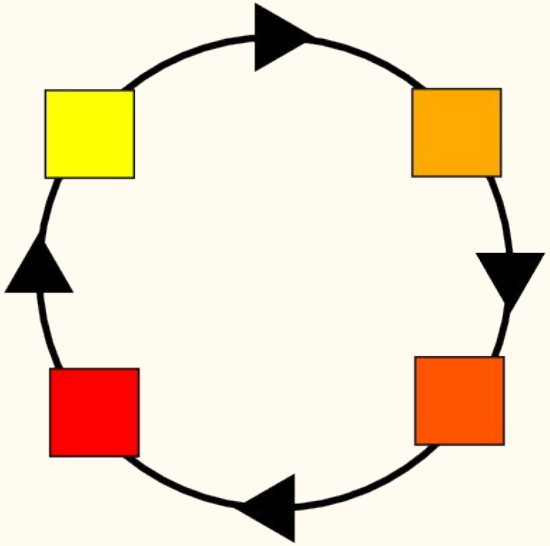
Simulateur de maladie par automate cellulaire

Par Arthur BARJOT, N° 49859

Est-il possible de créer un modèle simplifié d'épidémie, qui représente cependant efficacement la réalité ?

I. Qu’est ce qu’un automate cellulaire?

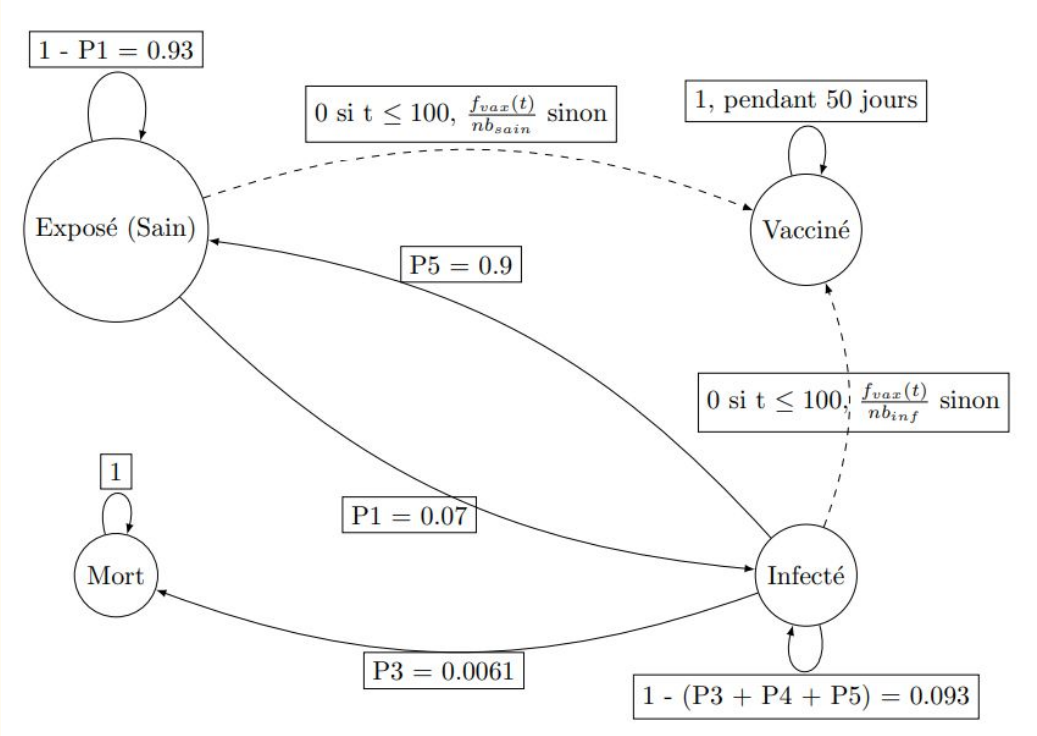
0	3	0	4	0	1
0	1	0	0	2	0
0	4	0	1	0	3
0	1	0	0	4	0
0	2	1	0	3	0
0	4	0	0	1	0



Source image :
https://fr.wikipedia.org/wiki/Automate_cellulaire

II. Le modèle considéré:

Transitions entre les différents états



III. Analyse du programme Python :

1-Création du pays

0	3	0	4	0	1
0	1	0	0	2	0
0	4	0	1	0	3
0	1	0	0	4	0
0	2	1	0	0	0
0	0	4	0	3	0

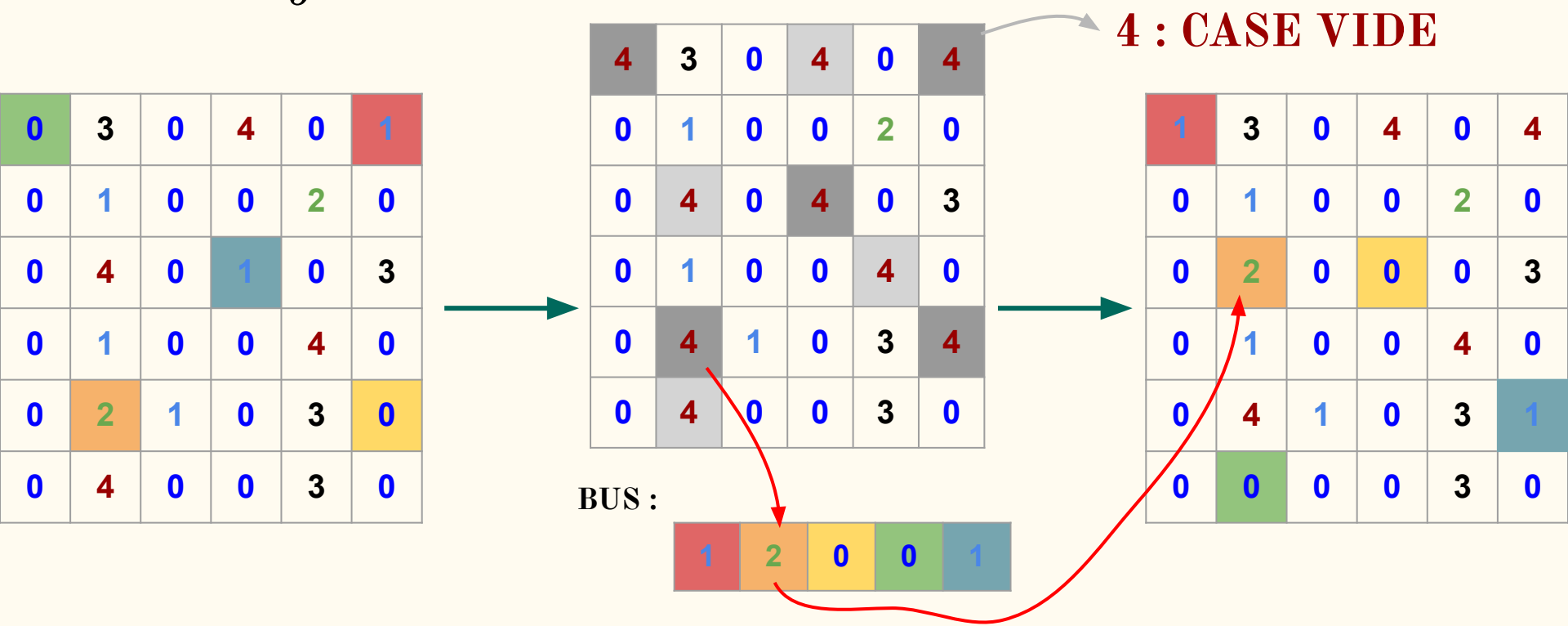
n

n

0 : Sain
1 : Infecté
2 : Vacciné
3 : Mort
4 : Case Vide

```
def creer_ville (n):
    V = []
    C = [0,0,0,0,0]
```

