Filière MP - ENS de Paris-Saclay, Lyon, Rennes et Paris - Session 2022

Page de garde du rapport de TIPE

NOM : Borjot	Prénoms : Arthur, Alan
Classe : MP*	
Lycée : Blaise Pascal	Numéro de candidat : 49859
Ville : Clermont-Ferrand	

Concours auxquels vous êtes admissible, dans la banque MP inter-ENS (les indiquer par une croix) :

ENS Cachan	MP - Option MP		MP - Option MPI	
	Informatique	X		
ENS Lyon	MP - Option MP		MP - Option MPI	
	Informatique - Option M	X	Informatique - Option P	
ENS Rennes	MP - Option MP		MP - Option MPI	
	Informatique	X		
ENS Paris	MP - Option MP		MP - Option MPI	
	Informatique	X		

Matière dominante du TIPE (la sélectionner d'une croix inscrite dans la case correspondante) :

Informatique	X	Mathématiques		Physique	

Titre du TIPE : Simulateur de maladie par automate cellulaire

Nombre de pages (à indiquer dans les cases ci-dessous) :

Texte	5	Illustration	4	Bibliographie	1

Attention, les illustrations doivent figurer dans le corps du texte et non en fin du document !

Résumé ou descriptif succinct du TIPE (6 lignes, maximum) :

J'ai crée un simulateur utilisant plusieurs automates cellulaires pour représenter à petite échelle la propagation d'une maladie. J'ai amélioré le système pour tester des mesures préventives pour freiner la maladie. J'ai produit et analysé de nombreux résultats sous différentes formes. J'ai réalisé un calcul de position d'équilibre dans ce système probabiliste.

À Clermont-Ferrand

Le 13/06/2022

Signature du (de la) candidat(e)

Borjot

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

BLAISE-PASCAL 04.73.98.36.50 Avenue Carnot 63037 CLERMONT-FD CEDEX

La signature du professeur responsable et le tampon de l'établissement ne sont pas indispensables pour les candidats libres (hors CPGE).

Simulateur de maladie par automate cellulaire

Arthur Barjot

Mai 2022

Table des matières

1 Introduction

1.1 Simulation épidémique

Le premier modèle de maladie infectieuse a été créé par Bernoulli en 1766, avec la variole, cette science est donc plutôt ancienne. Actuellement, et plus que jamais, les modèles épidémiologiques peuvent avoir un réel impact sur nos choix de gestion de crise. Plus généralement, les modèles ont différents objectifs : décrire des dynamiques épidémiques, étudier des paramètres qui sont cachés de l'observation directe, planifier ou tester et optimiser des plans d'expérience (tester des actions de santé publique, par exemple : le confinement va-t-il avoir une influence sur l'évolution de l'épidémie ?).

Nous avons donc créé deux simulateurs de propagation de maladie, l'un utilisant la méthode dite SIR, basé sur des équations différentielles. Et l'autre utilisant un automate cellulaire, définnissant une maladie par des fonctions précises et par le choix de probabilités (taux de contagiosité, mortalité,...).

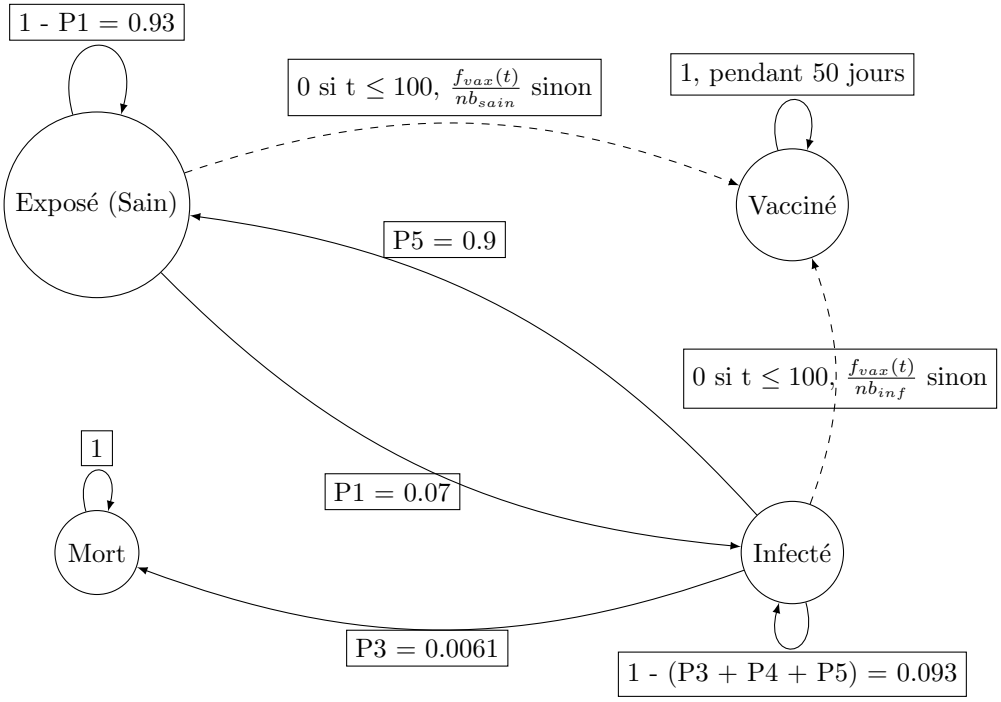
Personnellement je me suis chargé de la partie automate cellulaire, je vais donc présenter ce point. Par manque de temps, mon camarade ayant produit des résultats trop tardivement, il n'y aura pas de comparaison des modèles.

1.2 Automate cellulaire

Un automate cellulaire peut être représenté par une grille de « cellules », chacune caractérisée par un «état», les états sont en nombre fini. Cette grille peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est fonction de son état au temps t ainsi que de celui de ses cellules « voisines ». À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle « génération » de cellules.

Pour nous, l'automate cellulaire sera une matrice de cellules, chaque cellule est une case contenant ou non un individu. Ainsi on représente une ville par un automate cellulaire, si l'automate est de dimensions $n \times n$ (dimensions de la matrice) alors la ville aura n^2 cellules caractérisées par un des 5 "états" possibles. La cellule peut être vide (c'est le premier état), ou elle peut contenir un individu. Cet individu est sain, infecté, vacciné ou mort (les 4 autres états). L'état d'une personne peut changer grâce aux règles d'évolution de l'automate, et avec l'ajout de déplacements au modèle, une cellule peut accueillir différentes personnes au cours d'une simulation. Pour ce qui concerne les déplacements de personnes dans une ville, je parlerai de trajets en "bus". Pour les trajets d'une ville à une autre, je parlerai de trajets en "train".

Voici le graphe des principales transitions d'états dans notre automate cellulaire, réalisé avec les chiffres que j'ai choisi pour simuler le covid-19 :



2 Le simulateur

2.1 Présentation générale

Je suis plusieurs fois revenu sur mon programme pour l'améliorer, mais je vais maintenant présenter la version finale obtenue (voir le code en annexe). Mon simulateur utilise n automates cellulaires qui représentent n villes. Les villes sont de différentes tailles, les habitants de chaque ville se déplacent dans leur ville, et peuvent aussi se déplacer dans le pays vers une autre ville selon la matrice de déplacement présentée juste après.

