3 qui va partager une vraie information
730     j_1 = random.randint(0, pop - 1)
731     while comptes[3][j_1] != 3:
732         j_1 = random.randint(0, pop - 1)
733     comptes[6][j_1][0] = [1]
734     comptes[6][j_1][1] = 1
735     comptes[6][j_1][2] = 1
736
737     for _ in range(n):
738         # comptes_ = gain_abonnees(comptes_)
739         comptes = partage_vraie_information(comptes)
740         croit_pas = 0
741         croit = 0
742         neutre = 0
743         for i in range(pop):
744             if comptes[6][i][1] == -1:
745                 croit_pas += 1
746             elif comptes[6][i][1] == 0:
747                 neutre += 1
748             else:
749                 croit += 1
750         croit_pas_vraie.append(croit_pas)
751         neutre_vraie.append(neutre)
752         croit_vraie.append(croit)
753         moyenne_vraie.append((-croit_pas + croit)/pop)

```

```

755 # On effectue simulation de la propagation d'une fausse information
756 # On cherche un compte de fiabilité 1 qui va partager une vraie information
757 j_2 = random.randint(0, pop - 1)
758 while comptes_[3][j_2] != 1:
759     j_2 = random.randint(0, pop - 1)
760 comptes_[6][j_2][0] = [1]
761 comptes_[6][j_2][1] = 1
762 comptes_[6][j_2][2] = 1
763
764 for _ in range(n):
765     # comptes_ = gain_abonnees(comptes)
766     comptes_ = partage_fausse_information(comptes_)
767     croit_pas = 0
768     croit = 0
769     neutre = 0
770     for i in range(pop):
771         if comptes_[6][i][1] == -1:
772             croit_pas += 1
773         elif comptes_[6][i][1] == 0:
774             neutre += 1
775         else:
776             croit += 1
777     croit_pas_fausse.append(croit_pas)
778     neutre_fausse.append(neutre)
779     croit_fausse.append(croit)
780     moyenne_fausse.append((-croit_pas + croit)/pop)
781
782 plt.subplot(3, 1, 1)
783 plt.plot(list(range(n + 1)), croit_pas_fausse, label = 'nombre d\'utilisateurs qui ne croient pas en l\'information')
784 plt.legend(loc='center left')
785 plt.plot(list(range(n + 1)), neutre_fausse, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis de l\'information')
786 plt.legend(loc='center left')
787 plt.plot(list(range(n + 1)), croit_fausse, label = 'nombre d\'utilisateurs qui croient en l\'information')
788 plt.legend(loc='center left')
789 plt.title('Evolution des avis des utilisateurs sur une information faussée')
790
791 plt.subplot(3, 1, 2)
792 plt.plot(list(range(n + 1)), croit_pas_vraie, label = 'nombre d\'utilisateurs qui ne croient pas en l\'information')
793 plt.legend(loc='upper right')
794 plt.plot(list(range(n + 1)), neutre_vraie, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis de l\'information')
795 plt.legend(loc='upper right')
796 plt.plot(list(range(n + 1)), croit_vraie, label = 'nombre d\'utilisateurs qui croient en l\'information')
797 plt.legend(loc='upper right')
798 plt.title('Evolution des avis des utilisateurs sur une information vraie partagée')

```

```

800 plt.subplot(3, 1, 3)
801 plt.plot(list(range(n + 1)), moyenne_fausse, label = 'moyenne des avis des utilisateurs sur l\'information fausse partagée')
802 plt.legend(loc='upper left')
803 plt.plot(list(range(n + 1)), moyenne_vraie, label = 'moyenne des avis des utilisateurs sur l\'information vraie partagée')
804 plt.legend(loc='upper left')
805
806 plt.grid()
807 plt.show()
808
809
810
811 def partage_information(comptes):
812
813     # On copie le compte pour ne pas influencer la propagation de l'information
814     fiabilite = comptes[3]
815     abonnees = comptes[4]
816     abonnements = comptes[5]
817     informations = comptes[6]
818     pop = len(comptes[0])
819
820     comptes_ = copie_comptes(comptes)
821
822     # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
823     for i in range(pop):
824         fiab = fiabilite[i]
825         # on regarde si les comptes auxquels l'utilisateur i est abonné ont partagé l'information
826
827         for ab in abonnements[i]:
828             partage = informations[ab][2]
829             if partage == 1:
830                 if informations[ab][1] == 1:
831                     # 33 % confrontés à une vraie information la croient, 44 % ne pensent pas avoir été influencés, 23 % n'y croient pas
832                     # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
833                     influence = ([-1] * 23) + ([0] * 44) + ([1] * 33)
834                     influence_ab = random.choice(influence)
835                     comptes_[6][i][0].append(influence_ab)
836                 elif informations[ab][1] == -1:
837                     # 23 % confrontés à une fake news la croient, 50 % ne pensent pas avoir été influencés, 27 % n'y croient pas
838                     # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
839                     influence = ([-1] * 27) + ([0] * 50) + ([1] * 23)
840                     influence_ab = random.choice(influence)
841                     comptes_[6][i][0].append(influence_ab)
842             comptes_[6][i][1] = random.choice(comptes_[6][i][0])

```

