MODÉLISATION D'UNE ÉPIDÉMIE: COVID-19

Est-il possible de prévoir le comportement d'une épidémie en utilisant des modèles mathématiques ?

Proposé par le candidat 9227

Plan:

Etude d'un modèle de percolation :

- Etablissement du modèle
- Simulation numérique du modèle
- Amélioration du modèle

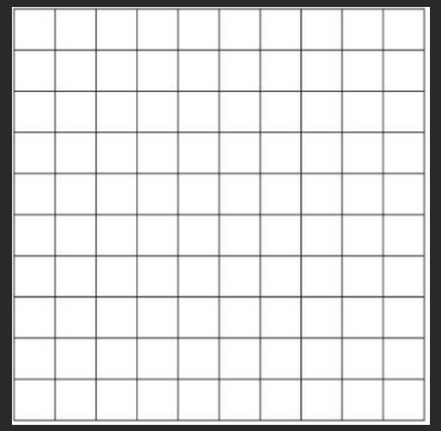
Etude du modèle SIR:

- Etablissement du modèle
- Résolution numérique
- Limitations du modèle

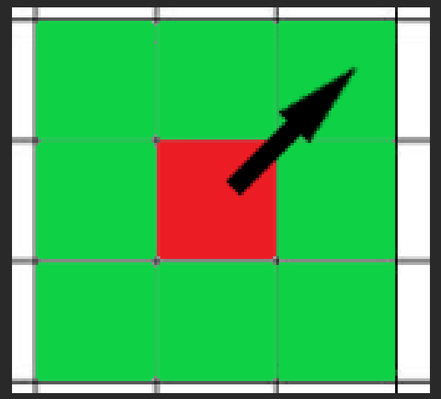
Comparaison des deux modèles

Conclusion

Modèle de percolation



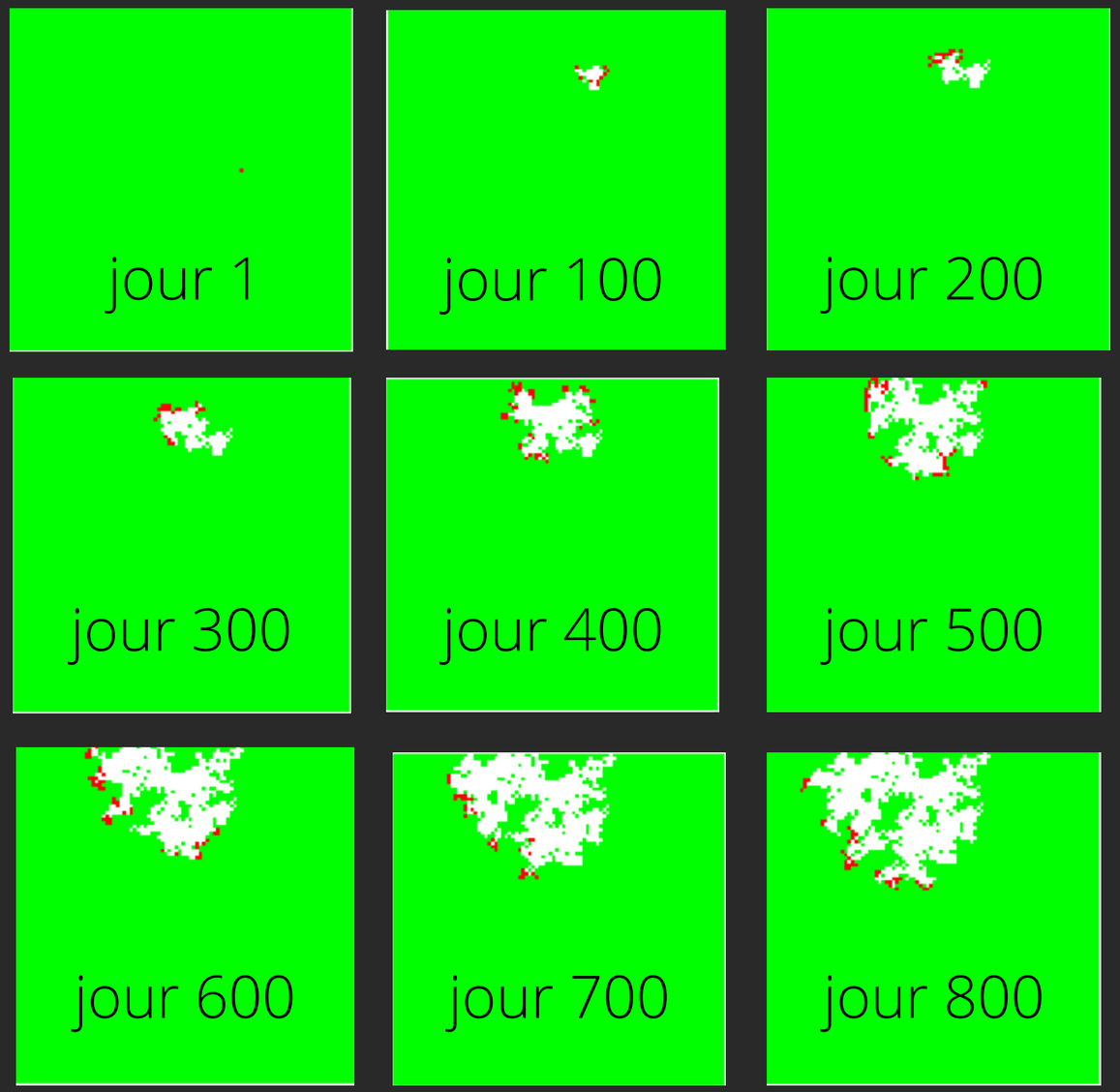
- On modélise la population à l'aide d'une grille carrée de taille n.
- Chaque élément de la grille représente une personne.
- On choisit au hazard une position où on place une personne inféctée et le reste de la grille sera susceptible.



- Chaque jour , chaque voisin d'une personne inféctée il ya au moins 7 jours a une probabilité p de devenir infécté.
- Chaque jour , chaque personne inféctée il y a au moins 14 jours a une probabilité q de se guérir.