Au début de la simulation, une personne aléatoire est sélectionnée pour être contaminée. La maladie se propage ensuite chaque jour, les voisins d'un malade ont une certaine probabilité d'être contaminés. Les malades peuvent redevenir sains après une durée d de maladie, ils peuvent aussi mourir ou rester infectés encore quelques jours avec différentes probabilités.

À partir d'un certain seuil de contaminés dans une ville, celle-ci est confinée pendant deux semaines, c'est à dire que ses habitants se déplacent beaucoup moins et deviennent moins contagieux. À l'échelle du pays également, à partir d'un certain taux de contaminés, toutes les villes sont confinées pendant 30 jours avec également une diminution des trajets entre villes.

Après 100 jours de propagation de la maladie, le pays a réussi à développer un vaccin. On utilise une fonction de vaccination qui à une date t associe le

nombre de personnes qui vont se faire vacciner au jour t. Pour ce qui est de la présentation des résultats, j'ai pu produire des cartes représentant l'état de la population d'une ville (voir partie suivante). J'ai aussi produit des graphiques d'évolution du nombre de personnes dans les différents états en fonction du temps.

2.2 Trajet entre villes

Nous considérons que chaque jour, dans chaque ville une même portion de personnes se déplace en train. Notons p cette portion, P le pays, et V_i la ville numéro i. On notera $|P|$ et $|V_i|$ respectivement les nombres d'habitants du pays et celui de la ville i. Nous allons maintenant tenter de simuler au mieux les déplacements entre différentes villes. Pour cela, on considère la matrice de déplacement :

$$matdep = \begin{pmatrix} m_{1,1} & ... & m_{1,n} \\ \vdots & (*) & \vdots \\ m_{n,1} & ... & m_{n,n} \end{pmatrix}$$

Le coefficient $m_{i,j}$ représente le nombre de personnes se déplaçant de la ville i vers la ville j chaque jour. On a les relations :

$$\forall i \in [\![1,n]\!], \sum_{j=1}^{n} m_{i,j} = \sum_{j=1}^{n} m_{j,i} = p \times |V_i|$$

En effet, la première somme représente le nombre de personnes quittant la ville i chaque jour, elle vaut donc $p \times |V_i|$. La deuxième somme représente le nombre de personnes arrivant à la ville i chaque jour. Pour que nos automates ne saturent pas, on considère qu'il y a autant d'entrées que de sorties pour chaque ville.

On veut tenir compte de la taille des villes, car une grande ville va être plus attractive chaque jour qu'un village. D'où ma proposition de solution :

$$\forall i,j \in [\![1,n]\!]^2, m_{i,j} = \frac{p \times |V_i| \times |V_j|}{|P|}$$

Cette solution vérifie les propriétés :

$$\forall i,j \in [\![1,n]\!]^2, \sum_{j=1}^{n} \frac{p \times |V_i| \times |V_j|}{|P|} = \sum_{j=1}^{n} \frac{p \times |V_j| \times |V_i|}{|P|} = \frac{p \times |V_i|}{|P|} \times \sum_{j=1}^{n} |V_j| = p \times |V_i|$$

2.3 Calcul d'une position d'équilibre

Pour le premier modèle que j'ai créé : sans confinement ni vaccin, avec des villes toutes de même taille. Je remarquais que toutes mes simulations semblaient converger vers une position d'équilibre, les nombres de personnes saines et infectées semblaient constants. Je vais maintenant trouver cet équilibre.

Considérons un pays de 9 villes, dont chaque ville comporte $100 \times 100 = 10000$ habitants, et un taux de cases vides de 0.07. Pour simplifier on oublie l'état

mort. On considère qu'une personne vulnérable (voisine d'une case infectée) a une probabilité de 0.05 de devenir infectée (il s'agit de la valeur utilisée pour produire le graphique présenté à la fin du paragraphe). Après 7 jours de maladie, une personne infectée a une probabilité de 0.9 de redevenir saine.

Supposons que le système est à l'équilibre : les taux de personnes infectées et de personnes saines sont devenus constants, on peut considérer que c'est le cas aussi dans toutes les villes : les personnes infectées arrivent autant qu'elles quittent une ville. Les villes sont régies par les mêmes probabilités, sont de même taille, donc on peut considérer que leurs conditions sont identiques à l'équilibre. Le taux de personnes infectées d'une des villes est donc égal au taux à l'échelle du pays. On se place donc maintenant dans une ville.

À l'équilibre, les taux x et (1-x) de personnes infectées et de personnes saines sont invariants. On en déduit que chaque jour, autant de personnes deviennent saines, qu'infectées. On considère aussi que le nombre de nouveaux infectés chaque jour est constant. On se place à la date t, longtemps après que l'équilibre soit en place. Ainsi à la date $t-7$, le système était à l'équilibre.

Quelle portion de la population devient infectée à la date t ?

Trouvons le taux d'infectés parmi les voisins d'une case quelconque : la case peut être sur un coin de la ville ou sur un bord. Son nombre de voisins est donc (on rappelle que notre ville contient 10 000 habitants) :

$$\frac{4 \times 3 + 396 \times 5 + 9604 \times 8}{10000} = 7.88$$

Ensuite il ne faut pas oublier les cases vides, il y en a un taux de 0.07, donc le nombre réel de voisins est plutôt $7.88 \times (1 - 0.07) = 7.25$. Le nombre de voisins infectés d'une case donnée est $x \times 7.25$, et la probabilité pour une case saine de devenir infectée est $x \times 7.25 \times 0.05$. Le nombre de personnes saines est $(1 - x)$. Au final, la portion de la population qui devient infectée est :

$$(1 - x) \times 0.05 \times 7.25x = 0.36x - 0.36x^2$$

Quelle portion de la population devient saine à la date t ?

Pour cela on cherche la part de la population pouvant devenir saine, c'est à dire le nombre de personnes infectées depuis plus de 7 jours. Notons ce nombre u_t, il est indépendant de t. La part de personnes devenues saines à la date t-1 est $0.9 \times u_{t-1}$, donc la part d'infectées depuis plus de 8 jours à la date t est $0.1 \times u_{t-1}$. Le nombre de personnes infectées depuis 7 jours ou moins vaut $x - 0.1 \times u_{t-1}$. Les personnes infectées depuis exactement 7 jours sont en même nombre que celles infectées depuis exactement 6 jours, 5 jours, ... car pour $d \in [\![0, 6]\!]$, les personnes infectées depuis d jours exactement à la date t, sont exactement celles qui le seront depuis d+1 à la date t+1. On en déduit que le nombre d'infectés depuis exactement 7 jours vaut $\frac{x - 0.1 \times u_{t-1}}{7}$ D'où la relation :

$$u_t = 0.1 \times u_{t-1} + \frac{x - 0.1 \times u_{t-1}}{7} = \frac{3}{35} \times u_{t-1} + \frac{x}{7}$$

Or on a dit plus tôt que $u_{t-1} = u_t$, d'où :