```
843     if fiab == 3:
844         if comptes_[6][i][1] == 1:
845             if comptes_[6][i][2] != 1:
846                 comptes_[6][i][2] = retweet_fiabilite(fiab)
847             else:
848                 comptes_[6][i][2] == 0
849     elif fiab == 1:
850         if comptes_[6][i][1] == -1:
851             if comptes_[6][i][2] != 1:
852                 comptes_[6][i][2] = retweet_fiabilite(fiab)
853             else:
854                 comptes_[6][i][2] == 0
855     else:
856         if comptes_[6][i][1] == 0:
857             comptes_[6][i][2] == 0
858         else:
859             if comptes_[6][i][2] != 1:
860                 comptes_[6][i][2] = retweet_fiabilite(fiab)
861
862     return comptes_
```

```

865 def propagation_information2_2(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n):
866     comptes, L2 = comptes_avec_abonnees(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
867     comptes = comptes_avec_information(comptes)
868     pop = len(comptes[0])
869
870     # Listes pour les tracés de graphiques
871     croit_en_vraie = [1]
872     neutre = [pop - 2]
873     croit_en_faux = [1]
874     moyenne = [0]
875
876     # On effectue simulation de la propagation d'une vraie information
877     # On cherche un compte de fiabilité 3 qui va partager une vraie information
878     j_1 = random.randint(0, pop - 1)
879     while comptes[3][j_1] != 3:
880         j_1 = random.randint(0, pop - 1)
881         # print(comptes[3][j_1])
882     comptes[6][j_1][0] = [1]
883     comptes[6][j_1][1] = 1
884     comptes[6][j_1][2] = 1
885
886     j_2 = random.randint(0, pop - 1)
887     while comptes[3][j_2] != 1:
888         j_2 = random.randint(0, pop - 1)
889     comptes[6][j_2][0] = [1]
890     comptes[6][j_2][1] = 1
891     comptes[6][j_2][2] = 1
892
893     for i in range(n):
894         # comptes = gain_abonnees(comptes, prox_max)
895         comptes = partage_information(comptes)
896         croit_vraie = 0
897         croit_faux = 0
898         croit_neutre = 0
899         for i in range(pop):
900             if comptes[6][i][1] == -1:
901                 croit_faux += 1
902             elif comptes[6][i][1] == 0:
903                 croit_neutre += 1
904             else:
905                 croit_vraie += 1
906         croit_en_vraie.append(croit_vraie)
907         neutre.append(croit_neutre)
908         croit_en_faux.append(croit_faux)
909         moyenne.append((-croit_faux + croit_vraie) / pop)

```

```
912 plt.subplot(2, 1, 1)
913 plt.plot(list(range(n + 1)), croit_en_vraie, label = 'nombre d\'utilisateurs qui croient en la vraie information')
914 plt.legend(loc='center left')
915 plt.plot(list(range(n + 1)), neutre, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis des informations')
916 plt.legend(loc='center left')
917 plt.plot(list(range(n + 1)), croit_en_faux, label = 'nombre d\'utilisateurs qui croient en la fausse information')
918 plt.legend(loc='center left')
919
920 plt.subplot(2, 1, 2)
921 plt.plot(list(range(n + 1)), moyenne, label = 'moyenne des avis')
922 plt.legend(loc='upper left')
923
924
925 plt.grid()
926 plt.show()
```

```

929 def partage_information_campagne_prevention(comptes, signalements, signalement_necessaire):
930
931     # On copie le compte pour ne pas influencer la propagation de l'information
932     fiabilite = comptes[3]
933     abonnes = comptes[4]
934     abonnements = comptes[5]
935     informations = comptes[6]
936     pop = len(comptes[0])
937     comptes_ = copie_comptes(comptes)
938     liste_signalements = []
939     for i in range(pop):
940         liste_signalements.append(comptes[7][i])
941     comptes_.append(liste_signalements)
942
943     # Si l'information fausse a été trop de fois signalée, on stoppe son partage
944     if signalements >= signalement_necessaire:
945         for i in range(pop):
946             if comptes[6][i][1] == -1:
947                 comptes[6][i][2] == 0
948
949     # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
950     for i in range(pop):
951         fiab = fiabilite[i]
952         # on regarde si les comptes auxquels l'utilisateur i est abonné ont partagé l'information
953         for ab in abonnements[i]:
954             partage = informations[ab][2]
955             if partage == 1:
956                 if informations[ab][1] == 1:
957                     # 33 % confrontés à une vraie information la croient, 44 % ne pensent pas avoir été influencés, 23 % n'y croient pas
958                     # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
959                     influence = ([-1] * 23) + ([0] * 44) + ([1] * 33)
960                     influence_ab = random.choice(influence)
961                     comptes_[6][i][0].append(influence_ab)
962                 elif informations[ab][1] == -1:
963                     # 23 % confrontés à une fake news la croient, 50 % ne pensent pas avoir été influencés, 27 % n'y croient pas
964                     # On va ajouter un avis à la liste des avis de l'utilisateur i, à partir de l'influence du compte auquel il est abonné
965                     influence = ([-1] * 27) + ([0] * 50) + ([1] * 23)
966                     influence_ab = random.choice(influence)
967                     comptes_[6][i][0].append(influence_ab)
968     comptes [6][i][1] = random.choice(comptes [6][i][0])

```

