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

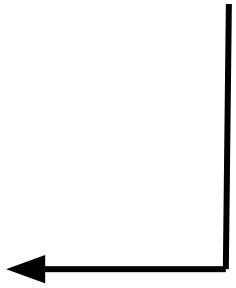
[liste_id, liste_nombre_abonnés, liste_proximité, liste_public, liste_fiabilité]

Liste de taille n
Le ième élément donne le nombre d’abonnés du ième compte

Liste de taille n
Le ième élément donne la nature du ième 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 prox_max

- Fiabilité : 1, 2, ou 3

- Nombre d’abonnés : entier aléatoire inférieur à n

Modélisation d'un réseau de comptes aléatoire

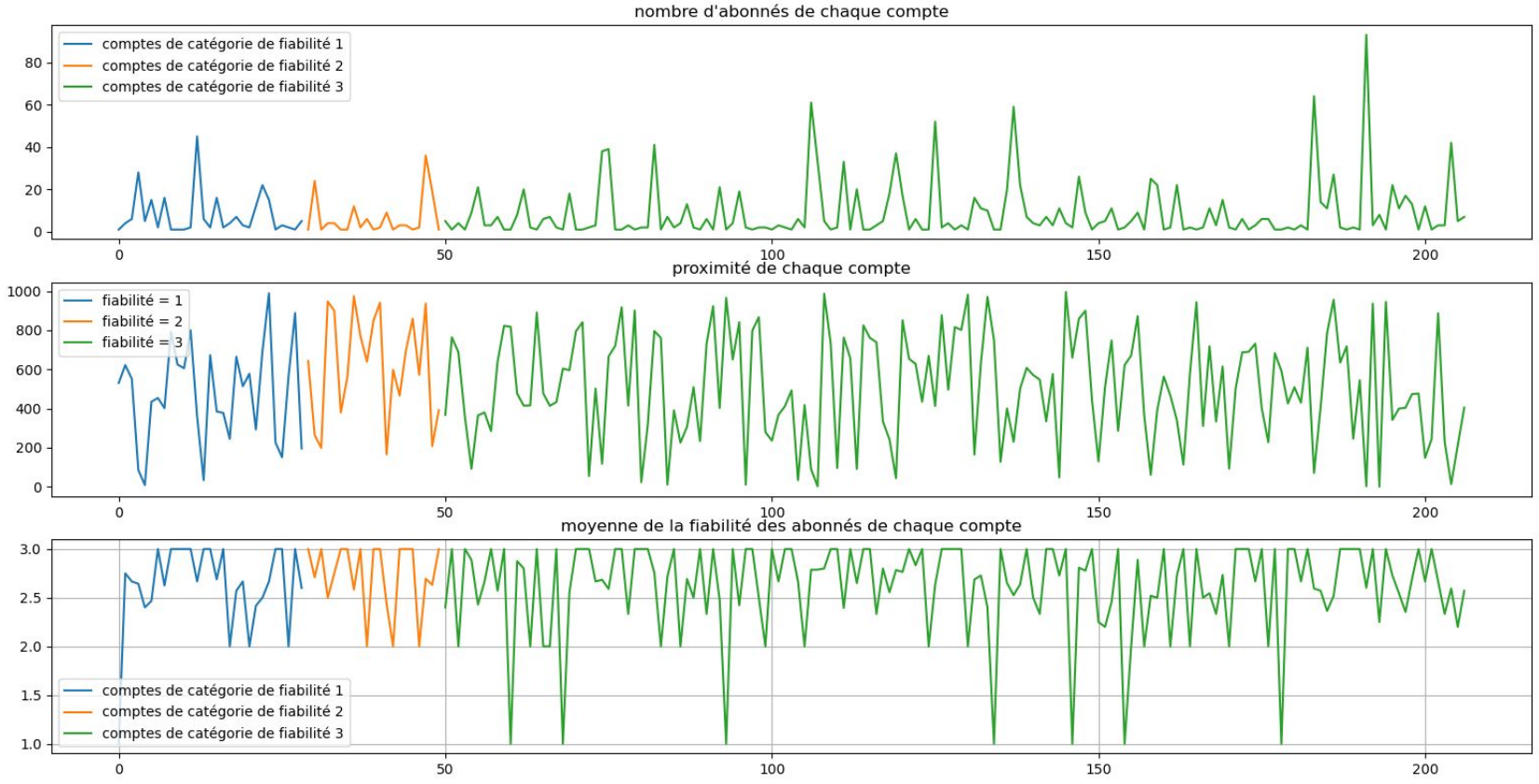
-
```
pop = len(comptes[0])
for i in range(pop):
    abonnes_compte = []
    while len(abonnes_compte) < comptes[1][i]:
        j = random.randint(0, pop - 1)
```

- 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:
    abonnes_compte.append(j)
```

Modélisation d’un réseau de comptes aléatoire



Modélisation d'un réseau de comptes aléatoire

```python
# 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 probabilite 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

```python
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, …, 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, … 0, 1, 0, …, 0, 0]

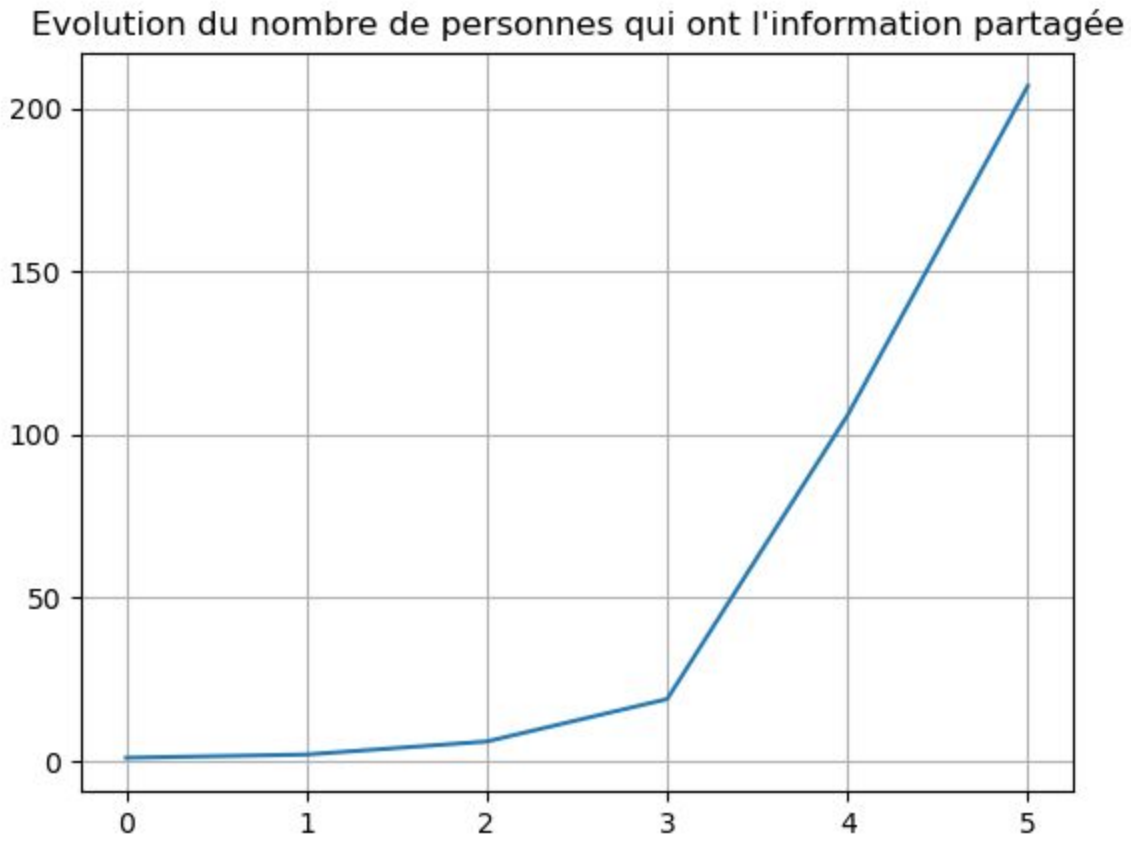
↓ ↓

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, … 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).

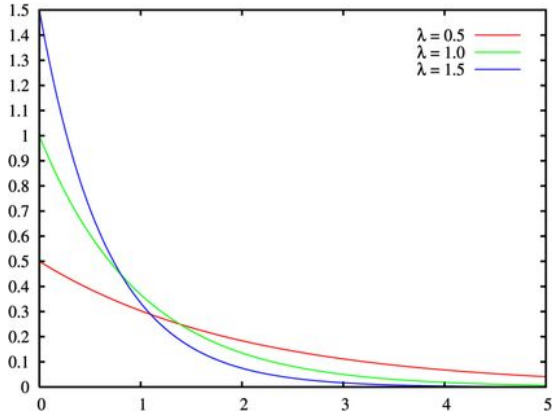
- Les utilisateurs n’ont pas d’avis sur l’information partagée

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)
```