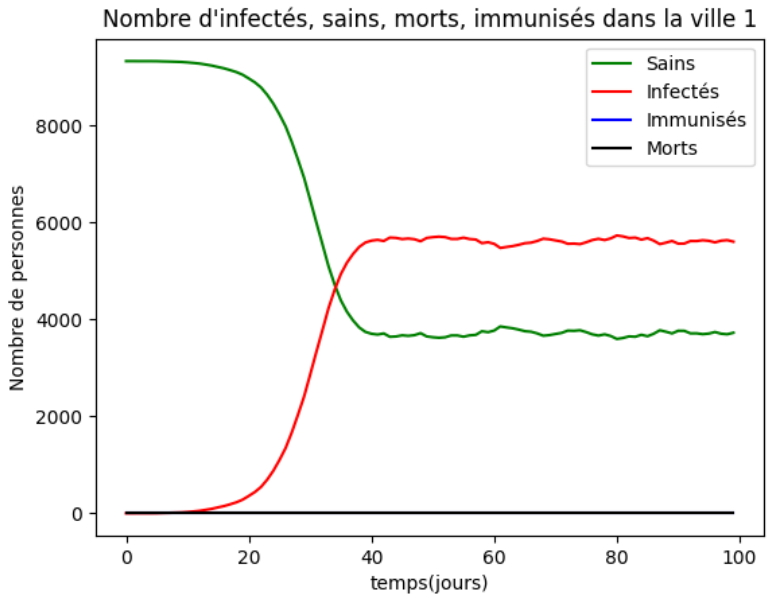
$$u_t = \frac{5}{32} \times x$$

Au final la part de la population qui devient saine à la date t est $0.9 \times u_t = \frac{9}{64} \times x$

À l'équilibre, le nombre de personnes devenant saines est le même que le nombre de personnes devenant infectées :

$$\frac{9}{64}x = 0.36x - 0.36x^2$$

$x = 0.61$ est la solution qui nous intéresse, c'est une approximation relativement correcte, car le taux réel d'infectés à l'équilibre sur ma simulation est 0.57.



3 Choix des chiffres

Le but est de simuler au mieux la Covid-19. Les chiffres qui vont être présentés sont pour la plupart issus du site *ourworldindata.org*. Cette source nous a été conseillée par le chercheur dans ce domaine David Hill, que nous avons pu rencontrer dans le cadre de notre TIPE.

Pour la durée de la maladie, trop de résultats différents sont trouvables selon les dates de parution... au final nous choisissons 7 jours. De plus on propose que 90% des personnes infectées depuis 7 jours redeviennent saines. Les autres peuvent mourir, ou rester infectées (pour la plupart) et avoir à nouveau 90% de chance de redevenir saines le lendemain.

Taux de contamination : le Ro est le nombre de personnes qu'un malade infecte en moyenne autour de lui. Une étude chinoise trouve un Ro entre 2 et 2.5 pour la Covid19, mais une autre étude américaine trouve un Ro de 5.7 environ, on choisit donc de faire la moyenne et de retenir un Ro de 3.975. Or

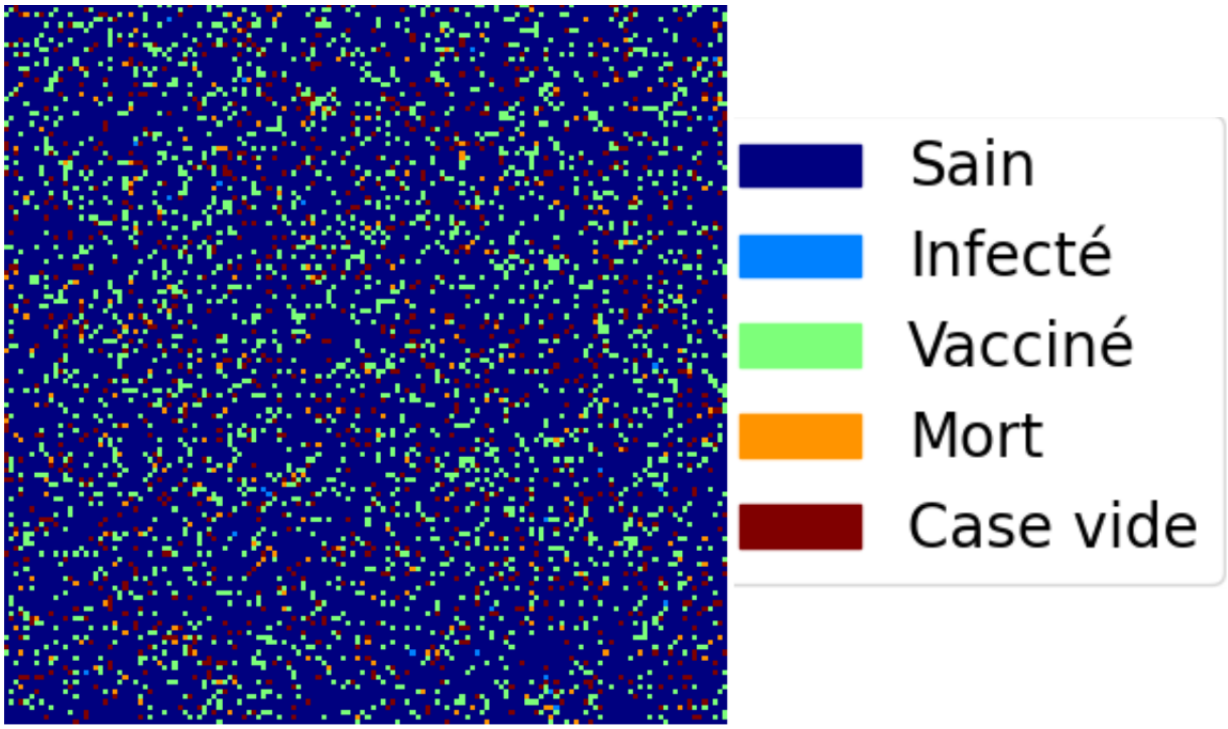
nous avons proposé que la maladie dure généralement 7 jours, donc chaque jour une personne infectée contamine environ $\frac{3.975}{7} = 0.568$. Un infecté a 8 voisins en général (sauf effet de bord), d'où la probabilité qu'il contamine un voisin donné est : $\frac{0.568}{8} = 0.07$. Ainsi une personne à 0.07 chance par voisin contaminé de devenir une infectée, c'est notre taux de contamination.

Taux de mortalité : en regardant les chiffres de la France aujourd'hui (26/02/2022), 138 027 personnes sont décédées de la Covid19, et 22.65 M ont été touchées. On fait le rapport pour obtenir 0.0061, le taux que nous choisissons pour la mortalité.