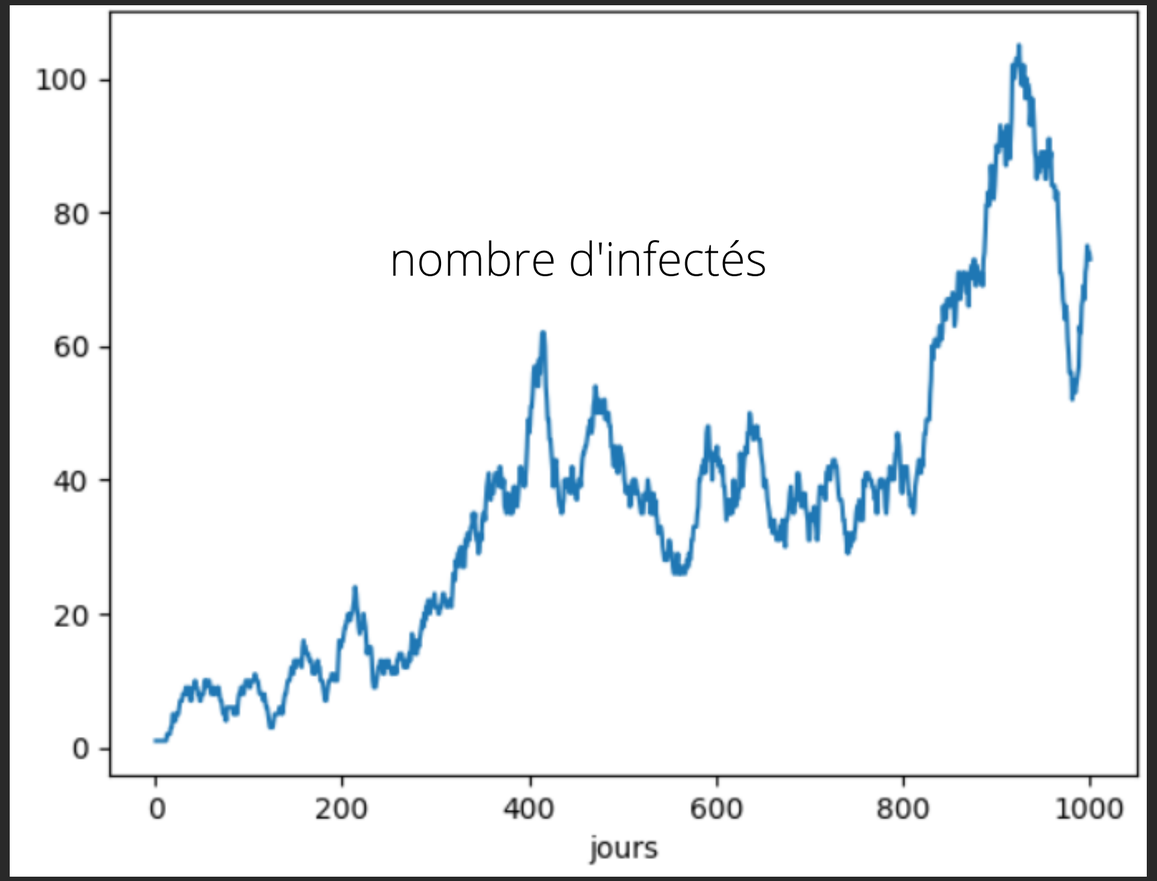
.

Résultats du modéle



Grille de taille 100x100

p=0.02 q=0.5



Amélioration du modèle

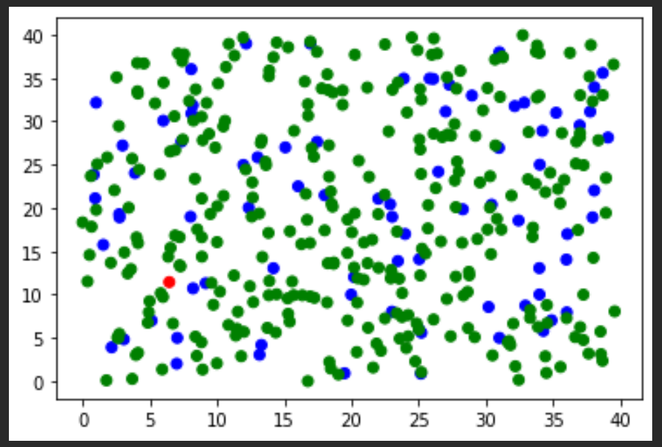
Limitations du modèle:

-Peut seulement être appliqué à des petits endroits fermés (école , navire ...)
-Ne tient pas compte du déplacement des individus .
-Ne tient pas compte des mesures sanitaires .
-La propagation est trop lente.

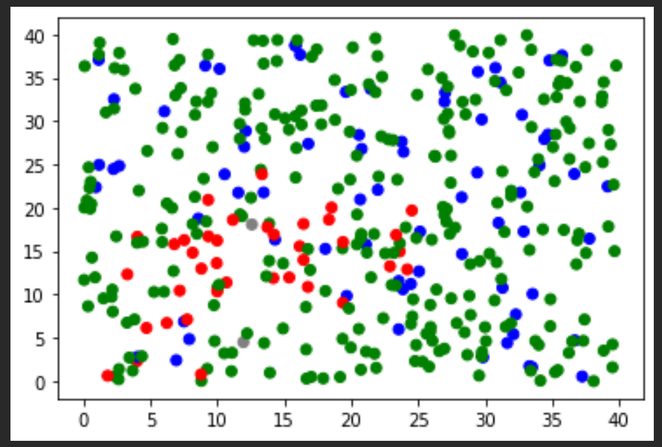
Pour améliorer le modèle on suppose que les individus se déplacent aléatoirement dans un espace carré .
Les personnes infféctés peuvent transmettre le virus avec une probabilité p lorsque il y a une personne susceptible proche.
Une partie de la population infectée est asymptomatique et a une probabilité reduite de transmettre le virus.

Résultats du nouveau modèle

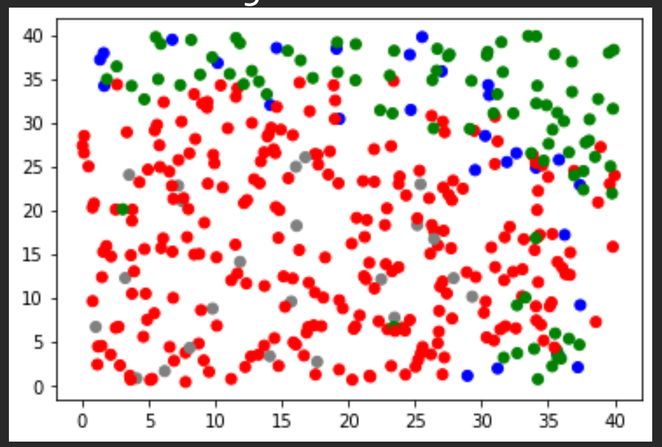
Sains **Sains asymptomatique** **Infectieux** **retirés**