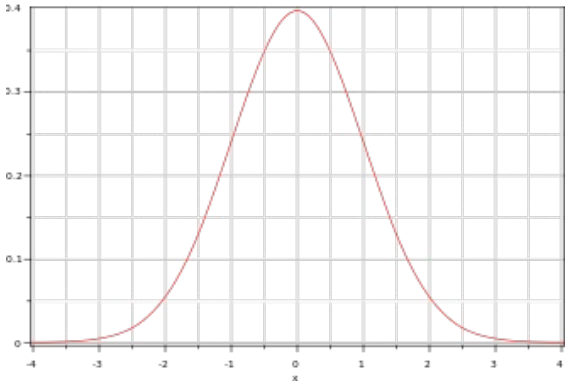
https://fr.wikipedia.org/wiki/Loi_exponentielle

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

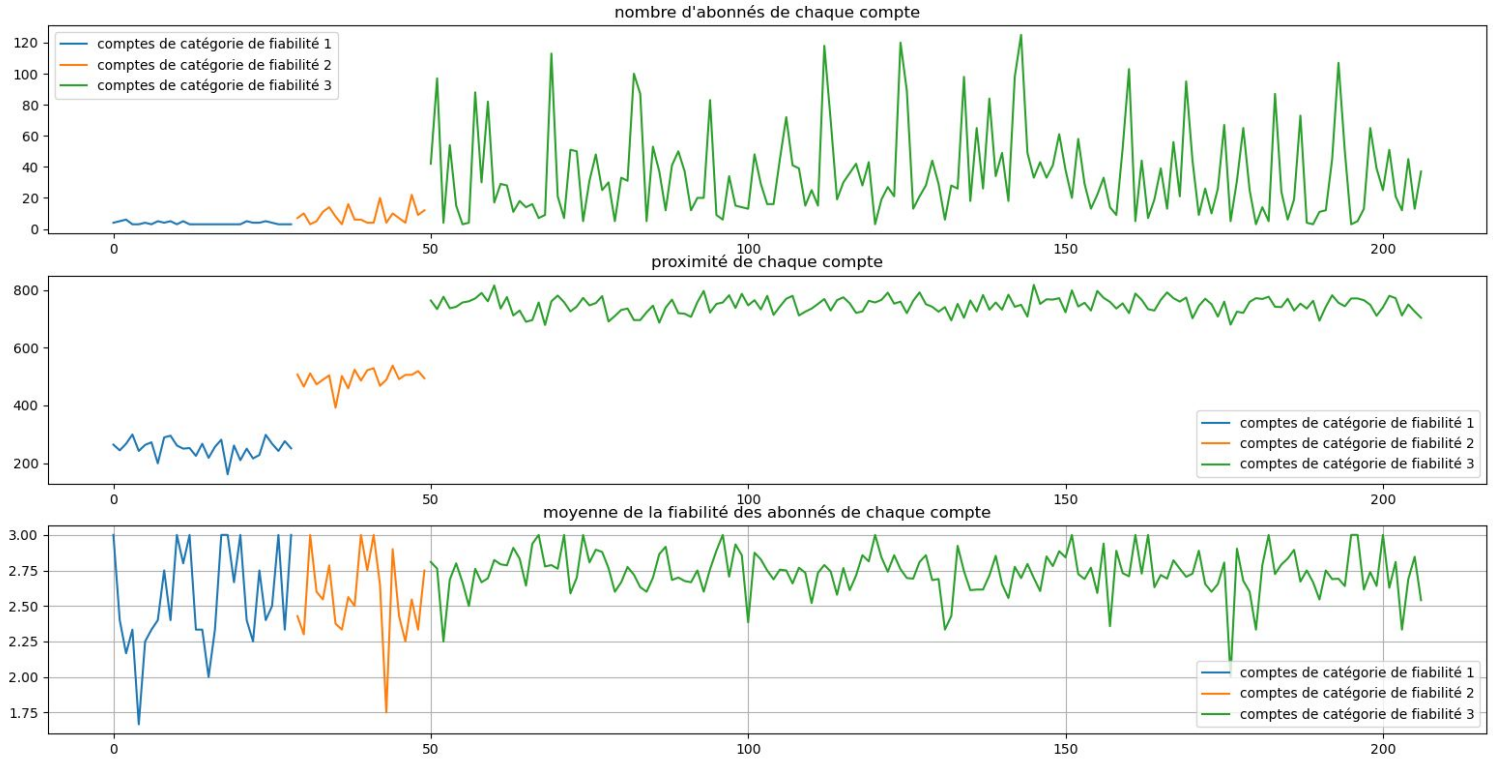
```
# moyenne des proximités des comptes de la catégorie de fiabilité 1/2/3
moyenne_1 = prox_max / 4
moyenne_2 = (2 * prox_max) / 4
moyenne_3 = (3 * prox_max) / 4
for i in range(pop):
    if liste_fiabilite[i] == 1:
        liste_proximite.append(int(random.gauss(moyenne_1, et1)))
    elif liste_fiabilite[i]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



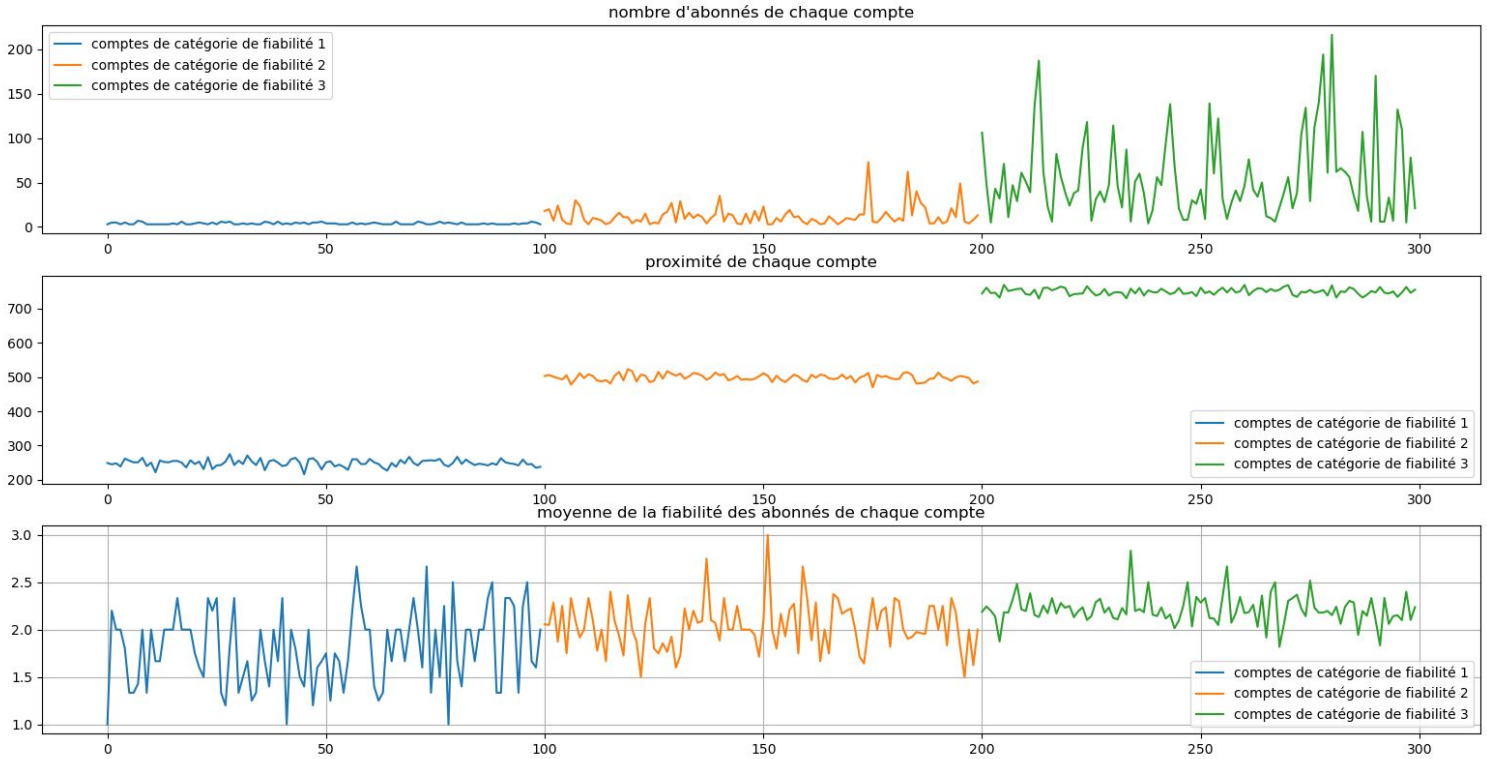
https://fr.wikipedia.org/wiki/Fonction_gaussienne

Modélisation d’un réseau de comptes aléatoire