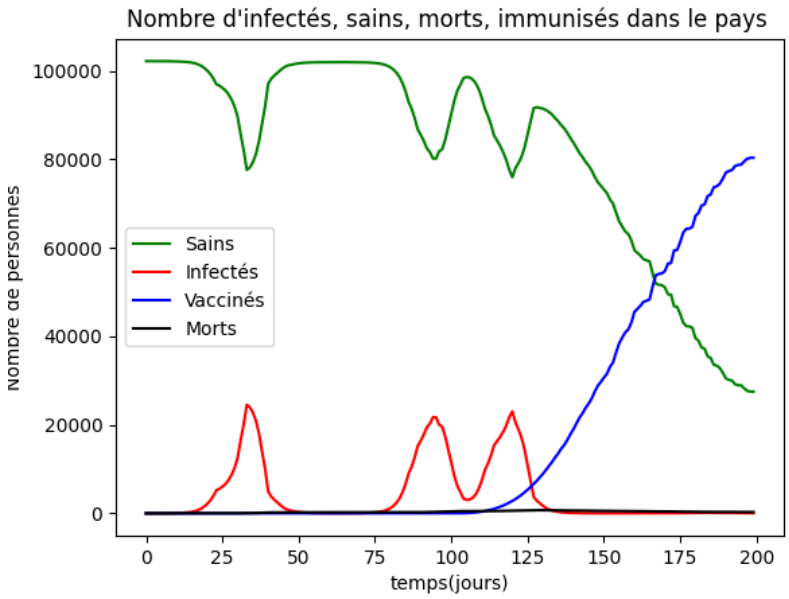
Pour ce qui est du vaccin, on choisit de dire qu'il finit d'être développé au bout de 100 jours au lieu d'environ un an pour la covid19 car cela n'a pas d'intérêt pour la simulation d'attendre. On propose, pour rester à notre petite échelle de dire qu'il est efficace et protège totalement les gens d'attraper la maladie, mais seulement pendant 50 jours.

4 Résultats

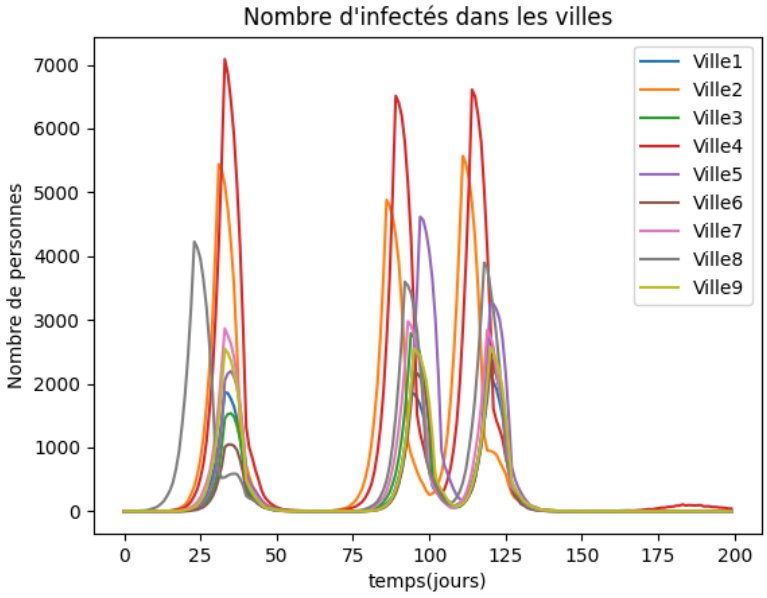
Voici une ville à un stade très avancé de la simulation :



Une simulation testant une vaccination quadratique de la population :

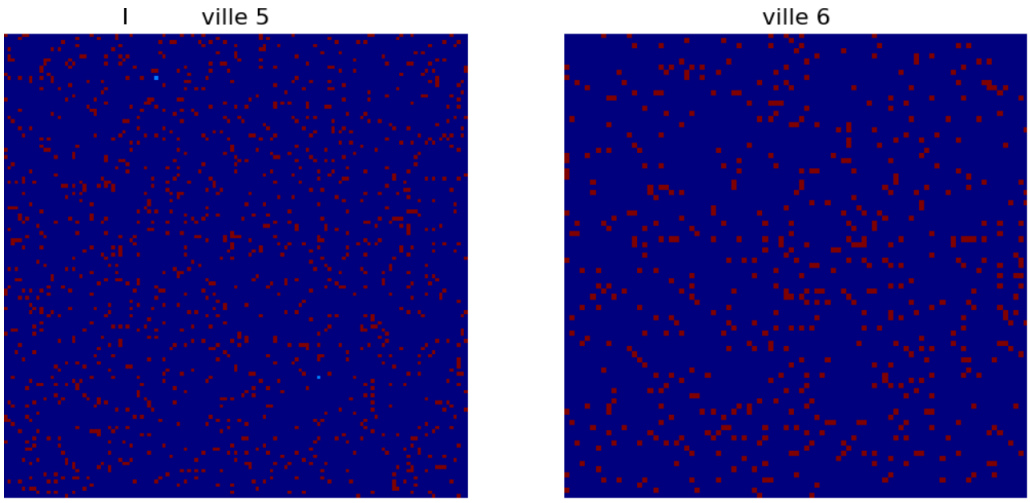


L'évolution du nombre d'infectés pour cette même simulation :



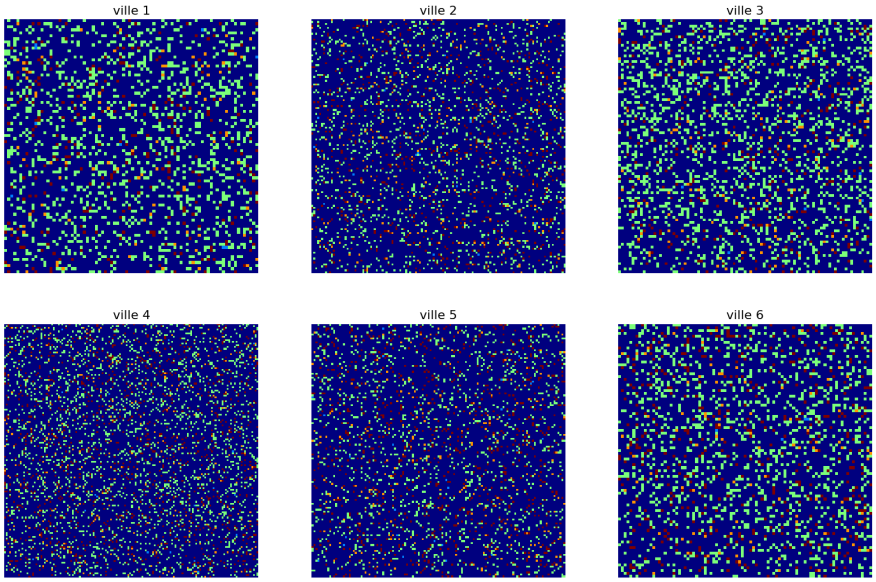
4.1 Vaccination linéaire

Présentons en détail les résultats obtenus avec une vaccination linéaire de la population, voici les villes 5 et 6 au début de la simulation (il y a 9 villes) :



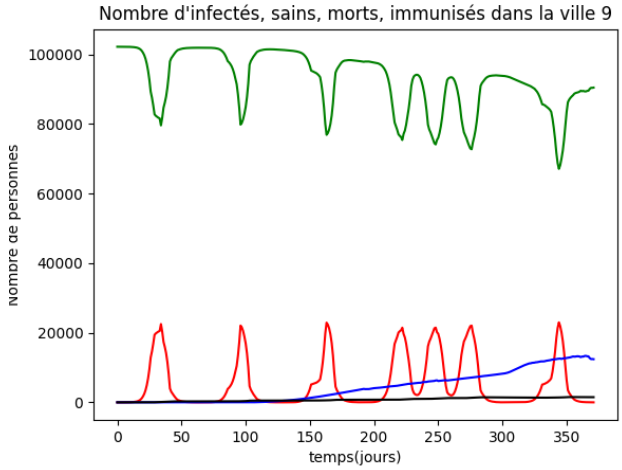
On peut visualiser que les premiers infectés sont dans la ville 5, toutes les autres villes ressemblent à la ville 6, elles ne contiennent que des personnes saines et une proportion de 0.07 de cases vides.