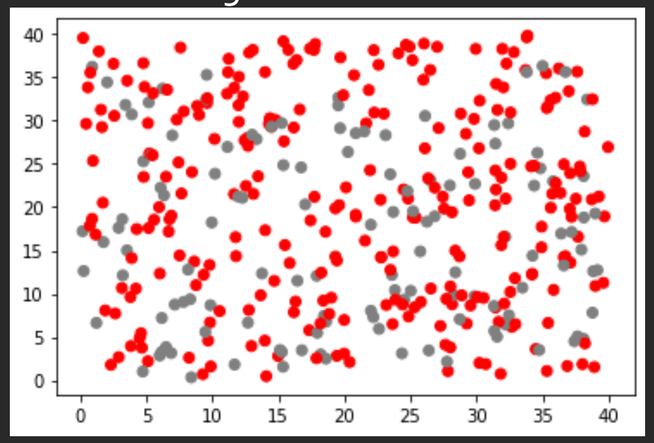
Jour 1



Jour 25

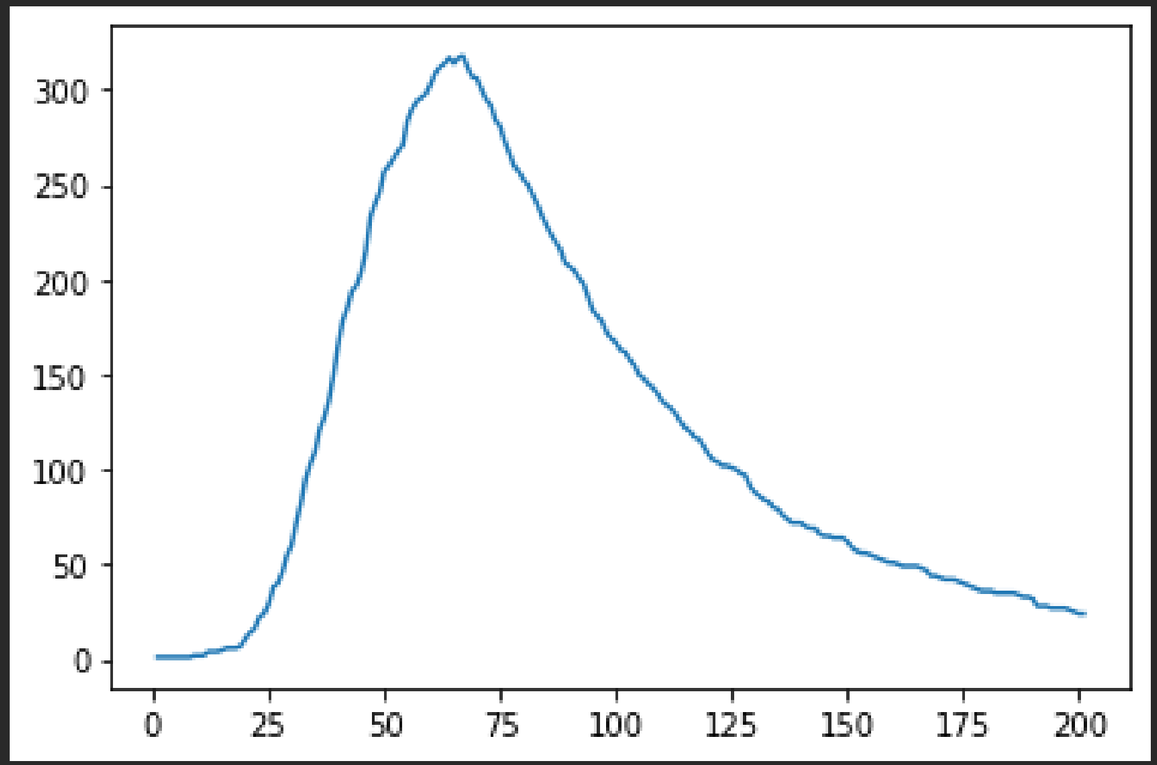


Jour 50

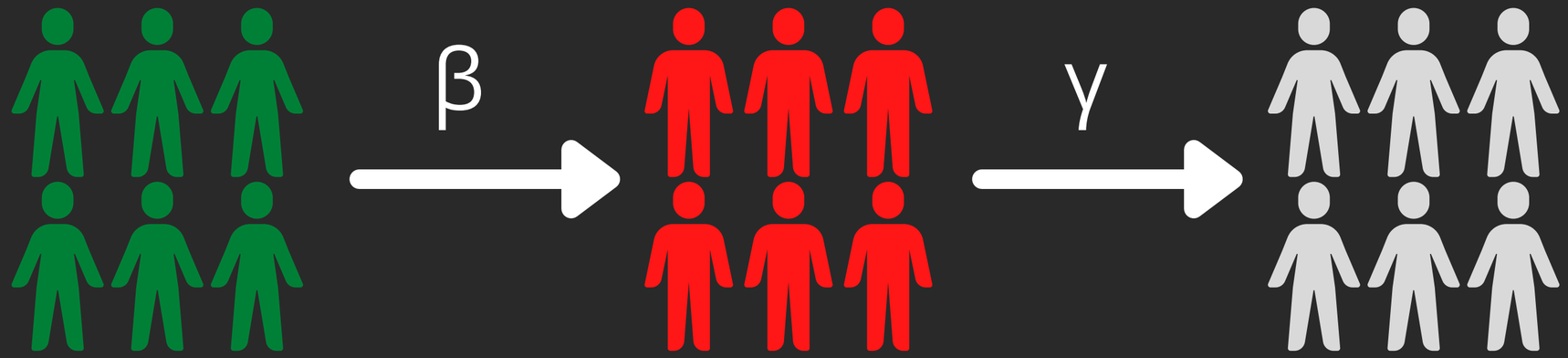


Jour 75

Nombre d'infectés au cours du temps



Modèle SIR



β : Taux de transmission du virus

γ: Taux de guérison

Le modèle SIR suppose que la population totale N est divisée en 3 catégories : personnes classées comme susceptibles d'être infectées (S), infectées (I) ou ne pouvant plus être infectées (R, pour retirées):

$$N(t)=S(t)+I(t)+R(t)=N(0)$$

-Les personnes susceptibles cesseront de l'être lorsqu'elles passent à l'état d'infection . Ils seront retirés proportionnellement à S , I et le taux de transmission β :

$$dS/dt= -\beta*I(t)*S(t)$$

-Les personnes infectées cesseront de l'être lorsqu'elles se guérissent avec un taux γ et l'effectif total accumulera ce qui sort de l'effectif S :

$$dI/dt=\beta*I(t)*S(t)-\gamma*I(t)$$

-L'effectif des personnes retirées accumulera ce qui sort de l'effectif de I :