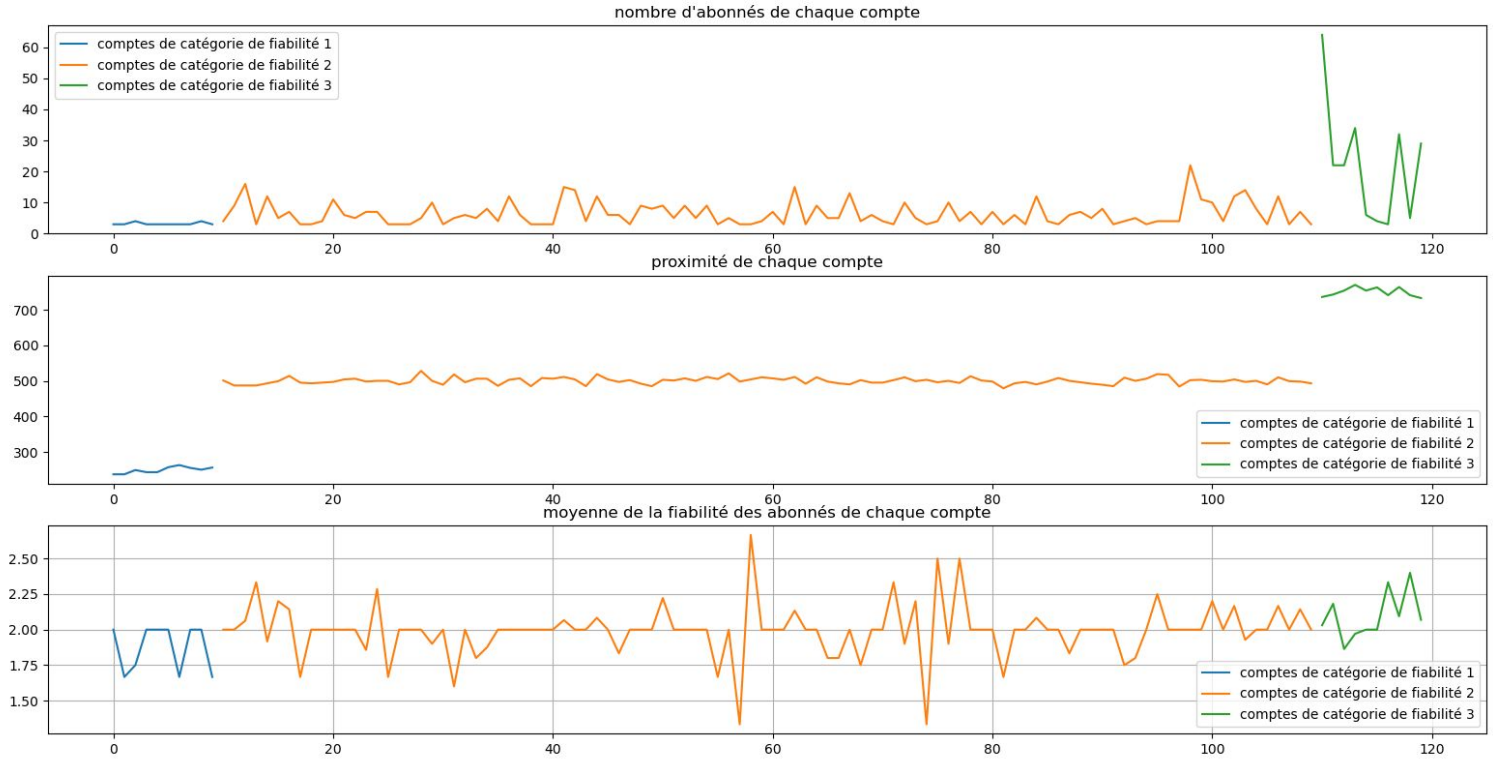
Modélisation d’un réseau de comptes aléatoire



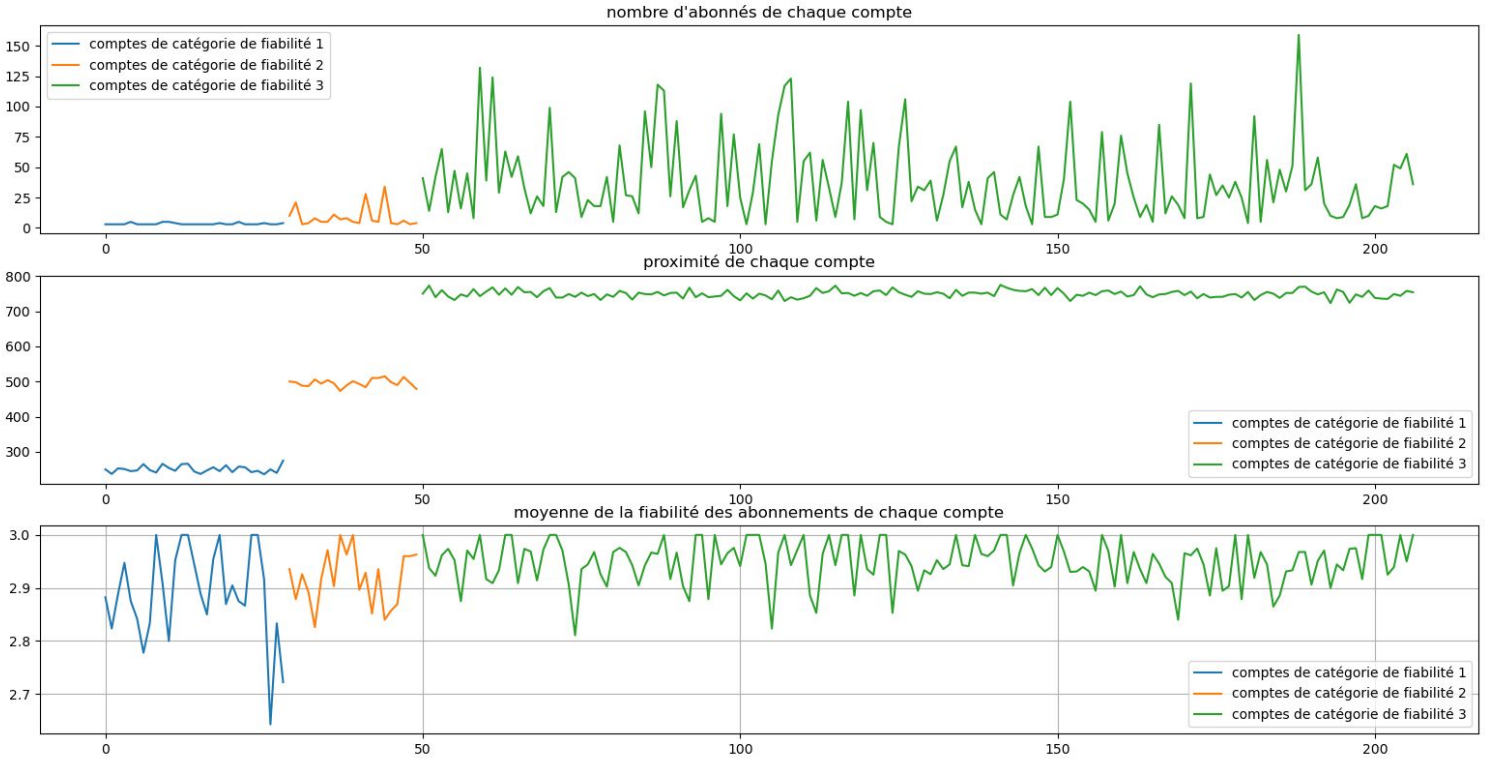
Modélisation d’un réseau de comptes aléatoire



Modélisation d’un réseau de comptes aléatoire

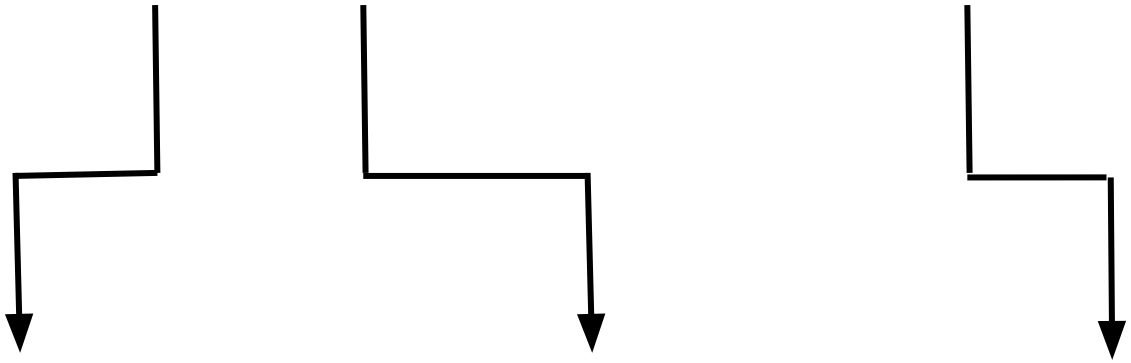


Modélisation d’un réseau de comptes aléatoire



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 partage
- 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 probabilite 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

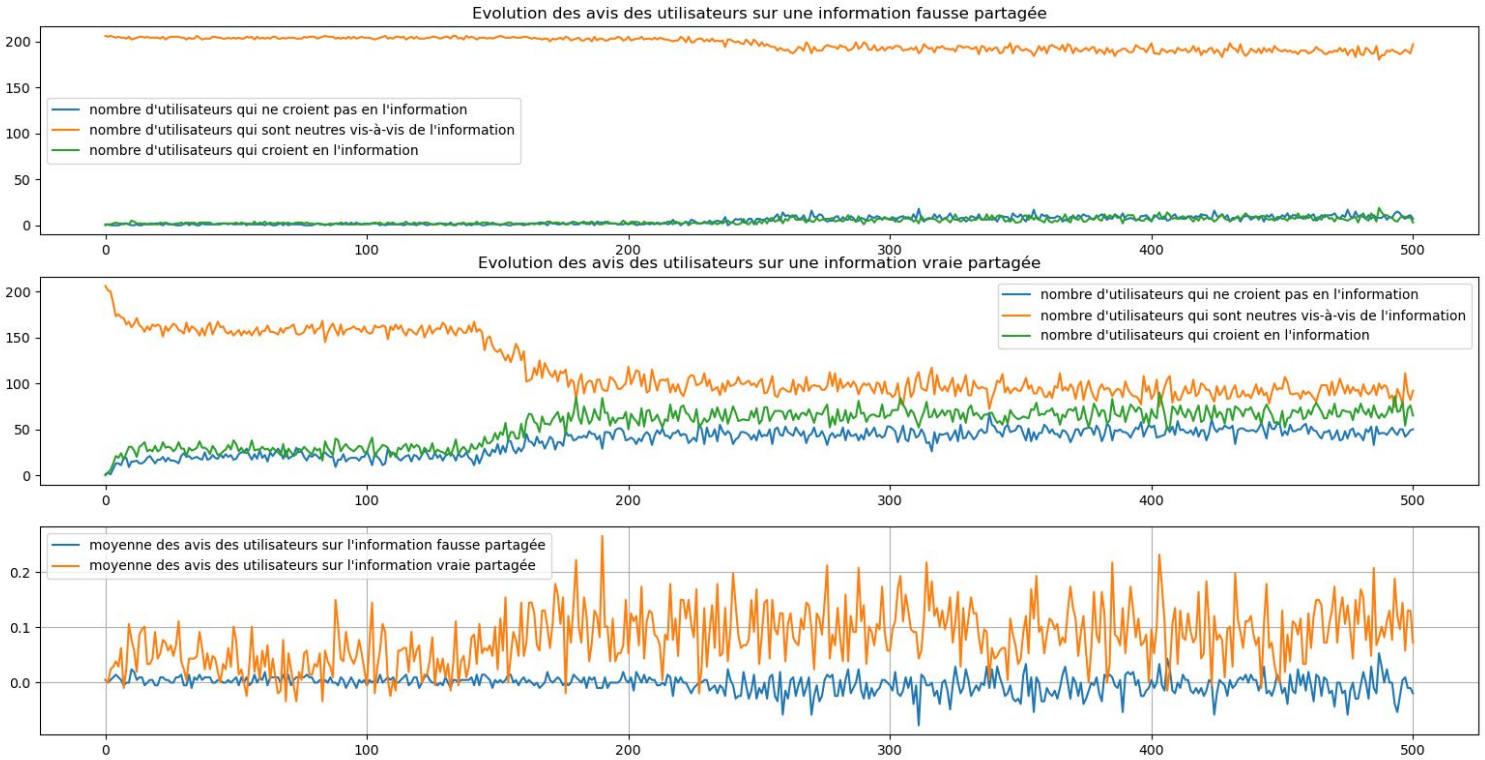
Modélisation d’un réseau de comptes aléatoire

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)
```