La vaccination a été choisie de telle façon qu'à la date t, supérieure à la date du début de la vaccination (100 jours), $(t - 100)$ personnes se fassent vacciner. Voici la situation à la fin de la simulation :

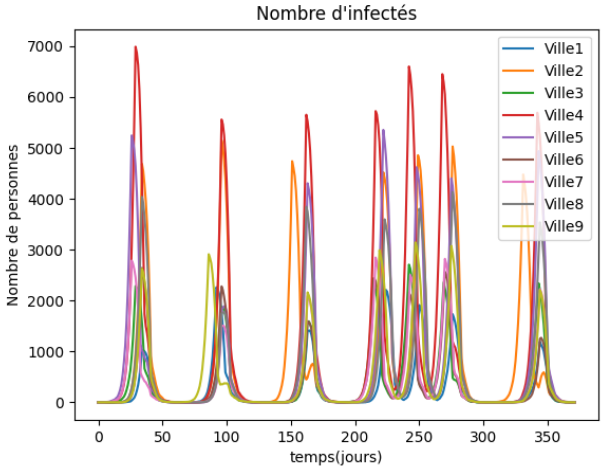


Cette simulation a été très longue à produire (360 jours, avec une production d'image tout les 20 jours). Le problème de ce mode de vaccination est qu'au bout de 50 jours de vaccination, on atteint un équilibre et on ne vaccine plus que 50 nouvelles personnes par jour, le processus est très long... Il faudrait plus de 2 000 jours pour vacciner toute la population. Donc la dynamique de phase n'est pas vraiment enrayée par ces mesures (voir dernier graphique).

Courbe d'évolution des différents états (sain en vert, infecté en rouge, mort en noir et vacciné en bleu) et ceci dans la ville 9 :



Visualisation des vagues avec les nombres de malades dans les différentes villes (la ville 4 est la plus peuplée, on peut le deviner sur le graphique) :

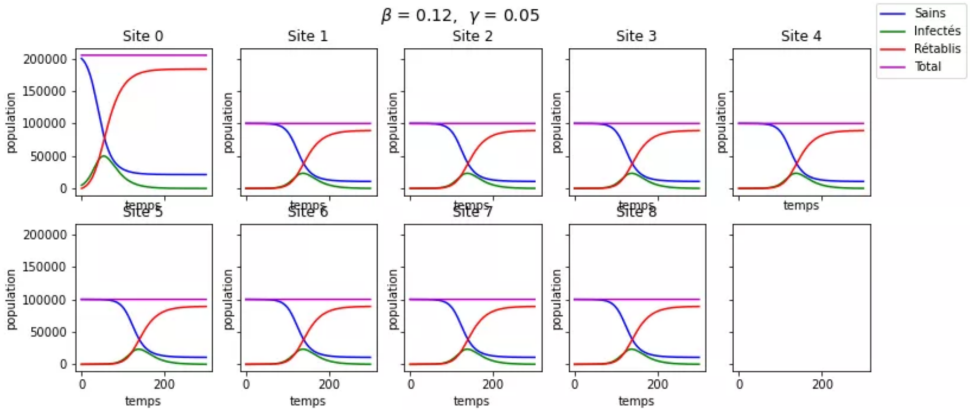


5 Annexes

5.1 Résultats de mon camarade

Mon binôme pour ce TIPE, Jules Barra n'a pu me fournir que très tardivement ses résultats nous n'avons donc pas fait de comparaison des modèles.

Voici un de ses principaux résultats obtenu avec la méthode des équations différentielles :



5.2 Listing du code utilisé

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 16 20:38:21 2021
@author: Arthur
"""
from random import random, randint
from math import *
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches


# Impl mentation d'un tri fusion
def fusion(T1, T2):
    T = []
    while len(T1) != 0 and len(T2) != 0:
        if T1[0] <= T2[0]:
            T.append(T1[0])
            T1 = T1 [1:]
        else:
            T.append(T2[0])
            T2 = T2 [1:]
    return T + T1 + T2

def tri_fusion(T):
    if len(T) <= 1:
        return T
```

```
27      else:
28          return fusion(tri_fusion(T[ : round(len(T) / 2)]),
        tri_fusion(T[round(len(T) / 2) : ]))
29
30
31  def bernoulli(p):
32      return random() < p
33
34  # Creer une ville de taille p (p**2 habitants),
35  # sous la forme d'une matrice.
36  # On rajoute    la fin un compteur qui s'actualisera
37  # des Sain, Contamin , R tablit , Mort et case vide presents dans
        la ville.
38  def creer_ville (p):
39      nb_vide = round(p * p * 0.07) #le nombre de case qui seront
        vides
40      C=[p * p - nb_vide,0,0,0,nb_vide]
41      V = [[[0,0] for i in range(p)] for i in range(p)]
42      entiers = [i for i in range(p * p)]
43
44      for k in range(nb_vide):
45          q = randint(0,len(entiers) - 1 - k)
46          i = entiers[q]
47          V[i//p][i%p] = [4,0]
48          entiers = entiers[:q]+entiers[q+1:] #ainsi on ne peut tirer
         deux fois la m me case
49
50      V.append(C)
51      return V
52
53
54  #n villes, la ville i est de taille lp[i] (lp[i]**2 habitants).
55  def creer_pays(n,lp):
56      P=[]
57      for i in range(n):
58          V = creer_ville(lp[i])
59          P.append(V)
60      return P
61
62
63  def melange_liste(l):
64      E = []
65      r = len(l)
66      for i in  range(r):
67          k = randint(0,r - 1 - i)
68          E.append(l[k])
69          l = l[:k] + l[k+1:]
70      return E
71
72
73  # trajets en bus dans une ville, V : ville,
74  # p : portion de gens qui se deplacent quotidiennement dans cette
        ville.
75  def trajet_bus(V,p):
76      k = len(V) - 1
77      nb = round((1 - p) * k * k)  #nb de gens qui ne se deplacent
        pas
```