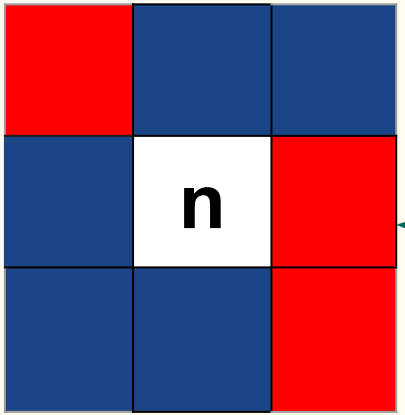
III. Analyse du programme Python :
2-Trajets dans une ville



III. Analyse du programme Python :
3-Propagation de la maladie

```
est_vulnerable(n,V):
```

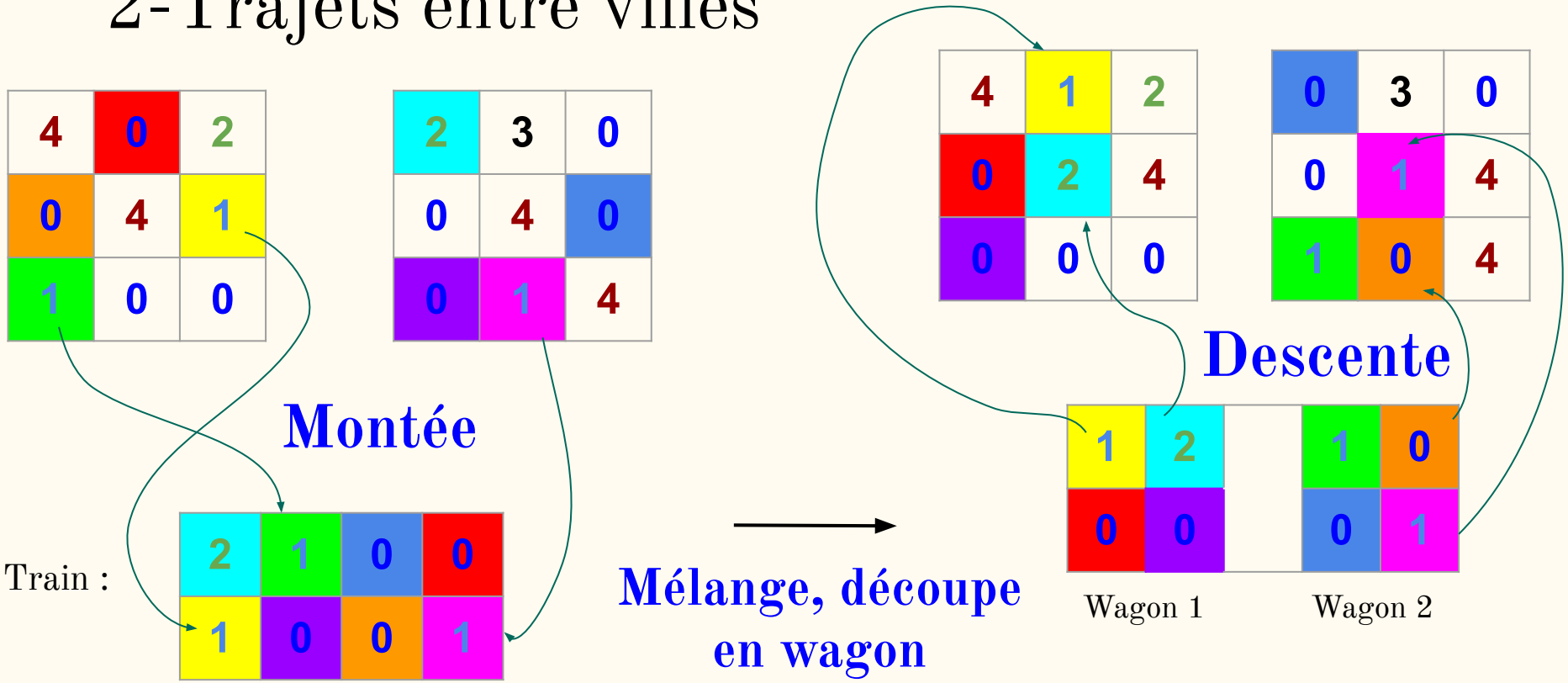
Susceptibilité de la personne n à être contaminée par ses voisins




```
apres_inf(V,d,p3,p4,p5):
```

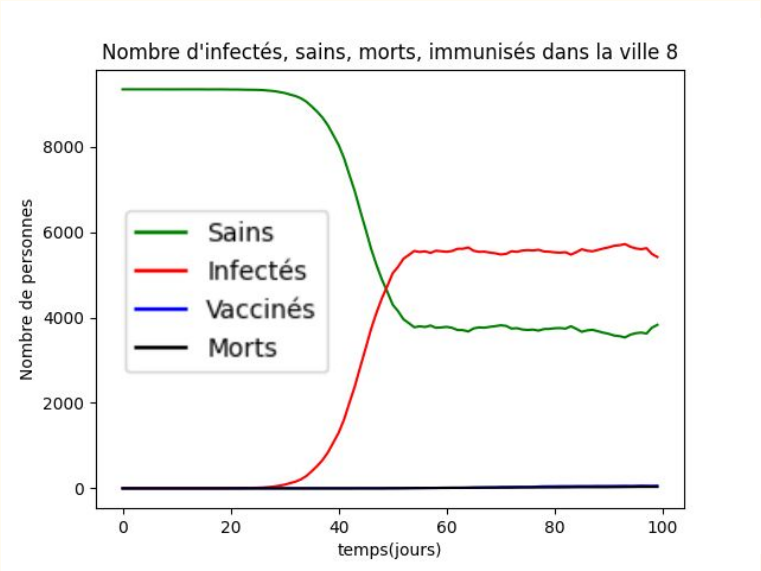
III. Analyse du programme Python :

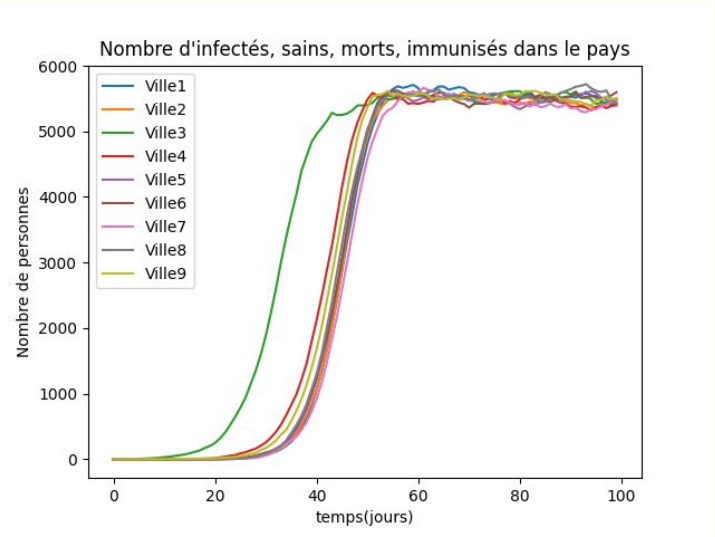
2-Trajets entre villes



III. Analyse du programme Python :
3-Concrétisation

```
def simulation_maladie_dans_pays(n,p,d,t,p1,p2,p3,p4,p5,p6,p7,p8,noms,f)
```