```

969     if fiab == 3:
970         if comptes_[6][i][1] == 1:
971             if comptes_[6][i][2] != 1:
972                 comptes_[6][i][2] = retweet_fiabilite(fiab)
973             else:
974                 comptes_[6][i][2] == 0
975     elif fiab == 1:
976         if comptes_[6][i][1] == -1:
977             if comptes_[6][i][2] != 1:
978                 comptes_[6][i][2] = retweet_fiabilite(fiab)
979             else:
980                 comptes_[6][i][2] == 0
981     else:
982         if comptes_[6][i][1] == 0:
983             comptes_[6][i][2] == 0
984         else:
985             if comptes_[6][i][2] != 1:
986                 comptes_[6][i][2] = retweet_fiabilite(fiab)
987 for i in range(pop):
988     if comptes_[3][i] == 3 and comptes_[6][i][1] == 1:
989         comptes_[7][i] = 1
990
991 return comptes_

```

```

994 def propagation_information2_2_prevention(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n, signalement_necessaire):
995     comptes, L2 = comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
996     comptes = comptes_avec_information(comptes)
997     pop = len(comptes[0])
998     liste_signalements = []
999     for _ in range(pop):
1000         liste_signalements.append(0)
1001     comptes.append(liste_signalements)
1002
1003     # Listes pour les tracés de graphiques
1004     croit_en_vraie = [1]
1005     neutre = [pop - 2]
1006     croit_en_faux = [1]
1007     moyenne = [0]
1008
1009     # On effectue simulation de la propagation d'une vraie information
1010     # On cherche un compte de fiabilité 3 qui va partager une vraie information
1011     j_1 = random.randint(0, pop - 1)
1012     while comptes[3][j_1] != 3:
1013         j_1 = random.randint(0, pop - 1)
1014         # print(comptes[3][j_1])
1015     comptes[6][j_1][0] = [1]
1016     comptes[6][j_1][1] = 1
1017     comptes[6][j_1][2] = 1

```

```

1019 j_2 = random.randint(0, pop - 1)
1020 while comptes[3][j_2] != 1:
1021     j_2 = random.randint(0, pop - 1)
1022 comptes[6][j_2][0] = 1
1023 comptes[6][j_2][1] = 1
1024 comptes[6][j_2][2] = 1
1025 signalements = 0
1026 a = 0
1027 for _ in range(n):
1028     # comptes = gain_abonnees(comptes, prox_max)
1029     comptes = partage_information_campagne_prevention(comptes, signalements, signalement_necessaire)
1030     signalements = 0
1031     croit_vraie = 0
1032     croit_faux = 0
1033     croit_neutre = 0
1034     for i in range(pop):
1035         signalements += comptes[7][i]
1036         if comptes[6][i][1] == -1:
1037             croit_faux += 1
1038         elif comptes[6][i][1] == 0:
1039             croit_neutre += 1
1040         else:
1041             croit_vraie += 1
1042     croit_en_vraie.append(croit_vraie)
1043     neutre.append(croit_neutre)
1044     croit_en_faux.append(croit_faux)
1045     moyenne.append((-croit_faux + croit_vraie) / pop)
1046     a += 1
1047     if signalements < signalement_necessaire:
1048         print(a)
1049
1050
1051 plt.subplot(2, 1, 1)
1052 plt.plot(list(range(n + 1)), croit_en_vraie, label = 'nombre d\'utilisateurs qui croient en la vraie information')
1053 plt.legend(loc='upper left')
1054 # plt.plot(list(range(n + 1)), neutre, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis des informations')
1055 # plt.legend(loc='center left')
1056 # plt.plot(list(range(n + 1)), croit_en_faux, label = 'nombre d\'utilisateurs qui croient en la fausse information')
1057 # plt.legend(loc='center left')
1058 #
1059 # plt.subplot(2, 1, 2)
1060 # plt.plot(list(range(n + 1)), moyenne, label = 'moyenne des avis')
1061 # plt.legend(loc='upper left')
1062
1063 plt.grid()
1064 plt.show()

```