```
78      bus = []
79      entiers = [i for i in range(k * k)]
80      E = []
81      for i in range(nb):
82          q = randint(0,k * k - 1 - i)
83          E.append(entiers[q])
84          entiers = entiers[:q] + entiers[q+1:]
85      E = tri_fusion(E)        #E contient les positions des gens qui
         ne se d placent pas
86      if E == []:
87          for i in range(k * k):
88              if V[i//k][i%k][0] != 4 and V[i//k][i%k][0] != 3:
89                  bus.append(V[i//k][i%k])     #mont e dans le bus
90                  V[i//k][i%k] = [4,0]          #la case devient donc
        vide
91      else:
92          #tout le monde monte sauf les personnes tir e au sort
        avant (dans E) et les morts
93          for i in range(k * k):
94              if i == E[0]:
95                  if len(E) > 1:
96                      E = E[1:]
97              else:
98                  if V[i//k][i%k][0] != 4 and V[i//k][i%k][0] != 3:
99                      bus.append(V[i//k][i%k])
100                     V[i//k][i%k] = [4,0]
101     bus = melange_liste(bus)
102     Case_vide = []
103     for i in range(k * k):
104         if V[i//k][i%k][0] == 4:
105             Case_vide.append(i)      #Case_vide contient les
        positions des cases vides
106     nb_vides = len(Case_vide)
107     long_bus = len(bus)
108     for j in range(long_bus):
109         n = randint(0, nb_vides - 1 - j)
110         q = randint(0,long_bus - 1 - j)
111         i = Case_vide[n]
112         V[i//k][i%k] = bus[q]        #descente du bus dans une case
         vide
113         bus = bus[:q] + bus[q+1:]
114         Case_vide = Case_vide[:n] + Case_vide[n+1:]
115     return V
116
117
118 # l une liste d'  lment  , la fonction renvoie k
119 # si il y a k fois la valeur 1 dans l
120 def aux1(l):
121     c = 0
122     for i in l:
123         if i == 1:
124             c += 1
125     return c
126
127
128 # la fonction determine le nombre de voisins
129 # de l'element en place n de la ville V sont infect s.
```

```
130 def est_vulnerable(n,V):
131     k = len(V) - 1
132     if V[n//k][n%k][0] == 0: #on ne s'int resse qu'aux personnes
        encore saines
133         if n == 0:              #coin sup rieur gauche
134             return aux1([V[0][1][0],V[1][0][0],V[1][1][0]])
135         elif n == k - 1:        #coin sup rieur droit
136             return aux1([V[0][k-2][0],V[1][k-2][0],V[1][k-1][0]])
137         elif n == k * (k - 1): #coin inf rieur gauche
138             return aux1([V[k-2][0][0],V[k-2][1][0],V[k-1][1][0]])
139         elif n == k * k - 1:   #coin inf rieur droit
140             return aux1([V[k-2][k-2][0],V[k-2][k-1][0],V[k-1][k
        -2][0]])
141         elif n//k == 0:         #arrete du haut
142             return aux1([V[0][n-1][0],V[0][n+1][0],V[1][n-1][0],V
        [1][n][0],V[1][n+1][0]])
143         elif n//k == k - 1:     #arrete du bas
144             return aux1([V[k-1][n%k-1][0],V[k-1][n%k+1][0],V[k-2][n
        %k-1][0],V[k-2][n%k][0],
145             V[k-2][n%k+1][0]])
146         elif n%k == 0:          #arrete de gauche
147             return aux1([V[n//k-1][0][0],V[n//k+1][0][0],V[n//k
        -1][1][0],V[n//k][1][0],
148             V[n//k+1][1][0]])
149         elif n%k == k-1:        #arrete de droite
150             return aux1([V[n//k-1][k-1][0],V[n//k+1][k-1][0],V[n//k
        -1][k-2][0],V[n//k][k-2][0],
151             V[n//k+1][k-2][0]])
152         else:                   #centre
153             return aux1([V[n//k-1][n%k-1][0],V[n//k][n%k-1][0],V[n
        //k+1][n%k-1][0],V[n//k-1][n%k][0],
154             V[n//k+1][n%k][0],V[n//k-1][n%k+1][0],V[n//k][n%k
        +1][0],V[n//k+1][n%k+1][0]])
155     else:
156         return 0
157
158
159 #modifie la ville en contaminant les personnes vuln rables avec
        une probabilit  p.
160 def contamination(V,p):
161     k = len(V) - 1
162     E = []
163     for i in range (k * k):
164         b = False
165         #si une personne a n voisins contamin s , elle a n fois
        plus de chance de le devenir
166         for j in range(est_vulnerable(i,V)):
167             b = (b or bernoulli(p))
168         if b:
169             E.append(i)
170     for i in E:
171         V[i//k][i%k] = [1,0]
172         V[k][1] += 1
173         V[k][0] -= 1
174
175
176 # Devenir d'une personne infect e; d = dur e de la maladie; p3=
```

```
    probabilit  de mourrir;
# p5=probabilit  de redevenir saint, il reste la probabilit  que l'infect  reste infect
def apres_inf(V,d,p3,p5):
    k = len(V) - 1
    for i in range(k * k):
        # partie infect  :
        if V[i//k][i%k][0] == 1:
#On passe chaque jour dans apres_inf, on en profite pour ajouter 1 au compteur de dur e de maladie
            V[i//k][i%k][1] += 1
            if V[i//k][i%k][1] >= d:
                #pass  la dur e de la maldie, la malade peut voluer
                if bernoulli(p3):
                    V[i//k][i%k] = [3,0]
                    V[k][1] -= 1
                    V[k][3] += 1
                elif bernoulli(p5/(1-p3)):
                    V[i//k][i%k] = [0,0]
                    V[k][1] -= 1
                    V[k][0] += 1
                # sinon, il reste infect
        # partie vaccin  :
        if V[i//k][i%k][0] == 2:
#On passe chaque jour dans apres_inf, on en profite pour ajouter 1 au compteur de dur e de vaccin
            V[i//k][i%k][1] += 1
            if V[i//k][i%k][1] >= 50:  #le vaccin est suppos  efficace 50 jours
                V[i//k][i%k] = [0,0]   # la personne redevient saine
                V[k][2] -= 1
                V[k][0] += 1


# p la portion de gens qui se d placent en train ( l ' chelle nationale) ;
# P un pays ; pop sont nombre d'habitants.
# La fonction renvoie la matrice de d placement associ e    cette  situation.
def creer_mat_dep(p,P,pop):
    n = len(P)
    m = [[0 for i in range(n)] for i in range(n)]
    for i in range(n):
        h = (len(P[i]) - 1) ** 2 # nb d'habitants de la ville i
        for j in range(n):
            if i != j:
                m[i][j] = round((len(P[j]) - 1) ** 2 * h * p / pop)
    return m


#P un pays; mat_dep la matrice de d placement
def trajet_train(P,mat_dep):
    n = len(P)
    gare = []   # va accueuillir les trains, puis les renvoyer dans  les bonnes villes
```