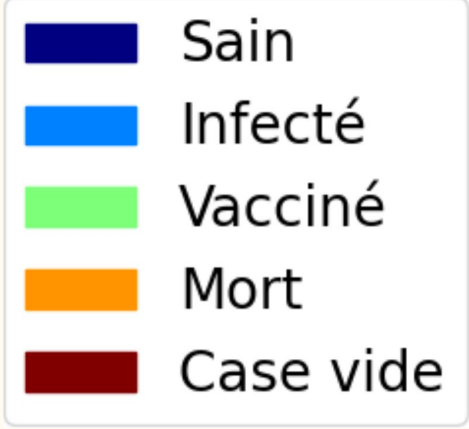




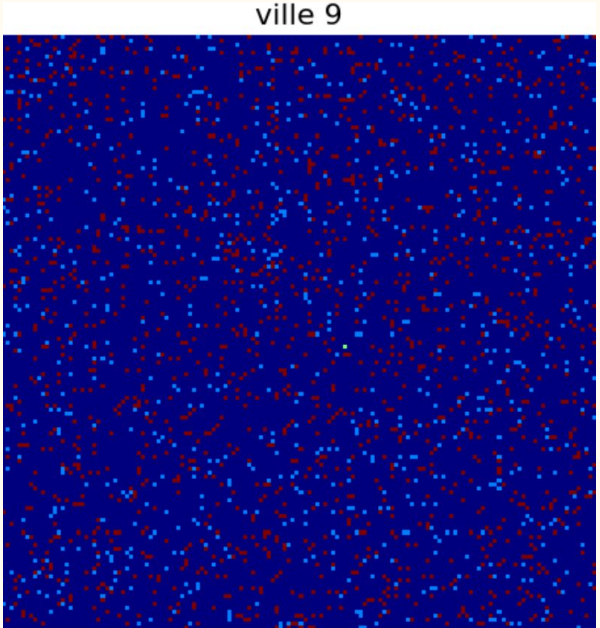
9 villes de 10 000 habitants

Démonstration en annexe de la position d'équilibre du système

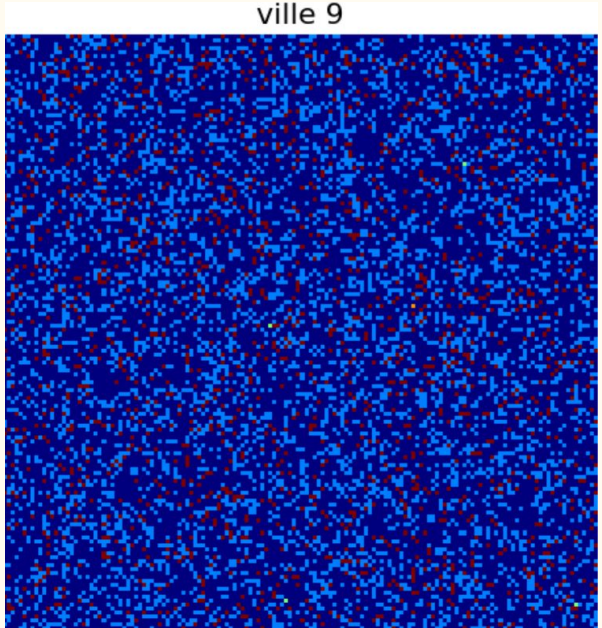
IV. Améliorations du système :
1- Mise en place de confinements