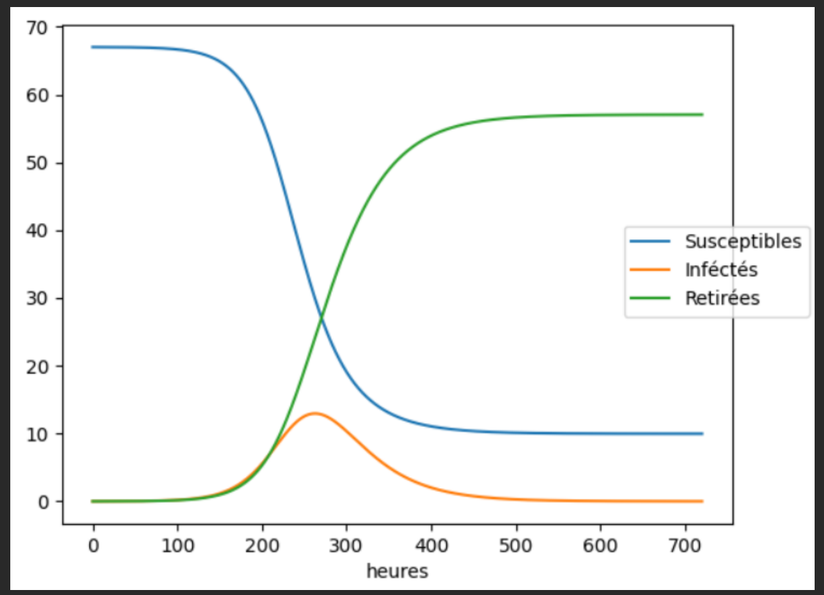
$$dR/dt=\gamma*I(t)$$

On aura finalement un systéme de 3 équations différentielles non linéaires qu'on va résoudre numériquement avec la méthode d'Euler explicite, on aura après discrétisation:

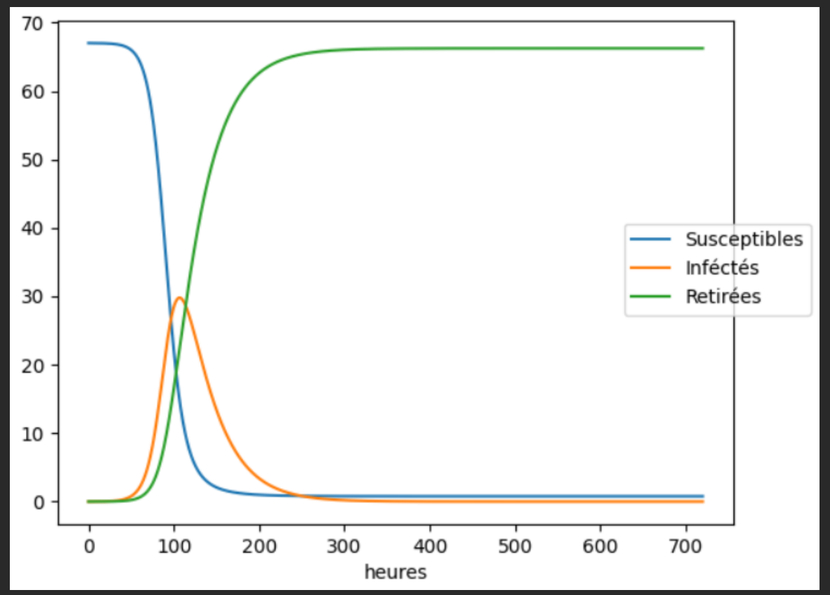
$S(n+1)=S(n)-\beta*I(n)*S(n)$ $I(n+1)=I(n)+\beta*I(n)*S(n)-\gamma*I(n)$ $R(n+1)=R(n)+\gamma*I(n)$

Résolution numérique des équations

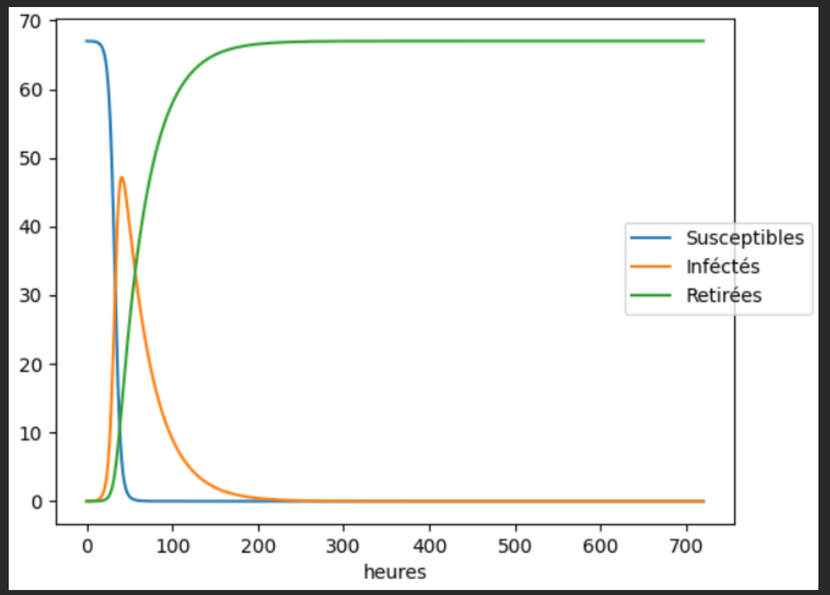
Résolution avec des valeurs différentes de β avec N(0)=67 millions (exemple de la France) , I(0)=5000 sur la durée de 30 jours:



β =0.001



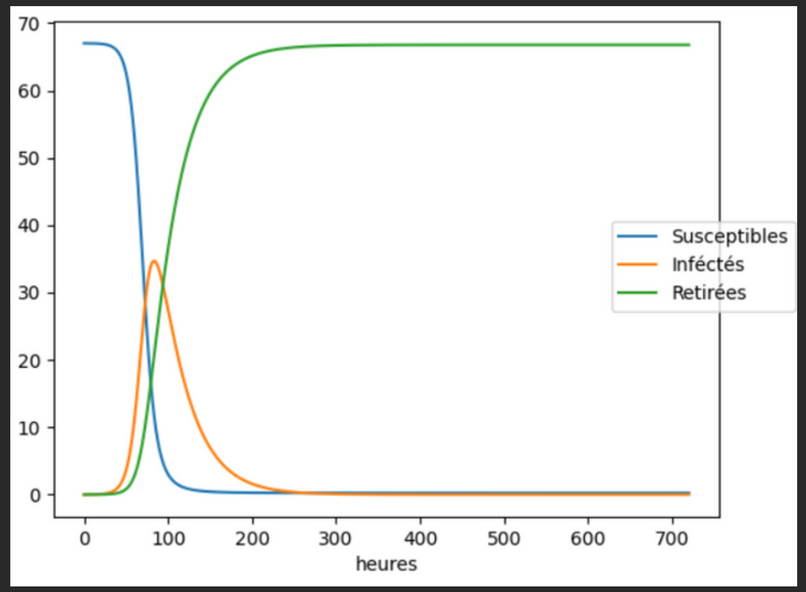
β =0.002



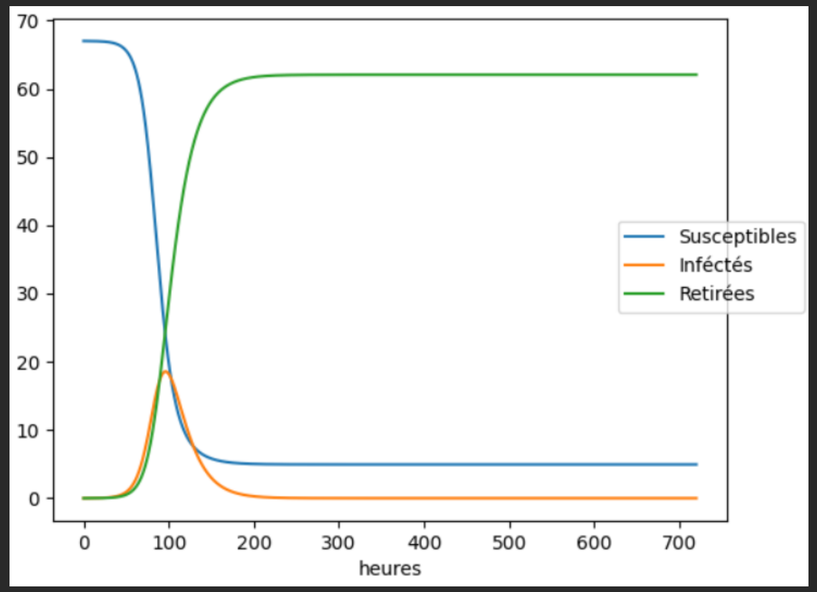
β =0.005

Résolution numérique des équations

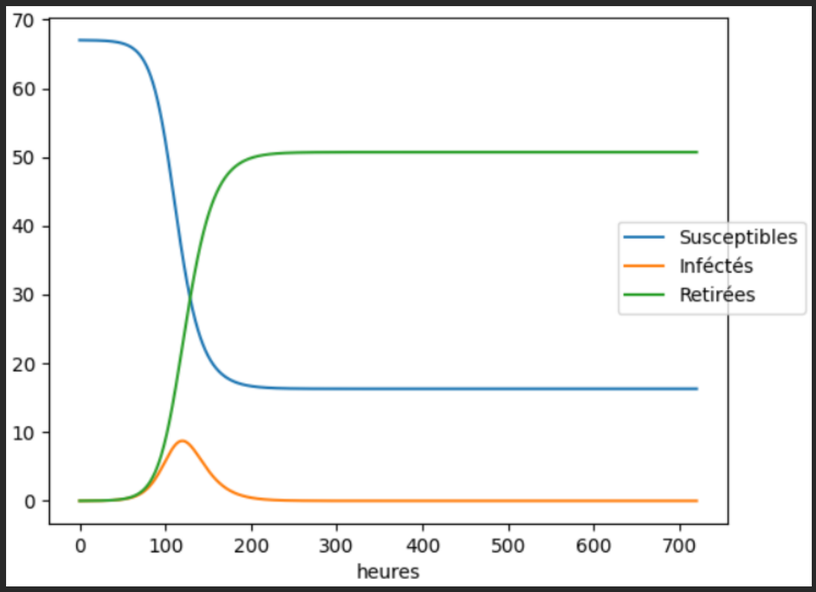
Résolution avec des différentes valeurs de γ :



γ=0.03



γ=0.06



γ=0.09

Limitations du modèle

On remarque que le nombre d'infféctés atteint un seul pic puis décroit ce qui n'est pas le cas dans la réalité.

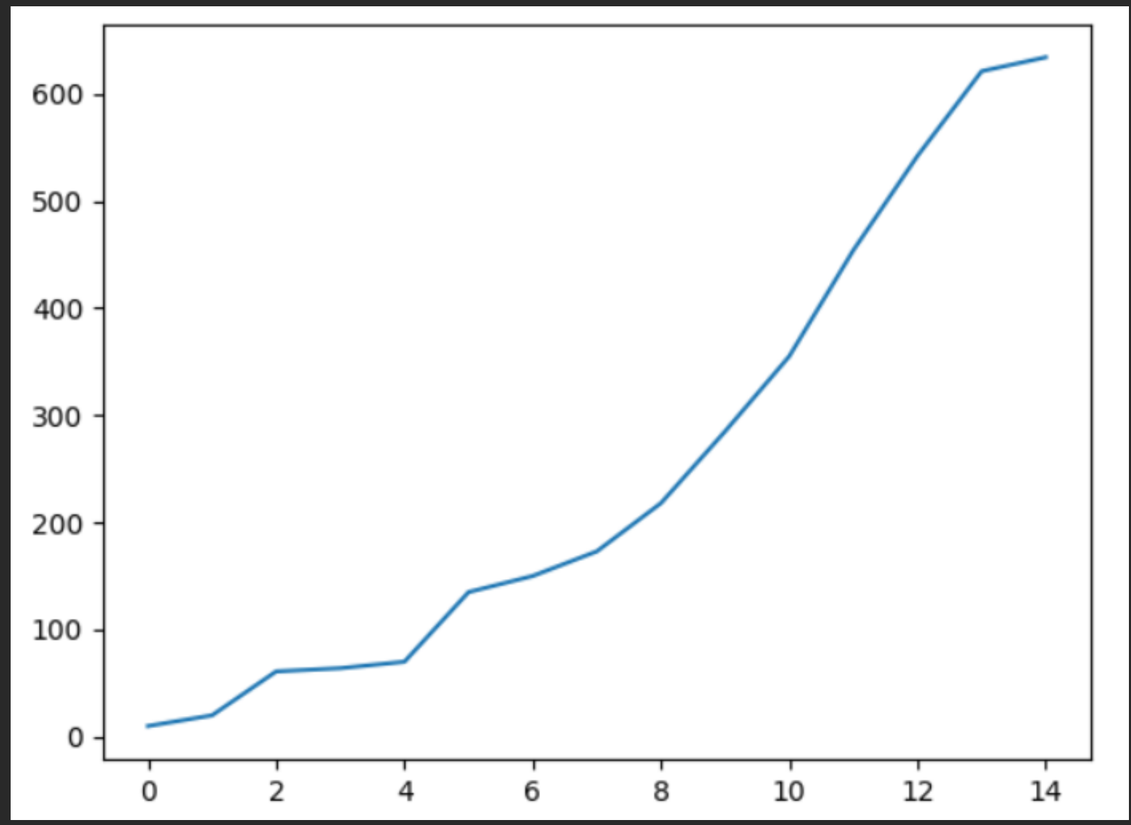
Les principaux problèmes du modèle sont la difficulté de trouver les taux de transmission et de guérison et qu'ils sont supposés constants.