```
224     for i in range(n):
225         train = []      # va accueuillir tout les voyageur allant
        la ville i
226         for j in range(n):
227             wagon = []    # le wagon allant de la ville j    la
    ville i
228             if i != j:
229                 p = len(P[j]) -1 #population de la ville j
230                 nb = mat_dep[j][i]  #nb de gens qui se deplacent
231                 while nb != 0:
232                     q=randint(0,p*p-1)
233                     etat = P[j][q//p][q%p][0]
234                     if etat != 3 or etat != 4:
235                         wagon.append(P[j][q//p][q%p])
236                         P[j][q//p][q%p] = [4,0]
237                         nb -= 1
238                         P[j][p][etat] -= 1
239                         #on ajoute pas de case vide car on va la
    remplir juste apr s
240             wagon = melange_liste(wagon)
241             train += wagon
242         gare.append(train)    # ainsi on maitrise les destinations,
        le train i va    la ville i.
243
244     for i in range(n): #redistribution
245         train = gare[i] #on reparti les personnes du train num ro
    i (ceux allant    i) dans la ville i
246         p = len(P[i]) - 1
247         case_vide = []
248         for j in range(p * p):
249             if P[i][j//p][j%p][0] == 4:
250             #on regarde les indices des cases vides pour distribuer
        dans celle-ci
251                 case_vide.append(j)
252         long = len(case_vide)
253         for k in range(len(train)):
254             r = randint(0,long - 1 - k)
255             c = case_vide[r]
256             P[i][c//p][c%p] = train[k]
257             etat = train[k][0]
258             P[i][p][etat] += 1
259             case_vide = case_vide[:r] + case_vide[r+1:]
260     return P
261
262
263 #P un pays de n ville, on va vacciner nb_vax personnes dans une des
        villes
264 def vaccination(nb_vax,P,n):
265     q = randint(0, n-1)
266     tailleq = len(P[q]) - 1
267     vaccinables = []
268     for k in range(tailleq * tailleq):
269         etat = P[q][k//tailleq][k%tailleq][0]
270         if etat == 0 or etat == 1:
271             vaccinables.append(k)
272     #on ne peut pas vacciner plus de gens que le nombre de gens
    vaccinables (sains ou infect s)
```

```
    for i in range(min(nb_vax,len(vaccinables))):
        r = randint(0,len(vaccinables) -1)
        etat = P[q][r//tailleq][r%tailleq][0]
        P[q][r//tailleq][r%tailleq] = [2,1]
        P[q][tailleq][etat] -= 1
        P[q][tailleq][2] += 1


#fonction utile    l'affichage.
def traduit(P, n):
    P0 = []
    for k in range(n):
        p = len(P[k]) - 1
        V0 = []
        for i in range(p):
            l0 = []
            for j in range(p):
                l0.append(P[k][i][j][0])
#P0 est le pays P o   les habitants ne sont plus un couple d'
    informations,
# mais juste une seule : leur  tat .
            V0.append(l0)
        P0.append(V0)
    return P0


#P0 le pays    mod liser (sous forme traduite), n un carr  d'
    entier : le nombre de villes.
def images(P0, n, noms, etats, c1, c2, j):
    fig, axs = plt.subplots(int(sqrt(n)), int(sqrt(n)),figsize
    =(20,20))
    for i,ax in enumerate(fig.axes):
        p = len(P0[i])
        g = ax.imshow(P0[i], cmap = 'jet')   #affichage de la ville
     i
        ax.set_title(noms[i],fontsize = 16)
        ax.axis('off')
    # L gende pour la figure
    couleur = [0,1,2,3,4]
    colors = [ g.cmap(g.norm(value)) for value in couleur]
    patches = [ mpatches.Patch(color = colors[j], label = etats[j]
    ) for j in range(len(couleur)) ]
    #on informe d'un confinement en cours dans le titre de l'image
    :
    if c2 != 0:
        plt.suptitle("Carte des villes pendant un confinement du
    pays    la date " + str(j),
        fontsize=26,x=0.5,y=0.1)
    elif c1 != [0 for i in range(n)] :
        E = ""
        fst = 0
        for i in range(n): #on selectionne les villes confin es
            if c1[i] != 0:
                if fst == 0:
                    E = str(i+1)
                    fst = 1
                else:
```

```
323                     E = E + ", " + str(i + 1)
324         if len(E) == 1:
325             plt.suptitle("Carte des villes pendant un confinement
    de la ville "+E,fontsize=26,
326             x=0.5,y=0.1)
327         else:
328             plt.suptitle("Carte des villes pendant un confinement
    des villes "+E,fontsize=26,
329             x=0.5,y=0.1)
330     else:
331         plt.suptitle("Cartes des villes    la date "+str(j),
    fontsize=26,x=0.5,y=0.1)
332         #on informe la date
333     plt.legend(handles = patches, bbox_to_anchor = (0.90, 0), loc =
     2,
334     borderaxespad = 0.,fontsize = 20 )
335     plt.savefig('.\carte_simul_v16_date_{}.png'.format(j),dpi =
    100)
336     plt.close()
337
338
339 # matrice est un matrice carr e , scal est un scalaire.
340 # la fonction renvoie 1/scal * matrice
341 def div(matrice,scal):
342     n = len(matrice)
343     m = [[0 for i in range(n)] for i in range(n)]
344     for i in range(n):
345         for j in range(n):
346             m[i][j] = matrice[i][j] * scal
347     return m
348
349
350 def simulation_maladie_dans_pays(n,lp,d,t,p1,p2,p3,fct_vax,p5,p6,p7
    ,p8,noms,f):
351     """n villes, leurs tailles sont dans lp; d=dur e de la
    maladie (jours);
352     l'exp rience a une dur e de t jours; p1=proba d' tre
    contamin  en  tant  vulnerable;
353     p2=part de la pop des villes qui se deplace en bus; p3=proba de
     mourrir pour un infect ;
354     fct_vax : rythme de vaccination; p5=proba de redevenir saint
    apr s une infection;
355     p6=part de la pop du pays qui se d place en train;
356     p7=part de la population du pays contamin e    partir de
    laquelle confinement global;
357     p8=part de la population d'une ville contamin e    partir de
    laquelle confinement de cette ville"""
358     P = creer_pays(n,lp)
359     pop = 0
360     for i in range(len(lp)):
361         pop += lp[i] ** 2     #pop sera le nombre d'habitant du
    pays
362     mat_dep = creer_mat_dep(p6,P,pop)
363     etats = ["Sain","Infect ","Vaccin ","Mort","Case vide"]
364     #dans l'ordre (   la place i il y a l' tat  n i)
365     #contamination d'une personne :
366     q = randint(0, n - 1)
```