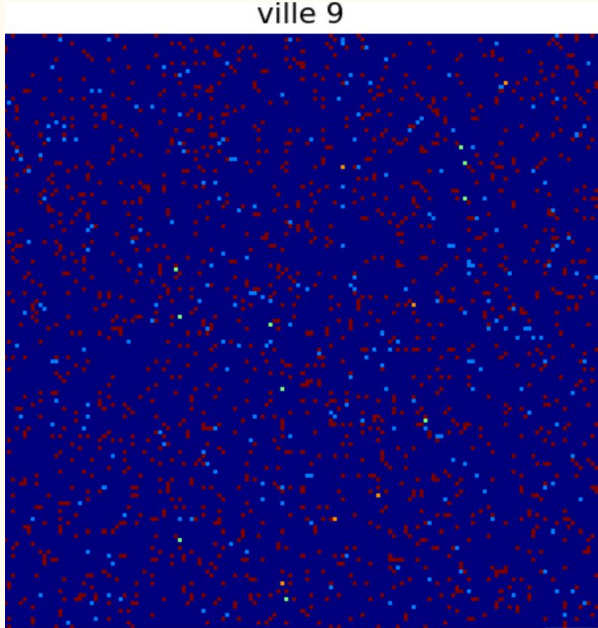
30 jours

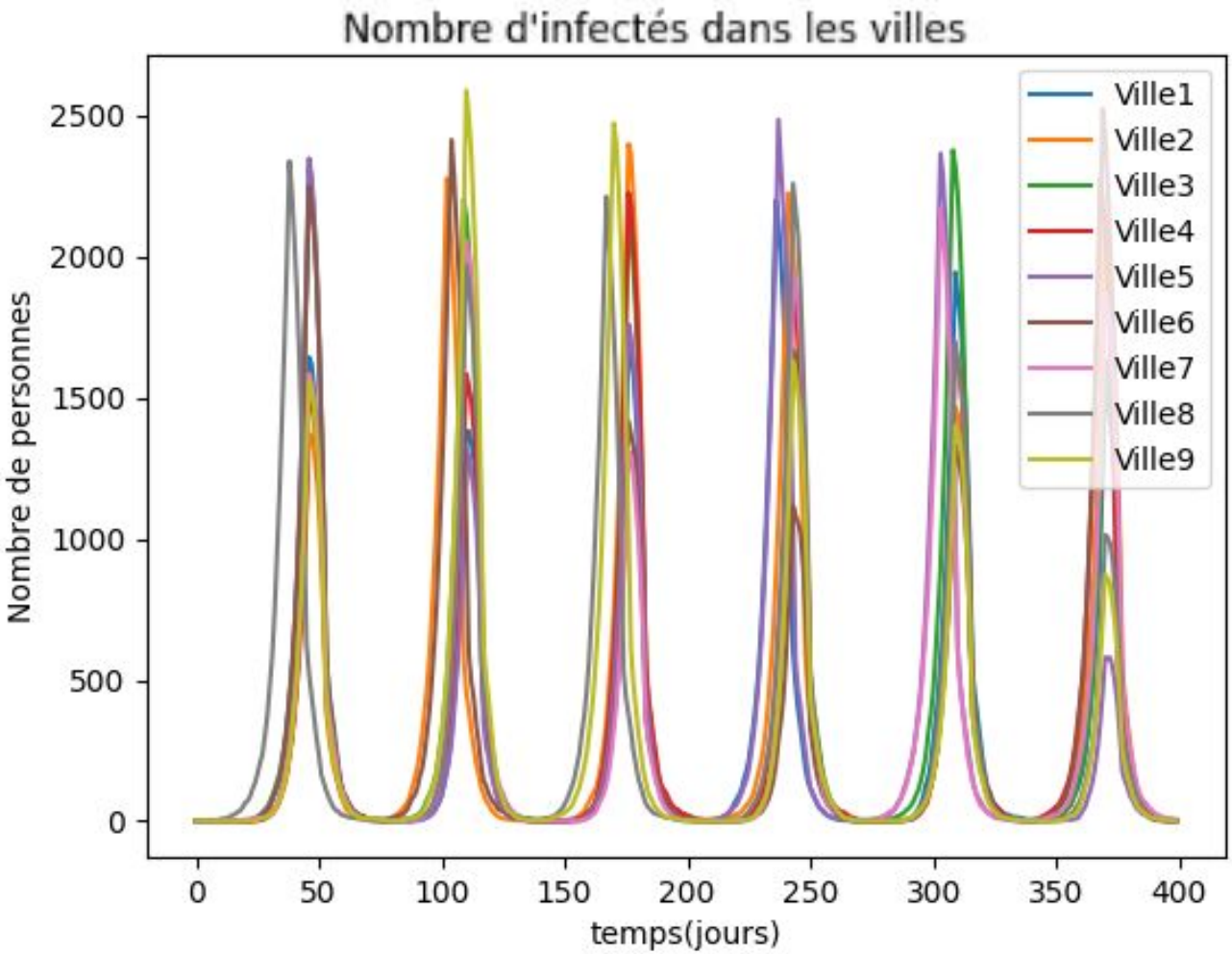


40 jours



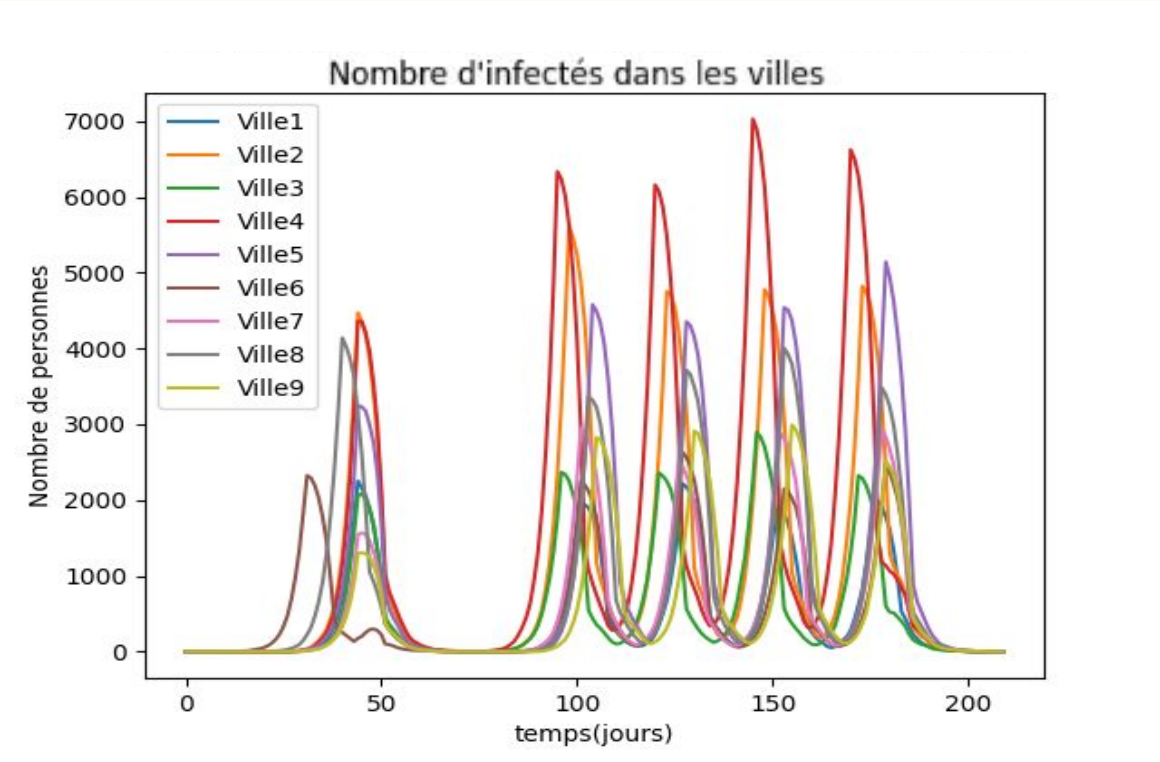
50 jours




Nombre d'infectés dans les villes
Nombre de personnes
temps(jours)
0
500
1000
1500
2000
2500
0
50
100
150
200
250
300
350
400
Ville1
Ville2
Ville3
Ville4
Ville5
Ville6
Ville7
Ville8
Ville9