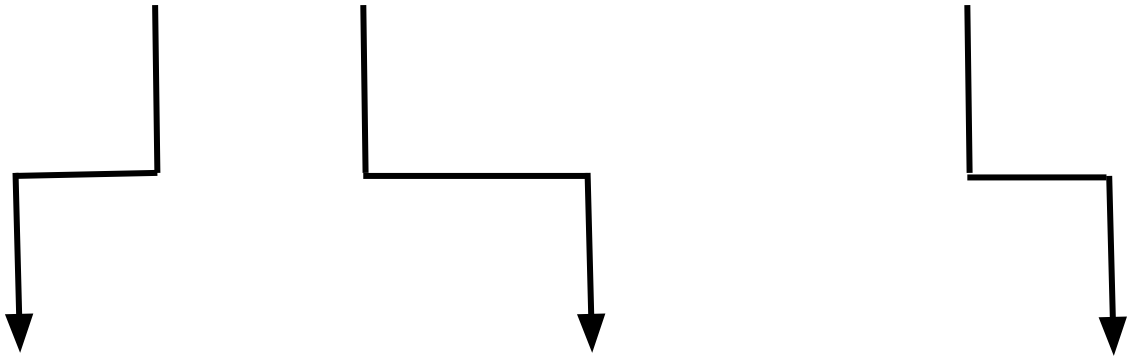
Les comptes de catégorie de fiabilité 3 ne partagent pas de fausses informations

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

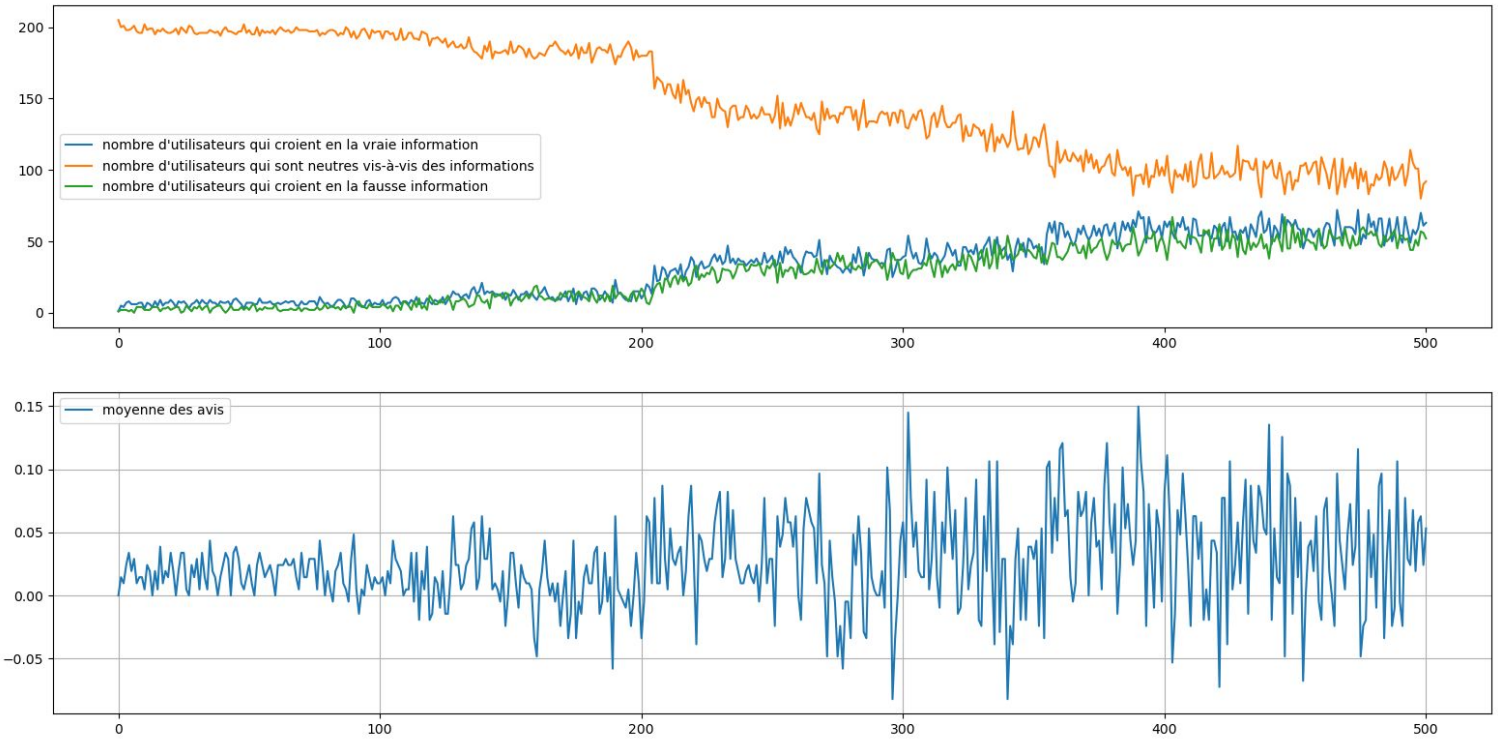
Avis de l’utilisateur:

- 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 partage
- 0 sinon

Modélisation d’un réseau de comptes aléatoire



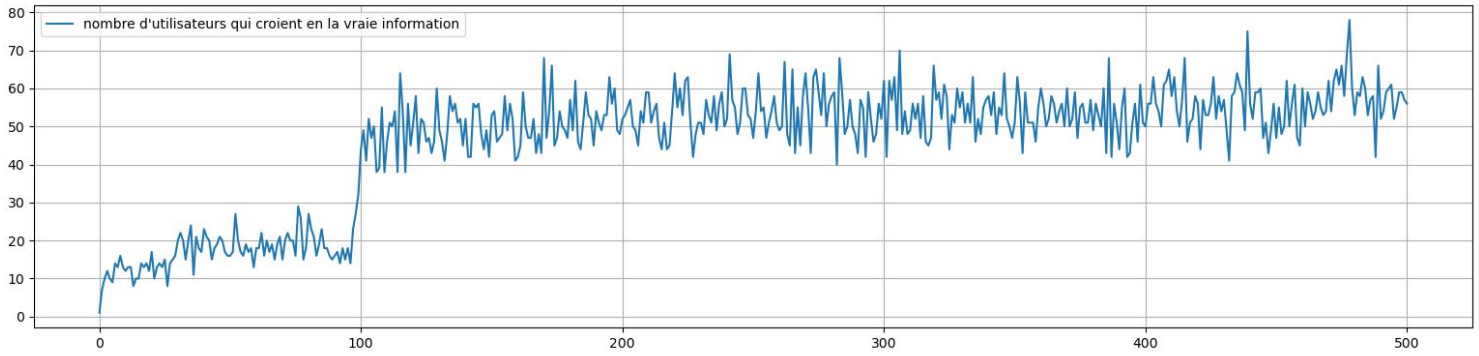
Modélisation d’un réseau de comptes aléatoire

- 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

```python
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 aléattoirement 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)
```

```
    # On dissocie le nombre d'abonnés et proximité en fonction des fiabiltés pour le tracé des graphiques
    nb_abo_1 = []
    nb_abo_2 = []
    nb_abo_3 = []
    prox1 = []
    prox2 = []
    prox3 = []
    for i in range(pop):
        if liste_fiabilite[i] == 1:
            nb_abo_1.append(liste_nb_abonnes[i])
            prox1.append(liste_proximite[i])
        elif liste_fiabilite[i] == 2:
            nb_abo_2.append(liste_nb_abonnes[i])
            prox2.append(liste_proximite[i])
        else:
            nb_abo_3.append(liste_nb_abonnes[i])
            prox3.append(liste_proximite[i])

    return [liste_num, liste_nb_abonnes, liste_proximite, liste_public, liste_fiabilite], [nb_abo_1, nb_abo_2, nb_abo_3, prox1, prox2, prox3]
```

```python
# Le programme d'avant nous a permis de déterminer le nombre d'abonnées de chaque compte. La fonction suivante va désormais déterminer à quels comptes chaque utilisateur est abonné. Plus leurs proximité est proche, plus la probabilité qu'ils s'abonnent entre eux est élevé

def abonnes_comptes(comptes, prox_max):
    pop = len(comptes[0])
    abonnees = []
    for i in range(pop):
        abonnes_i = []
        # 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 comptes ont plus de chance de s'abonner aux autres)
        while len(abonnes_i) < comptes[1][i]:
            j = random.randint(0, pop - 1)
            # 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
            e = 0
            for abonne in abonnes_i:
                if abonne == j:
                    e += 1
            # Un compte ne peux pas s'abonner à lui-même
            if j != i:
                if e == 0:
            # Plus l'utilisateur i a une proximité proche du j, plus le j a de chances de s'abonner au i
                    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)
                    # print(id_abo)
            # Si conditions remplies, j est ajouté aux abonnées de i, et i aux abonnements de j
                    if id > 0:
                        abonnes_i.append(j)
        abonnees.append(abonnes_i)
    return abonnes
```

```python
def comptes_avec_relation(prox_max, nb_1, nb_2, nb_3):
    comptes, L2 = creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3)
    abonnees = abonnes_comptes(comptes, prox_max)
    comptes.append(abonnees)

    # On cherche désormais à regarder la moyenne de fiabilité des abonnés de chaque compte i en fonction de la fiabilité de i
    # On utilisera abo pour abonnés
    moy_fiab_abo_1 = []
    moy_fiab_abo_2 = []
    moy_fiab_abo_3 = []
    pop = len(comptes[0])
    for i in range(pop):
        moy_fiab_abo = 0
        if comptes[4][i] == 1:
            for abonne in comptes[5][i]:
                moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
            moy_fiab_abo_1.append(moy_fiab_abo)
        elif comptes[4][i] == 2:
            for abonne in comptes[5][i]:
                moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
            moy_fiab_abo_2.append(moy_fiab_abo)
        else:
            for abonne in comptes[5][i]:
                moy_fiab_abo += comptes[4][abonne] / comptes[1][i]
            moy_fiab_abo_3.append(moy_fiab_abo)
    L2.append(moy_fiab_abo_1)
    L2.append(moy_fiab_abo_2)
    L2.append(moy_fiab_abo_3)

    # Tracé des graphiques
    plt.subplot(3, 1, 1)
    plt.plot(list(range(nb_1)), L2[0], label = 'comptes de catégorie de fiabilité 1')
    plt.legend(loc='upper left')
    plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[1], label = 'comptes de catégorie de fiabilité 2')
    plt.legend(loc='upper left')
    plt.plot(list(range(nb_1 + nb_2, pop)), L2[2], label = 'comptes de catégorie de fiabilité 3')
    plt.legend(loc='upper left')
    plt.title('nombre d\'abonnés de chaque compte')

    plt.subplot(3, 1, 2)
    plt.plot(list(range(nb_1)), L2[3], label = 'fiabilité = 1')
    plt.legend(loc='upper right')
    plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[4], label = 'fiabilité = 2')
    plt.legend(loc='upper right')
    plt.plot(list(range(nb_1 + nb_2, pop)), L2[5], label = 'fiabilité = 3')
    plt.legend(loc='upper left')
    plt.title('proximité de chaque compte')
```

```
    plt.subplot(3, 1, 3)
    plt.plot(list(range(nb_1)), L2[6], label = 'comptes de catégorie de fiabilité 1')
    plt.legend(loc='upper left')
    plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[7], label = 'comptes de catégorie de fiabilité 2')
    plt.legend(loc='upper left')
    plt.plot(list(range(nb_1 + nb_2, pop)), L2[8], label = 'comptes de catégorie de fiabilité 3')
    plt.legend(loc='lower left')
    plt.title('moyenne de la fiabilité des abonnés de chaque compte')

    plt.grid()
    plt.show()

    return comptes, L2


# On initialise la 7ième liste, information, de comptes
def creation_comptes_avec_information(comptes):
    pop = len(comptes[0])
    # On choisit aléatoirement un compte qui partage une information
    f = random.randint(0, pop - 1)
    information = []
    for i in range(pop):
        if i != f:
            information.append(0)
        else:
            information.append(1)
    comptes.append(information)
    return comptes
```

```python
# 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 probabilite 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


# A chaque tour de boucle dans les simulations, les comptes vont avoir la possibilité de gagner des abonnés
# Cette fonction marche de la même manière que la fonction abonnées comptes
def gain_abonnees(comptes, prox_max):
    pop = len(comptes[0])
    for i in range(pop):
        for j in range(pop):
            e = 0
            for l in comptes[5][i]:
                if comptes[0][j] == l:
                    e += 1
            if i != j:
                if e == 0:
                    # Les chances que le compte gagne des abonnés varient en fonction de la fiabilé de celui-ci
                    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)
        comptes[1][i] = len(comptes[5][i])
    return comptes
```

```python
# Retourne une liste contenant l'identifiant de tous les comptes ayant l'information
def info(comptes):
    pop = len(comptes[0])
    inf = []
    for k in range(pop):
        if comptes[6][k] == 1:
            inf.append(k)
    return inf


# Retourne une liste contenant l'identifiant de tous les comptes qui sont abonnés à un compte ayant l'information
def num(comptes, inf):
    num = []
    for i in inf:
        for j in comptes[5][i]:
            num.append(j)
    return num



# Tous les comptes qui sont abonnés à un compte ayant l'information ont désormais l'information
def comptes_rang_5(comptes, num):
    pop = len(comptes[0])
    for j in num:
        comptes[6][j] = 1
    return comptes



# équivalent des 3 dernières fonctions
def comptes_info_num(comptes):
    inf = info(comptes)
    num_ = num(comptes, inf)
    return comptes_rang_5(comptes, num_)