Ces taux dépendent de plusieurs facteurs notamment les mesures sanitaires comme le port du masque , la distanciation sociale , les confinements , les vaccinations ...

Si on veut se rapprocher de la réalité les taux de transmission et de guérison seront des fonctions très complexes qui dépendent de plusieurs facteurs .

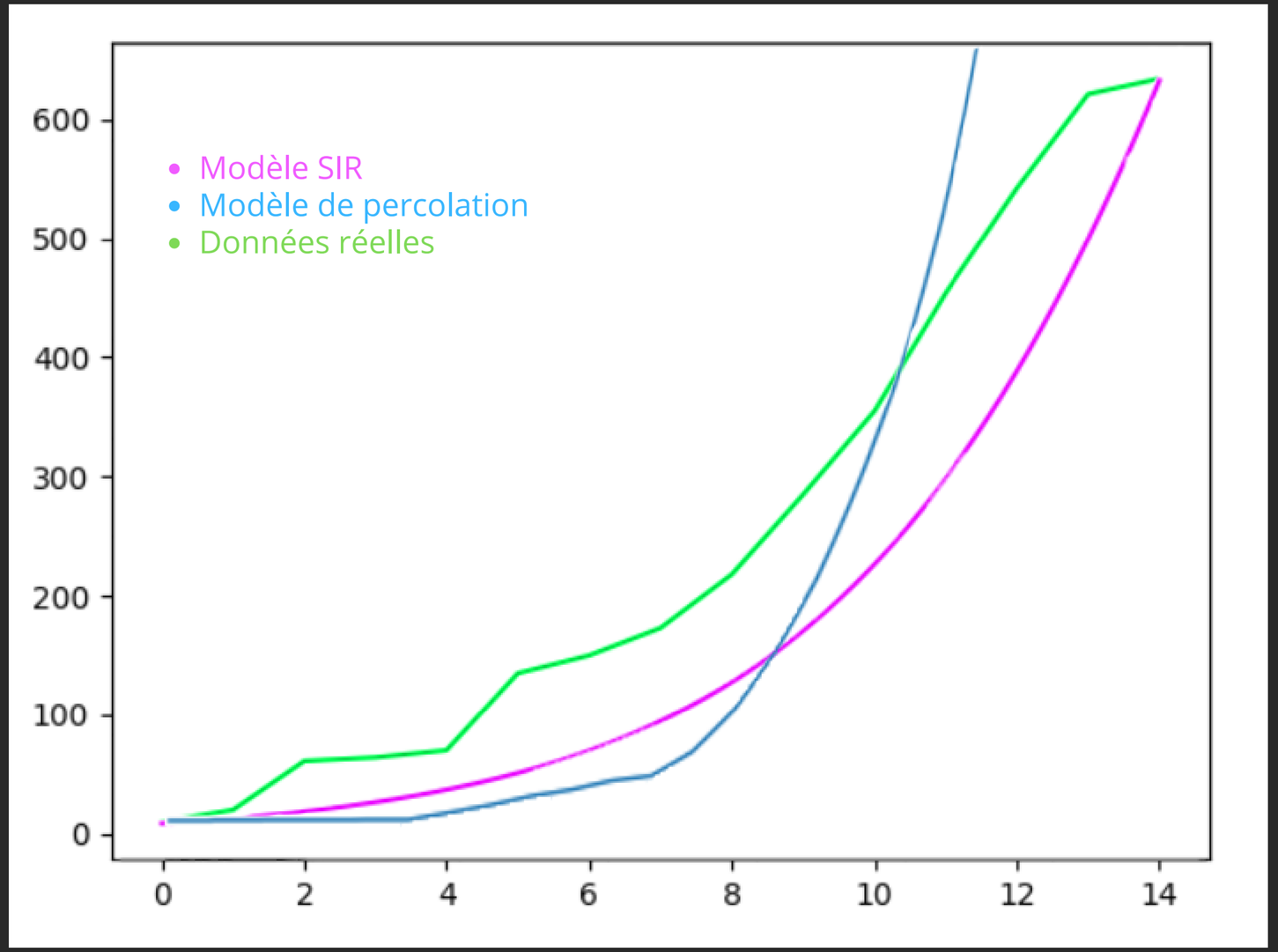
Comparaison des deux modèles appliqués à l'épidémie au bord du navire de croisière Diamond Princess

Durant une croisière qui a commencé en janvier 2020, des cas positifs de COVID-19 ont éte confirmés au bord du navire en février 2020 et un total de 712 sur 3711 ont été infféctés avec le virus



nombre de cas cumulatif entre le 5 et 19 février 2020

Comparaison des deux modèles appliqués à l'épidémie au bord du navire de croisière Diamond Princess



Comparaison des deux modèles appliqués à l'épidémie au bord du navire de croisière Diamond Princess

Pourtant que les modèles sont proches de la réalité sur cette courte période , les courbes cumulatif des modèles continuent à croitre comme ils ne tiennent pas compte des mécanismes qui bloquent la propagation du virus comme le confinement par exemple.

La situation sera plus compliquée si on se proposera de modéliser une épidémie dans un espace plus étendu comme une ville par exemple.

Cela explique le fait qu'il n'y a pas de modèle absolu dans l'épidémiologie et que les épidémiologistes améliorent des modèles déjà établis pour tenir compte de plusieurs facteurs négligés comme l'amélioration du modèle SIR pour obtenir des modèles dérivés de type SIIR , SEIR , SEIRS ...

Conclusion

- Chacun des deux modèles depend de deux variables principales (p et q pour le modèle de percolation/ β et γ pour le modèle SIR) qui sont impossible à determiner à l'aide d'une étude purement mathématique.
- Ces variables dépendent de plusieurs facteurs comme le port du masque , la distanciation sociale , les confinements et les différents variants du virus.
- Pour comprendre le comportement d'une épidémie , il faut tout d'abord bien comprendre le comportement du virus responsable (médicine) , le comportement des individus (étude sociale) et les décisions politiques qui est loin d'une étude mathématique.
- L'étude d'une épidémie est donc une étude multidisciplinaire et ne peut pas être réduite à une simple étude mathématique.