IV. Améliorations du système :

2- Tailles des villes, matrice de déplacement



84 Ville 1

130 Ville 2

93 Ville 3

146 Ville 4

129 Ville 5

89 Ville 6

95 Ville 7

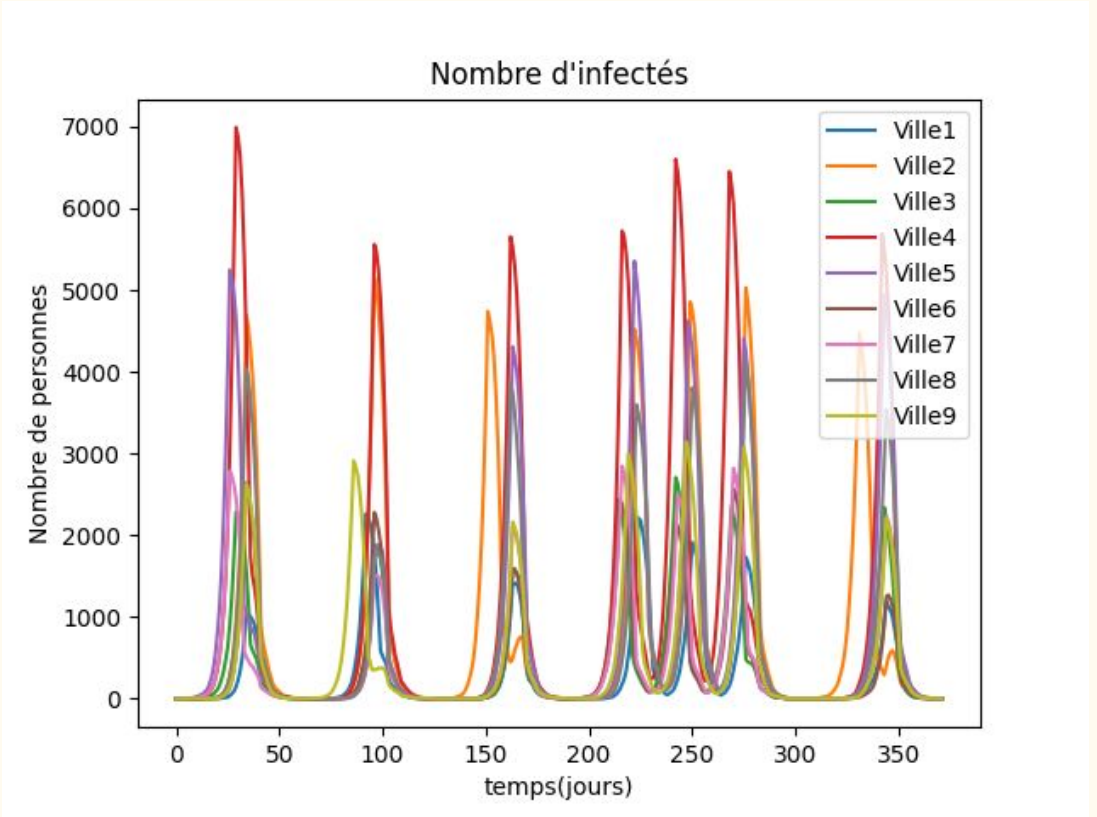
113 Ville 8

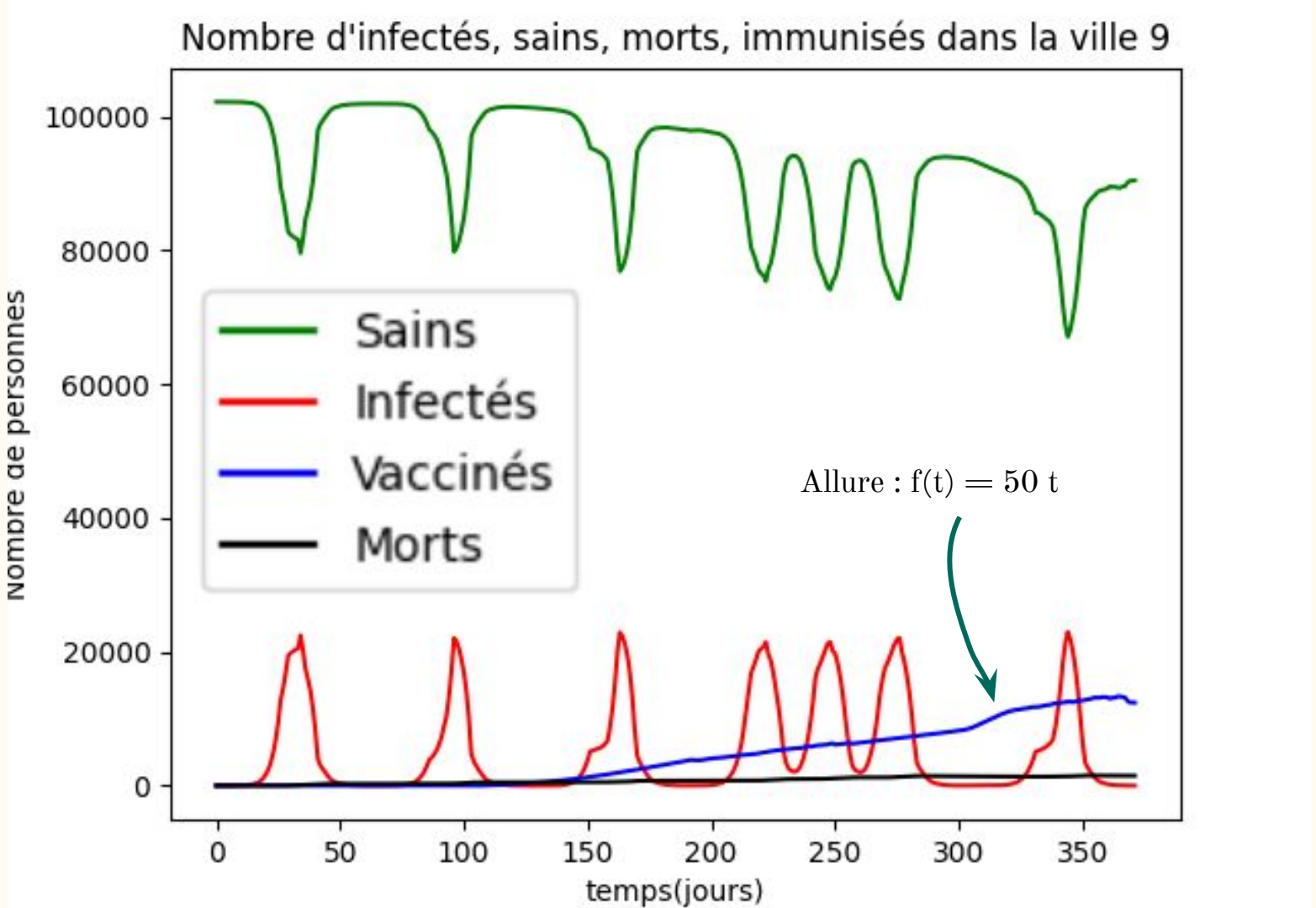
98 Ville 9

V. Simulation et analyse des données obtenue :

1- Résultats n°1

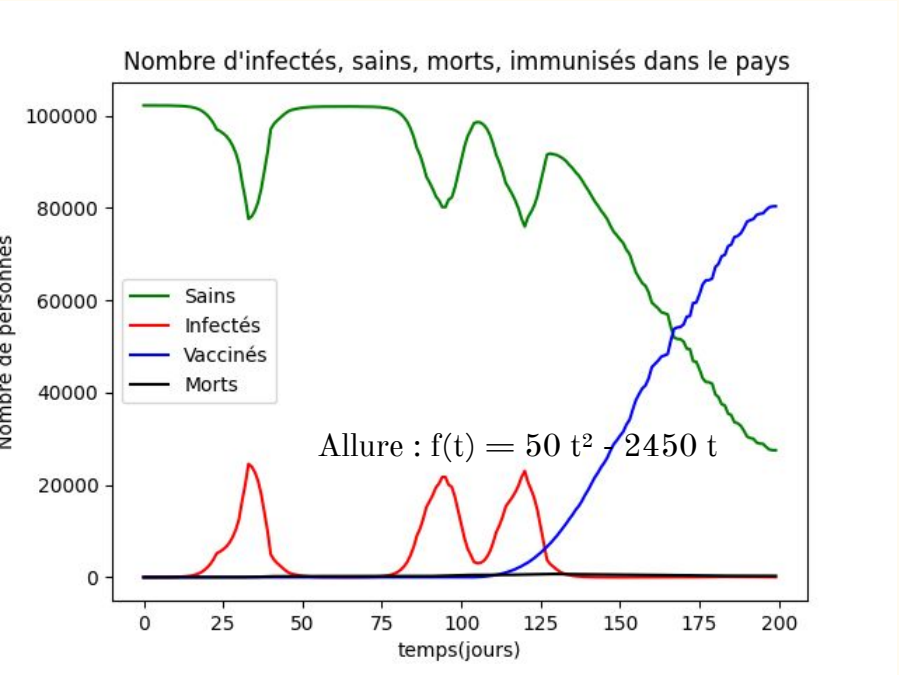
Ici on a choisit une augmentation linéaire du nombre de vaccinés (fct_vax(t) = t)

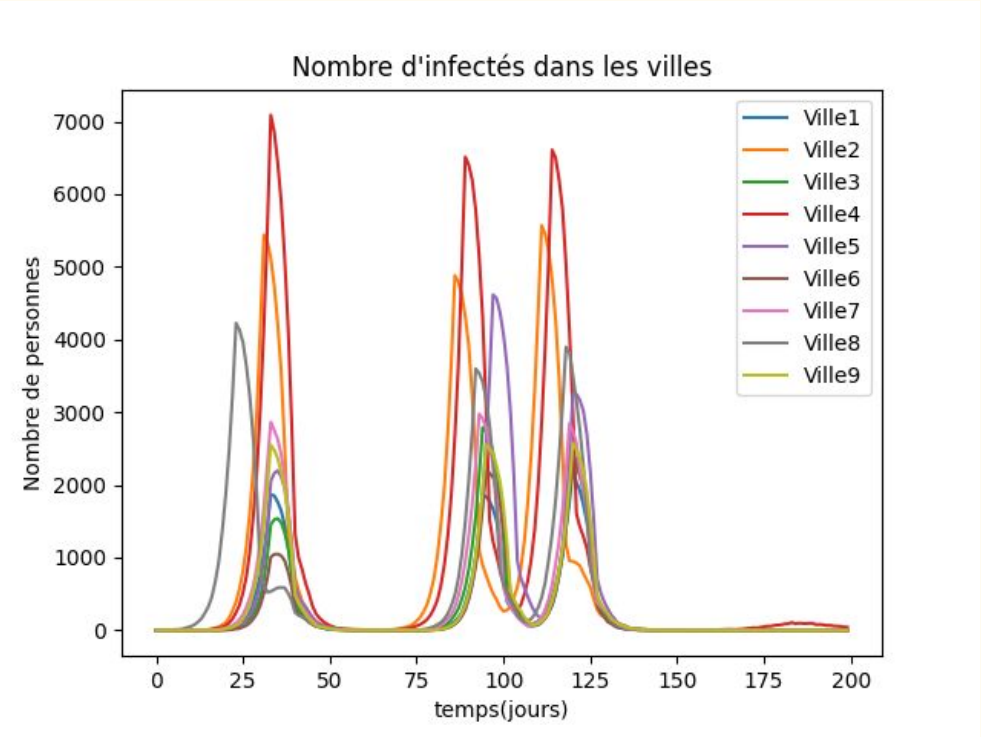



Nombre d'infectés, sains, morts, immunisés dans la ville 9
Nombre de personnes
100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300
350
temps(jours)
Sains
Infectés
Vaccinés
Morts
Allure : f(t) = 50 t

V. Simulation et analyse des données obtenue :

2- Résultats n°2





Ici on a choisi une augmentation quadratique du nombre de vaccinés : fct_vax(t) = t²

VII. Conclusions :

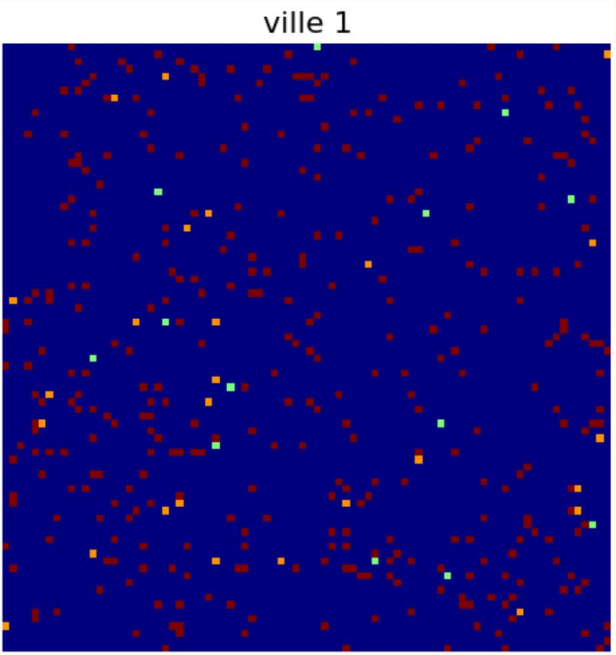
- Une modélisation se doit d'être précise, et réaliste.