```

```python
# 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
def simulation2(prox_max, nb_1, nb_2, nb_3):
    comptes, L2 = comptes_avec_relation(prox_max, nb_1, nb_2, nb_3)
    comptes = creation_comptes_avec_information(comptes)
    final = []
    pop = len(comptes[0])
    for _ in range(pop):
        final.append(1)
    a = 1
    liste_nb_info = [1]
    while comptes[6] != final:
        comptes = comptes_info_num(comptes)
        a += 1
        inf = 0
        for i in range(pop):
            inf += comptes[6][i]
        liste_nb_info.append(inf)
        comptes = gain_abonnees(comptes, prox_max)

    plt.plot(list(range(a)), liste_nb_info)
    plt.title('Evolution du nombre de personnes qui ont l\'information partagée')
    plt.grid()
    plt.show()

    return a, liste nb info
```

```python
# Crée un réseau de comptes aléatoire
# Simule le partage d'une information par chacun des comptes
# Renvoie le nombre moyen d'étapes nécessaires à la propagation de l'information sur le réseau pour chaque catégorie de fiabilité de compte
def graph_simulation2_(prox_max, nb_1, nb_2, nb_3):

    y_moy = [0, 0, 0]
    comptes = comptes_avec_relation(prox_max, nb_1, nb_2, nb_3)
    pop = len(comptes[0])
    for i in range(pop):
        comptes_ = list(comptes)
        information = []
        for j in range(pop):
            if j != i:
                information.append(0)
            else:
                information.append(1)
        comptes_.append(information)
        # temps de propagation
        e = propagation_information2(comptes_, prox_max)
        y.append(e)
        if a == 1:
            y_moy[0] += e / nb_1
        elif a == 2:
            y_moy[1] += e / nb_2
        else:
            y_moy[2] += e / nb_3
    return y_moy
```

```
## Simulation 2


# 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, et1, et2, et3):

    # listes des caractéristiques des comptes
    liste_nb_abonnes = []
    liste_proximite = []
    liste_fiabilite = []

    # on associe à chaque compte une catégorie de fiabilité aléatoire
    fiabilite_possible = [1] * nb_1 + [2] * nb_2 + [3] * nb_3
    # La population totale est donc:
    pop = nb_1 + nb_2 + nb_3
    for _ in range(pop):
        liste_fiabilite.append(fiabilite_possible.pop(random.randint(0, len(fiabilite_possible) - 1)))

    # on associe à chaque compte un identifiant
        liste_id = list(range(pop))

    # on associe à chaque compte une proximité. Les comptes de même catégorie de fiabilité ont plus de chances d'avoir une proximité proche.
    # On choisit une distribution gaussienne avec une moyenne différente et un écart-type différent en fonction de la catégorie de fiabilité des comptes
    # 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)))