```
    tailleq = len(P[q]) - 1
    while P[q][tailleq][1] != 1: #on attend qu'il y aie effectivement un infect
        k = randint(0,tailleq * tailleq -1)
        if P[q][k//tailleq][k%tailleq][0] == 0:
            P[q][k//tailleq][k%tailleq][0] = 1
            P[q][tailleq][1] += 1
            P[q][tailleq][0] -= 1
    Nb_0 = [[] for i in range(n)]  #Nb_0[i][t] contient le nombre de sains de la ville i    la date t
    Nb_1 = [[] for i in range(n)]  #Nb_0[i][t] contient le nombre de infect s de la ville i    la date t
    Nb_2 = [[] for i in range(n)]  #Nb_0[i][t] contient le nombre de vaccin s de la ville i    la date t
    Nb_3 = [[] for i in range(n)]  #Nb_0[i][t] contient le nombre de morts de la ville i    la date t
    c1 = [0 for i in range(n)]
    #si c1[i] = 0, la ville i n'est pas confin e , sinon c1[i]=t, la ville i est confin e depuis t jours
    c2 = 0
    #si c2 = 0, le pays n'est pas confin e , sinon c2=t, le pays est confin e depuis t jours
    temps = [i for i in range(t)]
    for j in range(t):
        if j >= 100:  #mise en place de la vaccination
            vaccination(round(fct_vax(j - 100)),P,n)
            #la fonction fct_vax nous donne le nombre de personne qui seront vaccin s    la date j
        E = []
        nbinftot = 0
        for i in range(n):
            taillei = len(P[i]) - 1
            Nb_0[i].append(P[i][taillei][0])
            Nb_1[i].append(P[i][taillei][1])
            Nb_2[i].append(P[i][taillei][2])
            Nb_3[i].append(P[i][taillei][3])
            if P[i][taillei][1] != 0:
                #on fait  voluer  la ville seulement si elle contient des infect s
                #les infect s qui le sont depuis plus de 7j  volue , deviennent mort, sain
                # ou restent infect s. Les vaccin s qui le sont depuis plus de 50j redeviennent sain
                apres_inf(P[i],d,p3,p5)
                if c2 != 0: #crit re de confinement nationnal
                    contamination(P[i],p1/10)
                    #on reduit de 10 les trajets dans les villes, et les contaminations
                    trajet_bus(P[i],p2/10)
                else:
                    if c1[i] == 0: #la ville i n'est pas confin e
                        if p8>(P[i][taillei][1]/(taillei*taillei)):
                            contamination(P[i],p1)
                            trajet_bus(P[i],p2)
                        else: #on a d pass  le seuil, le confinement de cette ville commence
                            c1[i] += 1
```

```
411                             contamination(P[i],p1/10)
412                             trajet_bus(P[i],p2/10)
413                     elif c1[i]==14: #on est arriv     la fin du
    confinement de la ville i
414                         c1[i]=0
415                         contamination(P[i],p1)
416                         trajet_bus(P[i],p2)
417                     else:   #on est en court de confinement de la
    ville i
418                         c1[i]+=1
419                         contamination(P[i],p1/10)
420                         trajet_bus(P[i],p2/10)
421                 nbinftot+=P[i][taillei][1]
422                 #nbinftot contiendra le nombre d'infect s du pays
       la date j    la fin de cette boucle
423         if c2==0 : #il n'y a pas de confinement global en court
424             if p7>(nbinftot/(pop)) :
425                 trajet_train(P, mat_dep)
426                 print(1)
427             else: #on a d pass  le seuil, on d clanche le
    confinement global
428                 c2+=1
429                 trajet_train(P, div(mat_dep,5))  #on reduit de 5
    les d placements entre villes
430         elif c2==30: #fin du confinement global
431             c2=0
432             c1 = [0 for i in range(n)]
433             trajet_train(P, mat_dep)
434         else: #on est en court de confinement global
435             c2+=1
436             trajet_train(P, div(mat_dep,5))
437
438         if nbinftot==0: #la maladie s'est  teinte
439             print(j)  #le programme nous informe de la date de l'
    h radication de la maladie
440             temps = temps[:j+1]
441             break
442         #on produit une image tout les k*f jours, k un entier
443         #1000 est le code indiquant qu'on ne veut pas produire d'
    image
444         #(fait gagner beaucoup en temps de calcul)
445         if j % f == 0 and f != 1000:
446             P0 = traduit(P, n)
447             images(P0, n, noms, etats, c1, c2, j)
448
449     plt.figure() #on produit le graphe du nombre d'infect s dans
    chaque ville
450     plt.title("Nombre d'infect s dans les villes")
451     plt.xlabel('temps(jours)')
452     plt.ylabel("Nombre de personnes")
453     for i in range(n):
454         plt.plot(temps,Nb_1[i], label = 'Ville'+str(i+1))
455     plt.legend()
456     plt.savefig('.\graphe_simul_v20_{}.png'.format(1),dpi=100)
457     plt.close()
458
459     #on calcul la liste des nombres de sains, infect s, vaccin s,
```

```
460     #morts aux dates j    l' chelle  du pays, en sommant les listes
         que l'on avait pour chaque villes
461     Nb_0_tot=[]
462     Nb_1_tot=[]
463     Nb_2_tot=[]
464     Nb_3_tot=[]
465     for j in temps:
466         tot0 = 0
467         tot1 = 0
468         tot2 = 0
469         tot3 = 0
470         for i in range(n):
471             tot0 += Nb_0[i][j]
472             tot1 += Nb_1[i][j]
473             tot2 += Nb_2[i][j]
474             tot3 += Nb_3[i][j]
475         Nb_0_tot.append(tot0)
476         Nb_1_tot.append(tot1)
477         Nb_2_tot.append(tot2)
478         Nb_3_tot.append(tot3)
479     plt.figure() #on produit le graghe des nombres de sain,
        infect , vaccin , mort    l' chelle  du pays
480     plt.title("Nombre d'infect s , sains, morts, immunis s dans le
         pays ")
481     plt.xlabel('temps(jours)')
482     plt.ylabel("Nombre de personnes")
483     plt.plot(temps,Nb_0_tot, color='green', label = 'Sains')
484     plt.plot(temps,Nb_1_tot, color='red', label = 'Infect s')
485     plt.plot(temps,Nb_2_tot, color='blue', label = 'Vaccin s')
486     plt.plot(temps,Nb_3_tot, color='black', label = 'Morts')
487     plt.legend()
488     plt.savefig('.\graphe_simul_dans_pays_v20_{}.png'.format(1),dpi
        =100)
489     plt.close()
490
491
492 n=9         #n villes, n doit  tre  un carr  d'entier naturel
493 lp=[84,130,93,146,129,89,95,113,98] #les tailles des villes, lp[i]
        est la taille de la ville i
494 d=7         #d=dur e de la maladie en jours
495
496 t=3      #l'exp rience a une dur e de t jours
497 p1=0.07     #p1=proba d'etre contamin  en etant vulnerable
498 p2=0.9      #p2=part de la pop qui se deplace en bus tout les jours
499
500 p3=0.0061  #p3=proba de mourrir pour un infect
501
502 def fct_vax(t):
503     f = t ** 4 * cos(t)
504     return (abs(f) + f) /2
505 #rythme de vaccination une fois que la vaccination est d clanch e
         (100 jours)
506
507 p5=0.9      #p5=proba de redevenir saint
508
509 p6=0.01     #p6 = part de la pop d'une ville qui se d place en
        train
```

```
p7=0.20     #p7 = taux de contamination du pays    partir duquel :
            #  moins de d placement entre ville,
            #  et confinement de toute les villes pendant 30 jours
p8=0.25     #p8 = taux de contamination du pays    partir duquel :
            #  moins de d placement dans cette ville,
            #  moins de propagation pendant 14 jour

noms=["ville "+str(i+1) for i in range(n)]
f = 100

simulation_maladie_dans_pays(n,lp,d,t,p1,p2,p3,fct_vax,p5,p6,p7,p8,
    noms,f)
```

5.3 Sources

Pour la plupart des chiffres :

https://ourworldindata.org/coronavirus

Pour le r_0 du covid-19 :

https://www.who.int/docs/default-source/coronaviruse/who-china-joint-mission-on-covid-19-final-report.pdf

https://wwwnc.cdc.gov/eid/article/26/7/20-0282_article