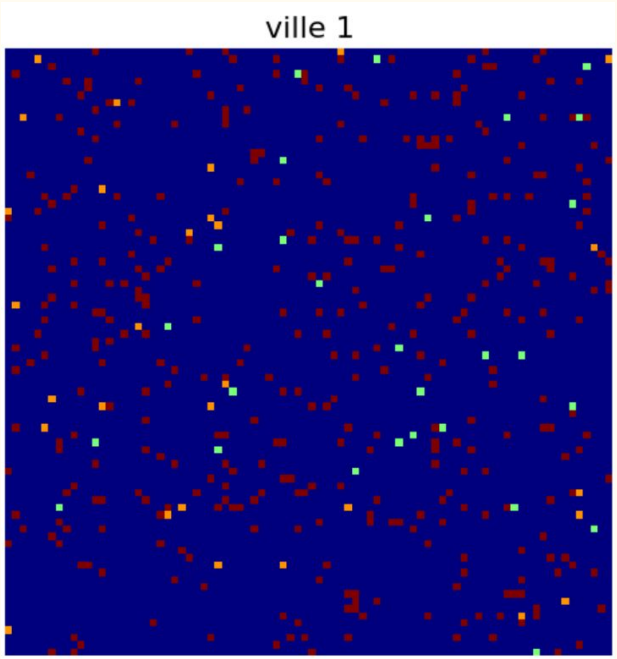
- Limites de puissance d'un ordinateur portable, plusieurs heures pour une simulation. Taille du pays.

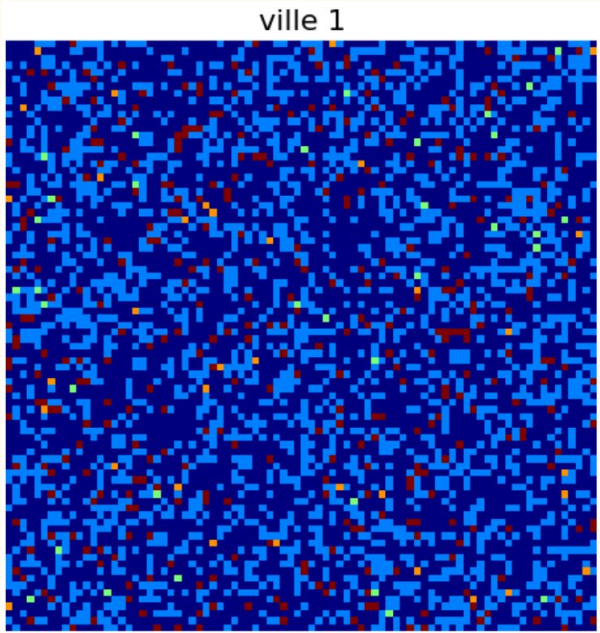
- Un modèle qui peut encore être amélioré sur le plan informatique, pour se rapprocher encore plus de la réalité.

Annexes

Résultats n°3



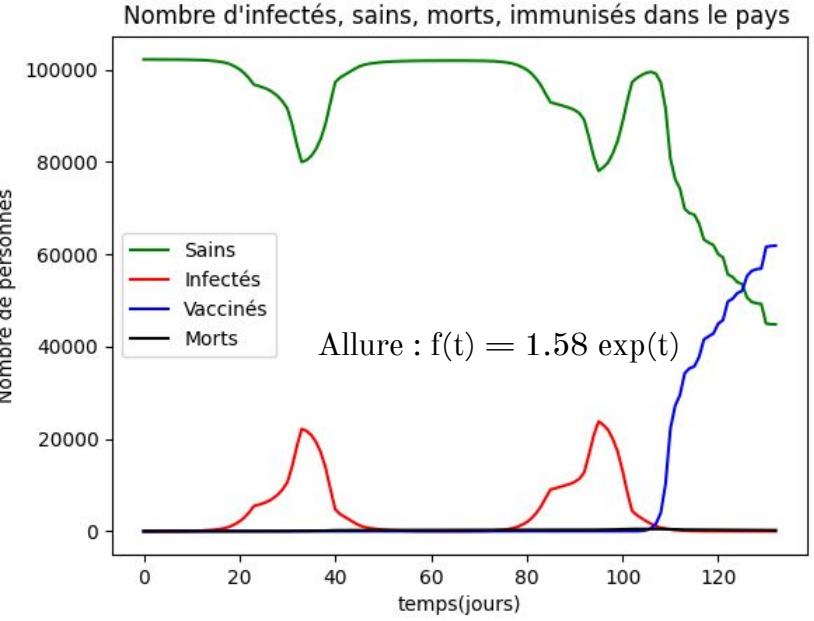


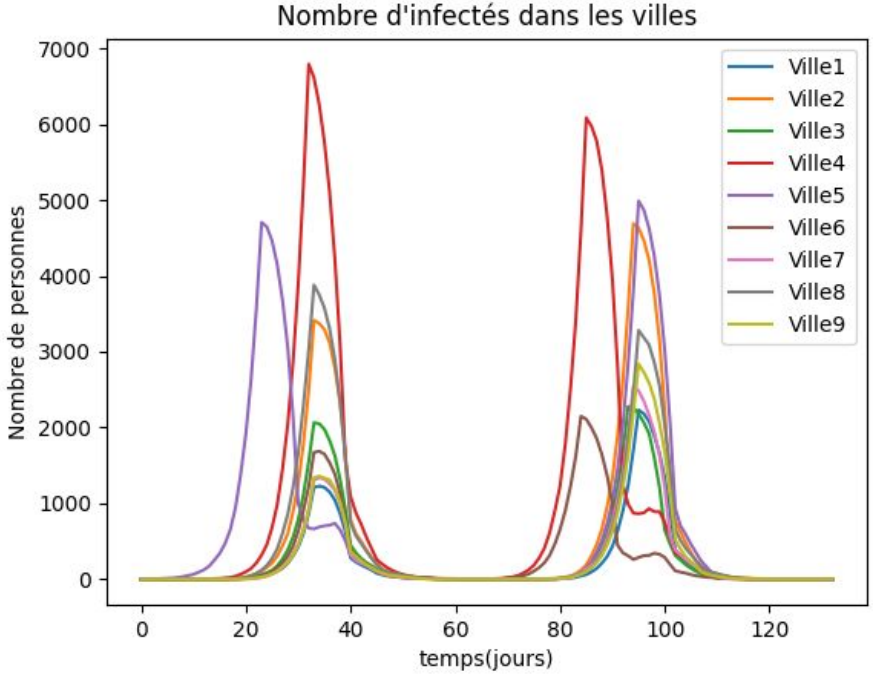


Ici on a choisit une augmentation du nombre de vaccinés nulle à partir d'un certain rang (fct_vax(t) = 5)

Annexes

Résultats n°4





Ici on a choisit une augmentation exponentielle du nombre de vaccinés (fct_vax(t) = exp(t))

Annexes

python > v2_automate_cellulaire.py > ...

```python
# -*- 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


# Implémentation 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
    else:
        return fusion(tri_fusion(T[ : round(len(T) / 2)]), tri_fusion(T[round(len(T) / 2) : ]))
```

Annexes

```python
def bernoulli(p):
    return random() < p

# Creer une ville de taille p (p**2 habitants),
# sous la forme d'une matrice.
# On rajoute à la fin un compteur qui s'actualisera
# des Sain, Contaminé, Rétablit, Mort et case vide presents dans la ville.
def creer_ville (p):
    nb_vide = round(p * p * 0.07) #le nombre de case qui seront vides
    C=[p * p - nb_vide,0,0,0,nb_vide]
    V = [[[0,0] for i in range(p)] for i in range(p)]
    entiers = [i for i in range(p * p)]

    for k in range(nb_vide):
        q = randint(0,len(entiers) - 1 - k)
        i = entiers[q]
        V[i//p][i%p] = [4,0]
        entiers = entiers[:q]+entiers[q+1:] #ainsi on ne peut tirer deux fois la même case

    V.append(C)
    return V


#n villes, la ville i est de taille lp[i] (lp[i]**2 habitants).
def creer_pays(n,lp):
    P=[]
    for i in range(n):
        V = creer_ville(lp[i])
        P.append(V)
    return P
```