ANNEXE

Euler explicite modèle SIR

```
from numpy import zeros, linspace
import matplotlib.pyplot as plt

# Unité de temps: 1 heure
beta = 0.0025
gamma = 0.06
dt = 0.5            # 30 minutes
D = 30              # Simulation pour D jours
N_t = int(D*24/dt)   # Nombre d'heures correspondant

t = linspace(0, N_t*dt, N_t+1)
S = zeros(N_t+1)
I = zeros(N_t+1)
R = zeros(N_t+1)

# Condition initiale
S[0] = 67
I[0] = 0.005
R[0] = 0


for n in range(N_t):
    S[n+1] = S[n] - dt*beta*S[n]*I[n]
    I[n+1] = I[n] + dt*beta*S[n]*I[n] - dt*gamma*I[n]
    R[n+1] = R[n] + dt*gamma*I[n]

fig = plt.figure()
l1, l2, l3 = plt.plot(t, S, t, I, t, R)
fig.legend((l1, l2, l3), ('Susceptibles', 'Infféctés', 'Retirées'),'center right')
plt.xlabel('heures')

plt.show()
```

ANNEXE

Modèle de percolation

```
from numpy import *
from random import *
import matplotlib.pyplot as plt

def population_avant_virus(n):
    A=zeros((n,n),dtype=[('x', 'int'), ('y', 'int')])
    return A

def creer_infect(A,n):
    I=[]
    i=randint(1,n-2)
    j=randint(1,n-2)
    A[i,j]=(1,0)
    I.append((i,j))
    return I

def condition_limite(A,n):
    A[0,:]=(2,0)
    A[:,0]=(2,0)
    A[n-1,:]=(2,0)
    A[:,n-1]=(2,0)

def transmission(A,I,i,j,p):
    for k in range(-1,2):
        for l in range(-1,2):
            if A[k+i,j+l][0]==0:
                if (random()<p):
                    A[k+i,j+l][0]=1
                    I.append((k+i,j+l))

def guerison(A,I,i,j,p):
    if random()<p:
        A[i,j][0]=2
        I.remove((i,j))
```

```
def Apres_trans(n,p,q,t,dt):
    A=population_avant_virus(n)
    condition_limite(A,n)
    I=creer_infect(A,n)
    L=[A]
    V=[1]
    for j in range(1,t):
        for y in I:
            (i,j)=y
            A[i,j][1]+=1
            if A[i,j][1]>=7 :
              transmission(A,I,i,j,p)
            if A[i,j][1]>=20 :
              guerison(A,I,i,j,q)
        if j%dt==0:
            B=copy(A)
            L.append(B)
        V.append(len(I)+len(L))
    return L,V
def image(B):
    c=shape(B)
    n=c[0]
    A=zeros((n,n,3),dtype=uint8)
    for i in range(n):
        for j in range(n):
            if B[i,j][0]==0:
                A[i,j]=(0,255,0)
            elif B[i,j][0]==1:
                A[i,j]=(255,0,0)
            else:
                A[i,j]=(255,255,255)
    imgplot = plt.imshow(A)
    plt.show()
def plot(V):
    T=[]
    for i in range (len(V)):
        T.append(i+1)
    plt.plot(T,V)
    plt.xlabel('jours')
    plt.show()
```

ANNEXE

Modèle de percolation amélioré

```python
import numpy as np
import matplotlib.pyplot as plt

def distance(coord_1,coord_2):
    return np.sqrt((coord_1[0]-coord_2[0])**2 + (coord_1[1]-coord_2[1])**2)

def creer_infect(limites,n):
    I=[]
    i=np.random.randint(1,n-2)
    j=np.random.randint(1,n-2)
    I.append(individu((i,j), 1, limites, np.random.random()))
    return I

def creer_assymptomatique(n,limites):
    Asp=[]
    for i in range(int(n*n*0.2)):
        i=np.random.randint(1,limites[0])
        j=np.random.randint(1,limites[1])
        Asp.append(individu((i,j), 3, limites, np.random.random()))
    return Asp

def guerison(ind,I,p):
    if np.random.random()<p:
        ind.etat = 2
        I.remove(ind)

class individu:

    def __init__(self,coords,etat,limites,vitesse):
        self.coord = coords
        self.etat = etat
        self.limites = limites
        self.vitesse = vitesse
        self.temps = 0

    def maj_direction(self):
        angle = np.random.uniform(0,2*np.pi)
        x_direction = np.cos(angle)
        y_direction = np.sin(angle)
        self.x_direction = x_direction
        self.y_direction = y_direction

    def maj_location(self):
        vitesse = self.vitesse
        limites = self.limites
        distance = vitesse*np.random.random()
        if self.coord[0] + distance*self.x_direction < 0:
            new_x = -distance*self.x_direction-self.coord[0]
            self.x_direction = -self.x_direction
        elif self.coord[0] + distance*self.x_direction > limites[0]:
            new_x = 2*limites[0]-distance*self.x_direction - self.coord[0]
            self.x_direction = -self.x_direction
        else:
            new_x = self.coord[0] + distance*self.x_direction

        if self.coord[1] + distance*self.y_direction < 0:
            new_y = -distance*self.y_direction-self.coord[1]
            self.y_direction = -self.y_direction
        elif self.coord[1] + distance*self.y_direction > limites[1]:
            new_y = 2*limites[1]-distance*self.y_direction - self.coord[1]
            self.y_direction = -self.y_direction
        else:
            new_y = self.coord[1] + distance*self.y_direction

        self.coord = (new_x,new_y)


    def transmission(self,I,autres,rayon,p,q):

        if (self.etat == 1):
            self.temps +=1
            if self.temps >= 7:
                for autre in autres:
                    if distance(self.coord, autre.coord)<=rayon and (autre.etat == 0):
                        if np.random.random()<p:
                            autre.etat = 1;
                            I.append(autre)
                    if distance(self.coord, autre.coord)<=rayon and (autre.etat == 3):
                        if np.random.random()<p:
                            autre.etat = 31;
                            I.append(autre)
            if self.temps >= 14:
                guerison(self,I, p)
        if (self.etat == 31):
            if self.temps >= 7:
                for autre in autres:
                    if distance(self.coord, autre.coord)<=rayon and (autre.etat == 0):
                        if np.random.random()<q:
                            autre.etat = 1;
                            I.append(autre)
                    if distance(self.coord, autre.coord)<=rayon and (autre.etat == 3):
                        if np.random.random()<q:
                            autre.etat = 31;
                            I.append(autre)
            if self.temps >= 14:
                guerison(self,I, p)


def Apres_trans(n,p,rayon,q,t,pa):
    limites = (40,40)
    I=creer_infect(limites, n)
    population = []
    population.append(I[0])
    Asp = creer_assymptomatique(n,limites)
    population.extend(Asp)
    m=len(population)
    for n in range(n*n-m):
        population.append(individu((40*np.random.random(),40*np.random.random()),
                                   0,limites,np.random.random()))
    for n in range(0,len(population)):
        population[n].maj_direction()
    plt.xlim(0,100)
    plt.ylim(0,100)
    V=[1]
    L=[]

    for n in range(t):
        coords = []
        etats = []
        for personne in population:
            coords.append(personne.coord)
            if (personne.etat == 1 or personne.etat == 31):
                etats.append('red')
            elif personne.etat == 0:
                etats.append('green')
            elif personne.etat == 2:
```