```

```
    # on associe à chaque compte un nombre d'abonnés initial
    # On choisit une distribution exponentielle du nombre d'abonnés, avec une moyenne différente en fonction de la catégorie de fiabilité du
compte
    # les comptes de catégorie 2(resp 3) ont une moyenne d'abonnés 6(resp 36) fois supérieure à ceux de catégorie 1
    # 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)
    # On sépare les proximités et nb d'abonnés en fonction de leur fiabilité pour le tracé des courbes
    nb_abo_1 = []
    nb_abo_2 = []
    nb_abo_3 = []
    prox1 = []
    prox2 = []
    prox3 = []
    for i in range(pop):
        if liste_fiabilite[i] == 1:
            nb_abo_1.append(liste_nb_abonnes[i])
            prox1.append(liste_proximite[i])
        elif liste_fiabilite[i] == 2:
            nb_abo_2.append(liste_nb_abonnes[i])
            prox2.append(liste_proximite[i])
        else:
            nb_abo_3.append(liste_nb_abonnes[i])
            prox3.append(liste_proximite[i])
    return [liste_id, liste_nb_abonnes, liste_proximite, liste_fiabilite], [nb_abo_1, nb_abo_2, nb_abo_3, prox1, prox2, prox3]
```

```python
# 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

def abonnes_comptes(comptes, prox_max):
    pop = len(comptes[0])
    abonnees = []
    abonnements = []
    for _ in range(pop):
        abonnements.append([])
    for i in range(pop):
        abonnes_i = []
        # 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)
        while len(abonnes_i) < comptes[1][i]:
            j = random.randint(0, pop - 1)
            # 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
            e = 0
            for abonne in abonnes_i:
                if abonne == j:
                    e += 1
            # Un compte ne peut pas s'abonner à lui-même
            if j != i:
                if e == 0:
            # Plus l'utilisateur i a une proximité proche du j, plus le j a de chances de s'abonner au i
                    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)
                    # print(id_abo)
            # si conditions remplies, j est ajouté aux abonnées de i, et i aux abonnements de j
                    if id > 0:
                        abonnes_i.append(j)
        abonnees.append(abonnes_i)
    for i in range(pop):
        for abonnee in abonnees[i]:
            abonnements[abonnee].append(i)

    return abonnees, abonnements
```

```python
# Crée un réseau de comptes où les comptes ont des abonnés
def comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3):
    comptes, L2 = creation_liste_aleatoire(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
    abonnees, abonnements = abonnes_comptes(comptes, prox_max)
    comptes.append(abonnees)
    comptes.append(abonnements)

    # On cherche désormais à regarder la moyenne de fiabilité des abonnés de chaque compte i en fonction de la fiabilité de i
    # On utilisera abo pour abonnés
    moy_fiab_abo_1 = []
    moy_fiab_abo_2 = []
    moy_fiab_abo_3 = []
    pop = len(comptes[0])
    for i in range(pop):
        moy_fiab_abo = 0
        if comptes[3][i] == 1:
            for abonne in comptes[4][i]:
                moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
            moy_fiab_abo_1.append(moy_fiab_abo)
        elif comptes[3][i] == 2:
            for abonne in comptes[4][i]:
                moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
            moy_fiab_abo_2.append(moy_fiab_abo)
        else:
            for abonne in comptes[4][i]:
                moy_fiab_abo += comptes[3][abonne] / comptes[1][i]
            moy_fiab_abo_3.append(moy_fiab_abo)
    L2.append(moy_fiab_abo_1)
    L2.append(moy_fiab_abo_2)
    L2.append(moy_fiab_abo_3)
```

```
    # On cherche désormais à regarder la moyenne de fiabilité des abonnements de chaque compte i en fonction de la fiabilité de i
    # On utilisera ab pour abonnements

    moy_fiab_ab_1 = []
    moy_fiab_ab_2 = []
    moy_fiab_ab_3 = []
    pop = len(comptes[0])
    for i in range(pop):
        moy_fiab_ab = 0
        if comptes[3][i] == 1:
            for abonnement in comptes[5][i]:
                moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
            moy_fiab_ab_1.append(moy_fiab_ab)
        elif comptes[3][i] == 2:
            for abonnement in comptes[5][i]:
                moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
            moy_fiab_ab_2.append(moy_fiab_ab)
        else:
            for abonnement in comptes[5][i]:
                moy_fiab_ab += comptes[3][abonnement] / len(comptes[5][i])
            moy_fiab_ab_3.append(moy_fiab_ab)
    L2.append(moy_fiab_ab_1)
    L2.append(moy_fiab_ab_2)
    L2.append(moy_fiab_ab_3)


    # plt.subplot(3, 1, 1)
    # plt.plot(list(range(nb_1)), L2[0], label = 'comptes de catégorie de fiabilité 1')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[1], label = 'comptes de catégorie de fiabilité 2')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1 + nb_2, pop)), L2[2], label = 'comptes de catégorie de fiabilité 3')
    # plt.legend(loc='upper left')
    # plt.title('nombre d\'abonnés de chaque compte')
    #
    # plt.subplot(3, 1, 2)
    # plt.plot(list(range(nb_1)), L2[3], label = 'comptes de catégorie de fiabilité 1')
    # plt.legend(loc='upper right')
    # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[4], label = 'comptes de catégorie de fiabilité 2')
    # plt.legend(loc='upper right')
    # plt.plot(list(range(nb_1 + nb_2, pop)), L2[5], label = 'comptes de catégorie de fiabilité 3')
    # plt.legend(loc='lower right')
    # plt.title('proximité de chaque compte')
```

```
    # plt.subplot(3, 1, 3)
    # plt.plot(list(range(nb_1)), L2[6], label = 'comptes de catégorie de fiabilité 1')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[7], label = 'comptes de catégorie de fiabilité 2')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1 + nb_2, pop)), L2[8], label = 'comptes de catégorie de fiabilité 3')
    # plt.legend(loc='lower right')
    # plt.title('moyenne de la fiabilité des abonnés de chaque compte')

    # plt.subplot(3, 1, 3)
    # plt.plot(list(range(nb_1)), L2[9], label = 'comptes de catégorie de fiabilité 1')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1, nb_1 + nb_2)), L2[10], label = 'comptes de catégorie de fiabilité 2')
    # plt.legend(loc='upper left')
    # plt.plot(list(range(nb_1 + nb_2, pop)), L2[11], label = 'comptes de catégorie de fiabilité 3')
    # plt.legend(loc='lower right')
    # plt.title('moyenne de la fiabilité des abonnements de chaque compte')

    plt.grid()
    plt.show()

    return comptes, L2


# A chaque étape les utilisateurs ont la possibilté de gagner des abonnés

def gain_abonnees(comptes, prox_max):
    pop = len(comptes[0])
    for i in range(pop):
        for j in range(pop):
            e = 0
            for l in comptes[4][i]:
                if comptes[0][j] == l:
                    e += 1
            if i != j:
                if e == 0:
                    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[4][i].append(id)
        comptes[1][i] = len(comptes[4][i])
    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
```

```
    # on effectue une copie élément par élément de comptes car la fonction bug sinon

def copie_comptes(comptes):
    pop = len(comptes[0])
    comptes_ = []
    for i in range(7):
        copie_compte = []
        if i == 4:
            abonne = []
            for j in range(len(comptes[4])):
                abonne_j = []
                for k in range(len(comptes[4][j])):
                    abonne_j.append(comptes[4][j][k])
                abonne.append(abonne_j)
            comptes_.append(abonne)
        elif i == 5:
            abonnement = []
            for j in range(len(comptes[5])):
                abonnement_j = []
                for k in range(len(comptes[5][j])):
                    abonnement_j.append(comptes[5][j][k])
                abonnement.append(abonnement_j)
            comptes_.append(abonnement)
        elif i == 6:
            inf = []
            for j in range(len(comptes[6])):
                inf_j = []
                for k in range(len(comptes[6][j])):
                    if k == 0:
                        liste_avis = []
                        for l in range(len(comptes[6][j][0])):
                            liste_avis.append(comptes[6][j][0][l])
                        inf_j.append(liste_avis)
                    else:
                        inf_j.append(comptes[6][j][k])
                inf.append(inf_j)
            comptes_.append(inf)
        else:
            for j in range(pop):
                copie_compte.append(comptes[i][j])
            comptes_.append(copie_compte)
    return comptes_
```

```python
# Méthode de partage d'une fausse information
def partage_fausse_information(comptes):

#        # On copie le compte pour ne pas influencer la propagation de l'information
    fiabilite = comptes[3]
    abonnes = comptes[4]
    abonnements = comptes[5]
    informations = comptes[6]
    pop = len(comptes[0])
    comptes_ = copie_comptes(comptes)

    # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
    for i in range(pop):
        fiab = fiabilite[i]
        # on regarde si les comptes auquels l'utilisateur i est abonné ont partagé l'information
        for ab in abonnements[i]:
            partage = informations[ab][2]
            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)
        comptes_[6][i][1] = random.choice(comptes_[6][i][0])
        # si l'utilisateur i croit désormais en l'information, il va avoir la possiblité de la partager
        if comptes_[6][i][1] == 1:
            if comptes_[6][i][2] != 1:
                # les comptes fiables ne partagent pas de fausses informations, les autres peuvent le faire
                if fiab != 3:
                    comptes_[6][i][2] = retweet_fiabilite(fiab)
                else:
                    comptes_[6][i][2]


    return comptes_
```

```python
# Méthode de partage d'une vraie information
def partage_vraie_information(comptes):

    # On copie le compte pour ne pas influencer la propagation de l'information
    fiabilite = comptes[3]
    abonnes = comptes[4]
    abonnements = comptes[5]
    informations = comptes[6]
    pop = len(comptes[0])
    comptes_ = copie_comptes(comptes)

    # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
    for i in range(pop):
        fiab = fiabilite[i]
        # on regarde si les comptes auquels l'utilisateur i est abonné ont partagé l'information
        for ab in abonnements[i]:
            partage = informations[ab][2]
            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)
        comptes_[6][i][1] = random.choice(comptes_[6][i][0])
        # si l'utilisateur i croit désormais en l'information, il va avoir la possiblité de la partager