Annexes

```
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 déplacent 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])     #montée dans le bus
90                  V[i//k][i%k] = [4,0]         #la case devient donc vide
```

Annexes

```
91      else:
92          #tout le monde monte sauf les personnes tirée 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
```

Annexes

```
# l une liste d'élément, la fonction renvoie k
# si il y a k fois la valeur 1 dans l
def aux1(l):
    c = 0
    for i in l:
        if i == 1:
            c += 1
    return c


# la fonction determine le nombre de voisins
# de l'element en place n de la ville V sont infectés.
def est_vulnerable(n,V):
    k = len(V) - 1
    if V[n//k][n%k][0] == 0: #on ne s'intéresse qu'aux personnes encore saines
        if n == 0:             #coin supérieur gauche
            return aux1([V[0][1][0],V[1][0][0],V[1][1][0]])
        elif n == k - 1:       #coin supérieur droit
            return aux1([V[0][k-2][0],V[1][k-2][0],V[1][k-1][0]])
        elif n == k * (k - 1): #coin inférieur gauche
            return aux1([V[k-2][0][0],V[k-2][1][0],V[k-1][1][0]])
        elif n == k * k - 1:   #coin inférieur droit
            return aux1([V[k-2][k-2][0],V[k-2][k-1][0],V[k-1][k-2][0]])
        elif n//k == 0:        #arrete du haut
            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]])
        elif n//k == k - 1:    #arrete du bas
            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],V[k-2][n%k+1][0]])
        elif n%k == 0:         #arrete de gauche
            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],V[n//k+1][1][0]])
        elif n%k == k-1:       #arrete de droite
            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],V[n//k+1][k-2][0]])
        else:                  #centre
            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],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]])
    else:
        return 0

```

Annexes

```
155  #modifie la ville en contaminant les personnes vulnérables avec une probabilité p.
156  def contamination(V,p):
157      k = len(V) - 1
158      E = []
159      for i in range (k * k):
160          b = False
161          #si une personne a n voisins contaminés, elle a n fois plus de chance de le devenir
162          for j in range(est_vulnerable(i,V)):
163              b = (b or bernoulli(p))
164          if b:
165              E.append(i)
166      for i in E:
167          V[i//k][i%k] = [1,0]
168          V[k][1] += 1
169          V[k][0] -= 1
170
171
```

Annexes

```
172  # Devenir d'une personne infectée; d = durée de la maladie; p3=probabilité de mourrir;
173  # p5=probabilité de redevenir saint, il reste la probabilité que l'infecté reste infecté
174  def apres_inf(V,d,p3,p5):
175      k = len(V) - 1
176      for i in range(k * k):
177          # partie infecté :
178          if V[i//k][i%k][0] == 1:
179  #On passe chaque jour dans apres_inf, on en profite donc pour ajouter 1 au compteur de durée de maladie des infectés
180              V[i//k][i%k][1] += 1
181              if V[i//k][i%k][1] >= d:
182                  #passé la durée de la maldie, la malade peut évoluer
183                  if bernoulli(p3):
184                      V[i//k][i%k] = [3,0]
185                      V[k][1] -= 1
186                      V[k][3] += 1
187                  elif bernoulli(p5/(1-p3)):
188                      V[i//k][i%k] = [0,0]
189                      V[k][1] -= 1
190                      V[k][0] += 1
191                  # sinon, il reste infecté
192          # partie vacciné :
193          if V[i//k][i%k][0] == 2:
194  #On passe chaque jour dans apres_inf, on en profite donc pour ajouter 1 au compteur de durée de vaccin des vaccinés
195              V[i//k][i%k][1] += 1
196              if V[i//k][i%k][1] >= 50:  #le vaccin est supposé efficace 50 jours
197                  V[i//k][i%k] = [0,0]   # la personne redevient saine
198                  V[k][2] -= 1
199                  V[k][0] += 1
```

a

Annexes

```
201
202  # p la portion de gens qui se déplacent en train (àl'échelle nationale) ;
203  # P un pays ; pop sont nombre d'habitants.
204  # La fonction renvoie la matrice de déplacement associée à cette situation.
205  def creer_mat_dep(p,P,pop):
206      n = len(P)
207      m = [[0 for i in range(n)] for i in range(n)]
208      for i in range(n):
209          h = (len(P[i]) - 1) ** 2 # nb d'habitants de la ville i
210          for j in range(n):
211              if i != j:
212                  m[i][j] = round((len(P[j]) - 1) ** 2 * h * p / pop)
213      return m
```

Annexes

```
216 #P un pays; mat_dep la matrice de déplacement
217 def trajet_train(P,mat_dep):
218     n = len(P)
219     gare = []   # va accueuillir les trains, puis les renvoyer dans les bonnes villes
220     for i in range(n):
221         train = []     # va accueuillir tout les voyageur allant à la ville i
222         for j in range(n):
223             wagon = []    # le wagon allant de la ville j à la ville i
224             if i != j:
225                 p = len(P[j]) -1 #population de la ville j
226                 nb = mat_dep[j][i]  #nb de gens qui se deplacent
227                 while nb != 0:
228                     q=randint(0,p*p-1)
229                     etat = P[j][q//p][q%p][0]
230                     if etat != 3 or etat != 4:
231                         wagon.append(P[j][q//p][q%p])
232                         P[j][q//p][q%p] = [4,0]
233                         nb -= 1
234                         P[j][p][etat] -= 1  #on ajoute pas de case vide car on va la remplir juste après
235             wagon = melange_liste(wagon)
236             train += wagon
237         gare.append(train)    # ainsi on maitrise les destinations, le train i va à la ville i.
238
```

Annexes

```
238
239     for i in range(n): #redistribution
240         train = gare[i] #on reparti les personnes du train numéro i (ceux allant à i) dans la ville i
241         p = len(P[i]) - 1
242         case_vide = []
243         for j in range(p * p):
244             if P[i][j//p][j%p][0] == 4: #on regarde les indices des cases vides pour distribuer dans celle-ci
245                 case_vide.append(j)
246         long = len(case_vide)
247         for k in range(len(train)):
248             r = randint(0,long - 1 - k)
249             c = case_vide[r]
250             P[i][c//p][c%p] = train[k]
251             etat = train[k][0]
252             P[i][p][etat] += 1
253             case_vide = case_vide[:r] + case_vide[r+1:]
254     return P
```

Annexes

```python
#P un pays de n ville, on va vacciner nb_vax personnes dans une des villes
def vaccination(nb_vax,P,n):
    q = randint(0, n-1)
    tailleq = len(P[q]) - 1
    vaccinables = []
    for k in range(tailleq * tailleq):
        etat = P[q][k//tailleq][k%tailleq][0]
        if etat == 0 or etat == 1:
            vaccinables.append(k)
    #on ne peut pas vacciner plus de gens que le nombre de gens vaccinables (personnes saines ou infectés)
    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 à 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 où les habitants ne sont plus un couple d'informations, mais juste une seule : leur état.
            V0.append(l0)
        P0.append(V0)
    return P0
```

Annexes

```
#P0 le pays à modèliser (sous forme traduite), n un carré 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')
    # Légende 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 à 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 confinées
            if c1[i] != 0:
                if fst == 0:
                    E = str(i+1)
                    fst = 1
                else:
                    E = E + ", " + str(i + 1)
        if len(E) == 1:
            plt.suptitle("Carte des villes pendant un confinement de la ville "+E,fontsize=26,x=0.5,y=0.1)
        else:
            plt.suptitle("Carte des villes pendant un confinement des villes "+E,fontsize=26,x=0.5,y=0.1)
    else:
        plt.suptitle("Cartes des villes à la date "+str(j),fontsize=26,x=0.5,y=0.1)   #on informe la date
    plt.legend(handles = patches, bbox_to_anchor = (0.90, 0), loc = 2, borderaxespad = 0.,fontsize = 20 )
    plt.savefig('.\carte_simul_v16_date_{}.png'.format(j),dpi = 100)
    plt.close()
```