ANNEXE

Modèle de percolation amélioré

```
            etats.append('grey')
          elif personne.etat == 3:
            etats.append('blue')
        etats = np.array(etats)
        coords = np.array(coords)
        L.append((coords[:,0],coords[:,1],etats))
        for i in range(0,len(population)):
          population[i].transmission(I,population[:i]+population[i+1:],rayon,p,pa)
        for i in range(0,len(population)):
          population[i].maj_location()
        V.append(len(I))
    return L,V

def plot(V):
    T=[]
    for i in range (len(V)):
        T.append(i+1)
    plt.plot(T,V)
    plt.show()
def show(L,i):
    plt.scatter(L[i][0],L[i][1],c=L[i][2])
```

ANNEXE

Cas confirmés au bord de Diamond Princess

Confirmed cases on *Diamond Princess* (v · t · e)

Date (JST)	Tested (cumulative)	Confirmed (cumulative)	Notes and ref(s)
3 February			Berthed at the Port of Yokohama
5 February	31	10	[99]
6 February	102	20	Calculated from reports[99][100]
7 February	273	61	[100]
8 February	279	64	[101]
9 February	336	70	[102]
10 February	439	135	[103]
12 February	492	174	Calculated from reports[103][104]
13 February	713	218	[104]
15 February	930	285	Includes 73 asymptomatic cases[105]
16 February	1,219	355	Includes 111 asymptomatic cases[106]
17 February	1,723	454	Includes 189 asymptomatic cases[38]
18 February	2,404	542	Includes 254 asymptomatic cases[107]
19 February	3,011	621	Includes 322 asymptomatic cases[108]
20 February	3,063	634	Includes 328 asymptomatic cases[109]

ANNEXE

Modèle de percolation amélioré version 2

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

def distance(coord_1,coord_2):
    return np.sqrt((coord_1[0]-coord_2[0])**2 + (coord_1[1]-coord_2[1])**2)

def creer_infect(limites,n):
    I=[]
    for k in range (n):
      i=np.random.randint(1,limites[0])
      j=np.random.randint(1,limites[1])
      I.append(individu((i,j), 1, limites, np.random.random()))
    return I

def creer_assymptomatique(n,limites):
    Asp=[]
    for i in range(int(n*0.2)):
        i=np.random.randint(1,limites[0])
        j=np.random.randint(1,limites[1])
        Asp.append(individu((i,j), 3, limites, np.random.random()))
    return Asp

def guerison(ind,I,p):
    if np.random.random()<p:
        ind.etat = 2
        I.remove(ind)

class individu:

    def __init__(self,coords,etat,limites,vitesse):
        self.coord = coords
        self.etat = etat
        self.limites = limites
        self.vitesse = vitesse
        self.temps = 0
```

```python
def maj_location(self):
    vitesse = self.vitesse
    limites = self.limites
    distance = vitesse*np.random.random()
    if self.coord[0] + distance*self.x_direction < 0:
        new_x = -distance*self.x_direction-self.coord[0]
        self.x_direction = -self.x_direction
    elif self.coord[0] + distance*self.x_direction > limites[0]:
        new_x = 2*limites[0]-distance*self.x_direction - self.coord[0]
        self.x_direction = -self.x_direction
    else:
        new_x = self.coord[0] + distance*self.x_direction

    if self.coord[1] + distance*self.y_direction < 0:
        new_y = -distance*self.y_direction-self.coord[1]
        self.y_direction = -self.y_direction
    elif self.coord[1] + distance*self.y_direction > limites[1]:
        new_y = 2*limites[1]-distance*self.y_direction - self.coord[1]
        self.y_direction = -self.y_direction
    else:
        new_y = self.coord[1] + distance*self.y_direction

    self.coord = (new_x,new_y)


def transmission(self,I,autres,rayon,p,q):

    ni = 0
    if (self.etat == 1):
        self.temps +=1
        if self.temps >= 7:
            for autre in autres:
                if distance(self.coord, autre.coord)<=rayon and (autre.etat == 0):
                    if np.random.random()<p:
                      autre.etat = 1
                      ni+=1
                      I.append(autre)
                if distance(self.coord, autre.coord)<=rayon and (autre.etat == 3):
                    if np.random.random()<p:
                      autre.etat = 31
                      ni+=1
                      I.append(autre)
        if self.temps >= 14:
            guerison(self,I, p)
    if (self.etat == 31):
        self.temps +=1
        if self.temps >= 7:
            for autre in autres:
                if distance(self.coord, autre.coord)<=rayon and (autre.etat == 0):
                    if np.random.random()<q:
                      autre.etat = 1
                      ni+=1
                      I.append(autre)
                if distance(self.coord, autre.coord)<=rayon and (autre.etat == 3):
                    if np.random.random()<q:
                      autre.etat = 31
                      ni+=1
                      I.append(autre)
        if self.temps >= 14:
            guerison(self,I, p)
    return ni
```

ANNEXE

Modèle de percolation amélioré version 2

```
def Apres_trans(n,ni,p,rayon,q,t,pa):
    limites = (40,40)
    I=creer_infect(limites,ni)
    population = []
    population.extend(I)
    Asp = creer_assymptomatique(n,limites)
    population.extend(Asp)
    m=len(population)
    for n in range(n-m):
        population.append(individu((40*np.random.random(),40*np.random.random()),
                                   0,limites,np.random.random()))
    for n in range(0,len(population)):
       population[n].maj_direction()
    plt.xlim(0,100)
    plt.ylim(0,100)
    V=[1]
    L=[]
    nc=1
    for n in range(t):
       coords = []
       etats = []
       for personne in population:
         coords.append(personne.coord)
         if (personne.etat == 1 or personne.etat == 31):
           etats.append('red')
         elif personne.etat == 0:
           etats.append('green')
         elif personne.etat == 2:
           etats.append('grey')
         elif personne.etat == 3:
           etats.append('blue')
       etats = np.array(etats)
       coords = np.array(coords)
       L.append((coords[:,0],coords[:,1],etats))
       for i in range(0,len(population)):
        nc+= population[i].transmission(I,population[:i]+population[i+1:],rayon,p,pa)
       for i in range(0,len(population)):
         population[i].maj_location()
       V.append(nc)
    return L,V
```

```
def plot(V):
    T=[]
    for i in range (len(V)):
        T.append(i+1)
    plt.plot(T,V)
    plt.show()
def show(L,i):
    plt.scatter(L[i][0],L[i][1],c=L[i][2])
```