        if comptes_[6][i][1] == 1:
            if comptes_[6][i][2] != 1:
                # les comptes diffusant des fake news ne partagent pas les vraies informations, les autres peuvent le faire
                if fiab != 1:
                    comptes_[6][i][2] = retweet_fiabilite(fiab)
                else:
                    comptes_[6][i][2] = 0

    return comptes
```

```python
# On simule la propagation d'une information vraie et d'une information fausse sur n étapes
def propagation_information2(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n):
    comptes, L2 = comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
    comptes = comptes_avec_information(comptes)
    pop = len(comptes[0])
    comptes_ = copie_comptes(comptes)
    # Listes pour les tracés de graphiques
    croit_pas_vraie = [0]
    neutre_vraie = [pop - 1]
    croit_vraie = [1]
    moyenne_vraie = [1 / pop]
    croit_pas_fausse = [0]
    neutre_fausse = [pop - 1]
    croit_fausse = [1]
    moyenne_fausse = [1 / pop]

    # On effectue simulation de la propagation d'une vraie information
    # On cherche un compte de fiabilité 3 qui va partager une vraie information
    j_1 = random.randint(0, pop - 1)
    while comptes[3][j_1] != 3:
        j_1 = random.randint(0, pop - 1)
    comptes[6][j_1][0] = [1]
    comptes[6][j_1][1] = 1
    comptes[6][j_1][2] = 1

    for _ in range(n):
        # comptes_ = gain_abonnees(comptes_)
        comptes = partage_vraie_information(comptes)
        croit_pas = 0
        croit = 0
        neutre = 0
        for i in range(pop):
            if comptes[6][i][1] == -1:
                croit_pas += 1
            elif comptes[6][i][1] == 0:
                neutre += 1
            else:
                croit += 1
        croit_pas_vraie.append(croit_pas)
        neutre_vraie.append(neutre)
        croit_vraie.append(croit)
        moyenne_vraie.append((-croit_pas + croit)/pop)
```

```
    # On effectue simulation de la propagation d'une fausse information
    # On cherche un compte de fiabilité 1 qui va partager une vraie information
    j_2 = random.randint(0, pop - 1)
    while comptes_[3][j_2] != 1:
        j_2 = random.randint(0, pop - 1)
    comptes_[6][j_2][0] = [1]
    comptes_[6][j_2][1] = 1
    comptes_[6][j_2][2] = 1

    for _ in range(n):
        # comptes_ = gain_abonnees(comptes)
        comptes_ = partage_fausse_information(comptes_)
        croit_pas = 0
        croit = 0
        neutre = 0
        for i in range(pop):
            if comptes_[6][i][1] == -1:
                croit_pas += 1
            elif comptes_[6][i][1] == 0:
                neutre += 1
            else:
                croit += 1
        croit_pas_fausse.append(croit_pas)
        neutre_fausse.append(neutre)
        croit_fausse.append(croit)
        moyenne_fausse.append((-croit_pas + croit)/pop)

    plt.subplot(3, 1, 1)
    plt.plot(list(range(n + 1)), croit_pas_fausse, label = 'nombre d\'utilisateurs qui ne croient pas en l\'information')
    plt.legend(loc='center left')
    plt.plot(list(range(n + 1)), neutre_fausse, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis de l\'information')
    plt.legend(loc='center left')
    plt.plot(list(range(n + 1)), croit_fausse, label = 'nombre d\'utilisateurs qui croient en l\'information')
    plt.legend(loc='center left')
    plt.title('Evolution des avis des utilisateurs sur une information fausse partagée')

    plt.subplot(3, 1, 2)
    plt.plot(list(range(n + 1)), croit_pas_vraie, label = 'nombre d\'utilisateurs qui ne croient pas en l\'information')
    plt.legend(loc='upper right')
    plt.plot(list(range(n + 1)), neutre_vraie, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis de l\'information')
    plt.legend(loc='upper right')
    plt.plot(list(range(n + 1)), croit_vraie, label = 'nombre d\'utilisateurs qui croient en l\'information')
    plt.legend(loc='upper right')
    plt.title('Evolution des avis des utilisateurs sur une information vraie partagée')
```

```
    plt.subplot(3, 1, 3)
    plt.plot(list(range(n + 1)), moyenne_fausse, label = 'moyenne des avis des utilisateurs sur l\'information fausse partagée')
    plt.legend(loc='upper left')
    plt.plot(list(range(n + 1)), moyenne_vraie, label = 'moyenne des avis des utilisateurs sur l\'information vraie partagée')
    plt.legend(loc='upper left')

    plt.grid()
    plt.show()



def partage_information(comptes):

    # On copie le compte pour ne pas influencer la propagation de l'information
    fiabilite = comptes[3]
    abonnes = comptes[4]
    abonnements = comptes[5]
    informations = comptes[6]
    pop = len(comptes[0])

    comptes_ = copie_comptes(comptes)

    # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
    for i in range(pop):
        fiab = fiabilite[i]
        # on regarde si les comptes auquels l'utilisateur i est abonné ont partagé l'information

        for ab in abonnements[i]:
            partage = informations[ab][2]
            if partage == 1:
                if informations[ab][1] == 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)
                elif informations[ab][1] == -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)
        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_
```

```python
def propagation_information2_2(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n):
    comptes, L2 = comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
    comptes = comptes_avec_information(comptes)
    pop = len(comptes[0])

    # Listes pour les tracés de graphiques
    croit_en_vraie = [1]
    neutre = [pop - 2]
    croit_en_faux = [1]
    moyenne = [0]

    # On effectue simulation de la propagation d'une vraie information
    # On cherche un compte de fiabilité 3 qui va partager une vraie information
    j_1 = random.randint(0, pop - 1)
    while comptes[3][j_1] != 3:
        j_1 = random.randint(0, pop - 1)
        # print(comptes[3][j_1])
    comptes[6][j_1][0] = [1]
    comptes[6][j_1][1] = 1
    comptes[6][j_1][2] = 1

    j_2 = random.randint(0, pop - 1)
    while comptes[3][j_2] != 1:
        j_2 = random.randint(0, pop - 1)
    comptes[6][j_2][0] = [1]
    comptes[6][j_2][1] = 1
    comptes[6][j_2][2] = 1

    for i in range(n):
        # comptes = gain_abonnees(comptes, prox_max)
        comptes = partage_information(comptes)
        croit_vraie = 0
        croit_faux = 0
        croit_neutre = 0
        for i in range(pop):
            if comptes[6][i][1] == -1:
                croit_faux += 1
            elif comptes[6][i][1] == 0:
                croit_neutre += 1
            else:
                croit_vraie += 1
        croit_en_vraie.append(croit_vraie)
        neutre.append(croit_neutre)
        croit_en_faux.append(croit_faux)
        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()
```

```python
def partage_information_campagne_prevention(comptes, signalements, signalement_necessaire):

    # On copie le compte pour ne pas influencer la propagation de l'information
    fiabilite = comptes[3]
    abonnes = comptes[4]
    abonnements = comptes[5]
    informations = comptes[6]
    pop = len(comptes[0])
    comptes_ = copie_comptes(comptes)
    liste_signalements = []
    for i in range(pop):
        liste_signalements.append(comptes[7][i])
    comptes_.append(liste_signalements)

    # Si l'information fausse a été trop de fois signalée, on stoppe son partage
    if signalements >= signalement_necessaire:
        for i in range(pop):
            if comptes[6][i][1] == -1:
                comptes[6][i][2] == 0

    # Chaque compte va avoir la possibilité de changer d'avis, et de partager ou non l'information en fonction de sa fiabilité
    for i in range(pop):
        fiab = fiabilite[i]
        # on regarde si les comptes auquels l'utilisateur i est abonné ont partagé l'information
        for ab in abonnements[i]:
            partage = informations[ab][2]
            if partage == 1:
                if informations[ab][1] == 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)
                elif informations[ab][1] == -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)
        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_
```

```python
def propagation_information2_2_prevention(prox_max, nb_1, nb_2, nb_3, et1, et2, et3, n, signalement_necessaire):
    comptes, L2 = comptes_avec_abonnes(prox_max, nb_1, nb_2, nb_3, et1, et2, et3)
    comptes = comptes_avec_information(comptes)
    pop = len(comptes[0])
    liste_signalements = []
    for _ in range(pop):
        liste_signalements.append(0)
    comptes.append(liste_signalements)

    # Listes pour les tracés de graphiques
    croit_en_vraie = [1]
    neutre = [pop - 2]
    croit_en_faux = [1]
    moyenne = [0]

    # On effectue simulation de la propagation d'une vraie information
    # On cherche un compte de fiabilité 3 qui va partager une vraie information
    j_1 = random.randint(0, pop - 1)
    while comptes[3][j_1] != 3:
        j_1 = random.randint(0, pop - 1)
        # print(comptes[3][j_1])
    comptes[6][j_1][0] = [1]
    comptes[6][j_1][1] = 1
    comptes[6][j_1][2] = 1
```

```
    j_2 = random.randint(0, pop - 1)
    while comptes[3][j_2] != 1:
        j_2 = random.randint(0, pop - 1)
    comptes[6][j_2][0] = [1]
    comptes[6][j_2][1] = 1
    comptes[6][j_2][2] = 1
    signalements = 0
    a = 0
    for _ in range(n):
        # comptes = gain_abonnees(comptes, prox_max)
        comptes = partage_information_campagne_prevention(comptes, signalements, signalement_necessaire)
        signalements = 0
        croit_vraie = 0
        croit_faux = 0
        croit_neutre = 0
        for i in range(pop):
            signalements += comptes[7][i]
            if comptes[6][i][1] == -1:
                croit_faux += 1
            elif comptes[6][i][1] == 0:
                croit_neutre += 1
            else:
                croit_vraie += 1
        croit_en_vraie.append(croit_vraie)
        neutre.append(croit_neutre)
        croit_en_faux.append(croit_faux)
        moyenne.append((-croit_faux + croit_vraie) / pop)
        a += 1
        if signalements < signalement_necessaire:
            print(a)


    plt.subplot(2, 1, 1)
    plt.plot(list(range(n + 1)), croit_en_vraie, label = 'nombre d\'utilisateurs qui croient en la vraie information')
    plt.legend(loc='upper left')
    # plt.plot(list(range(n + 1)), neutre, label = 'nombre d\'utilisateurs qui sont neutres vis-à-vis des informations')
    # plt.legend(loc='center left')
    # plt.plot(list(range(n + 1)), croit_en_faux, label = 'nombre d\'utilisateurs qui croient en la fausse information')
    # plt.legend(loc='center left')
    #
    # plt.subplot(2, 1, 2)
    # plt.plot(list(range(n + 1)), moyenne, label = 'moyenne des avis')
    # plt.legend(loc='upper left')

    plt.grid()
    plt.show()
```