Annexes

```
# matrice est un matrice carrée, scal est un scalaire.
# la fonction renvoie 1/scal * matrice
def div(matrice,scal):
    n = len(matrice)
    m = [[0 for i in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            m[i][j] = matrice[i][j] * scal
    return m


def simulation_maladie_dans_pays(n,lp,d,t,p1,p2,p3,fct_vax,p5,p6,p7,p8,noms,f):
    """n villes, leurs tailles sont dans lp; d=durée de la maladie (jours); l'expérience a une durée de t jours;
    p1=proba d'être contaminé en étant vulnerable; p2=part de la pop des villes qui se deplace en bus;
    p3=proba de mourrir pour un infecté; fct_vax : rythme de vaccination;
    p5=proba de redevenir saint après une infection; p6=part de la pop du pays qui se déplace en train;
    p7=part de la population du pays contaminée à partir de laquelle confinement nationnale;
    p8=part de la population d'une ville contaminée à partir de laquelle confinement de cette ville"""
    P = creer_pays(n,lp)
    pop = 0
    for i in range(len(lp)):
        pop += lp[i] ** 2     #pop sera le nombre d'habitant du pays
    mat_dep = creer_mat_dep(p6,P,pop)
    etats = ["Sain","Infecté","rétablit","Mort","case vide"] #dans l'ordre (à la place i il y a l'état n°i)
    #contamination d'une personne :
    q = randint(0, n - 1)
    tailleq = len(P[q]) - 1
    while P[q][tailleq][1] != 1: #on attend qu'il y aie effectivement un infecté
        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
```

Annexes

```
359             P[q][tailleq][0] -= 1
360     Nb_0 = [[] for i in range(n)]  #Nb_0[i][t] contiendra le nombre de sains (etat = 0) de la ville i à la date t
361     Nb_1 = [[] for i in range(n)]  #Nb_0[i][t] contiendra le nombre d'infectés (etat = 1) de la ville i à la date t
362     Nb_2 = [[] for i in range(n)]  #Nb_0[i][t] contiendra le nombre de vaccinés (etat = 2) de la ville i à la date t
363     Nb_3 = [[] for i in range(n)]  #Nb_0[i][t] contiendra le nombre de morts (etat = 3) de la ville i à la date t
364     c1 = [0 for i in range(n)]
365     #si c1[i] = 0, la ville i n'est pas confinée, sinon c1[i]=t, la ville i est confinée depuis t jours
366     c2 = 0
367     #si c2 = 0, le pays n'est pas confinée, sinon c2=t, le pays est confinée depuis t jours
368     temps = [i for i in range(t)]
369     for j in range(t):
370         if j >= 100:  #mise en place de la vaccination
371             vaccination(round(fct_vax(j - 100)),P,n)
372             #la fonction fct_vax nous donne le nombre de personne qui seront vaccinés à la date j
373         E = []
374         nbinftot = 0
375         for i in range(n):
376             taillei = len(P[i]) - 1
377             Nb_0[i].append(P[i][taillei][0])
378             Nb_1[i].append(P[i][taillei][1])
379             Nb_2[i].append(P[i][taillei][2])
380             Nb_3[i].append(P[i][taillei][3])
381             if P[i][taillei][1] != 0:   #on fait évoluer la ville seulement si elle contient des infectés
382                 #les infectés qui le sont depuis plus de 7j évolue, deviennent mort, sain ou restent infectés
383                 #les vaccinés qui le sont depuis plus de 50j redeviennent sain
384                 apres_inf(P[i],d,p3,p5)
385                 if c2 != 0: #critère de confinement nationnal
386                     contamination(P[i],p1/10) #on reduit de 10 les trajets dans les villes, et les contaminations
387                     trajet_bus(P[i],p2/10)
```

Annexes

```
388                 else:
389                     if c1[i] == 0: #la ville i n'est pas confinée
390                         if p8>(P[i][taillei][1]/(taillei*taillei)):
391                             contamination(P[i],p1)
392                             trajet_bus(P[i],p2)
393                         else: #on a dépassé le seuil, le confinement de cette ville commence
394                             c1[i] += 1
395                             contamination(P[i],p1/10)
396                             trajet_bus(P[i],p2/10)
397                     elif c1[i]==14: #on est arrivé à la fin du confinement de la ville i
398                         c1[i]=0
399                         contamination(P[i],p1)
400                         trajet_bus(P[i],p2)
401                     else:   #on est en court de confinement de la ville i
402                         c1[i]+=1
403                         contamination(P[i],p1/10)
404                         trajet_bus(P[i],p2/10)
405                 nbinftot+=P[i][taillei][1]
406                 #cette contiendra le nombre d'infecté du pays à la date j à la fin de cette boucle
407         if c2==0 : #il n'y a pas de confinement nationnal en court
408             if p7>(nbinftot/(pop)) :
409                 trajet_train(P, mat_dep)
410                 print(1)
411             else: #on a dépassé le seuil, on déclanche le confinement nationnal
412                 c2+=1
413                 trajet_train(P, div(mat_dep,5))  #on reduit de 5 les déplacements entre villes
414         elif c2==30: #fin du confinement nationnal
415             c2=0
416             c1 = [0 for i in range(n)]
417             trajet_train(P, mat_dep)
418         else: #on est en court de confinement nationnal
419             c2+=1
420             trajet_train(P, div(mat_dep,5))
```

Annexes

```
422         if nbinftot==0: #la maladie s'est éteinte
423             print(j)  #le programme nous informe de la date de l'héradication de la maladie
424             temps = temps[:j+1]
425             break
426         #on produit une image tout les k*f jours, k un entier
427         #1000 est le code indiquant qu'on ne veut pas produire d'image (fait gagner beaucoup en temps de calculs)
428         if j % f == 0 and f != 1000:
429             P0 = traduit(P, n)
430             images(P0, n, noms, etats, c1, c2, j)
431 
432     plt.figure() #on produit le graphe du nombre d'infectés dans chaque ville
433     plt.title("Nombre d'infectés dans les villes")
434     plt.xlabel('temps(jours)')
435     plt.ylabel("Nombre de personnes")
436     for i in range(n):
437         plt.plot(temps,Nb_1[i], label = 'Ville'+str(i+1))
438     plt.legend()
439     plt.savefig('.\graphe_simul_v20_{}.png'.format(1),dpi=100)
440     plt.close()
```

Annexes

```
443  n=9         #n villes, n doit être un carré d'entier naturel
444  lp=[84,130,93,146,129,89,95,113,98] #les tailles des villes, lp[i] est la taille de la ville i
445  d=7         #d=durée de la maldie en jours
446
447  t=3     #l'expérience a une durée de t jours
448  p1=0.07     #p1=proba d'etre contaminé en etant vulnerable
449  p2=0.9      #p2=part de la pop qui se deplace en bus tout les jours
450
451  p3=0.0061  #p3=proba de mourrir pour un infecté
452
453  def fct_vax(t):
454      f = t ** 4 * cos(t)
455      return (abs(f) + f) /2
456          #rythme de vaccination une fois qu'elle est déclanchée (100 jours)
457
458  p5=0.9      #p5=proba de redevenir saint
459
460  p6=0.01     #p6 = part de la pop d'une ville qui se déplace en train
461  p7=0.20     #p7 = taux de contamination du pays à partir duquel :
462          #  moins de déplacement entre ville,
463          #  et confinement de toute les villes pendant 30 jours
464  p8=0.25     #p8 = taux de contamination du pays à partir duquel :
465          #  moins de déplacement dans cette ville,
466          #  moins de propagation pendant 14 jours
467
468  noms=["ville "+str(i+1) for i in range(n)]
469  f = 100
470
471  simulation_maladie_dans_pays(n,lp,d,t,p1,p2,p3,fct_vax,p5,p6,p7,p8,noms,f)
```