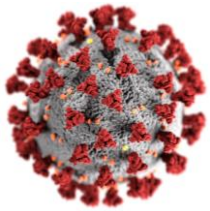
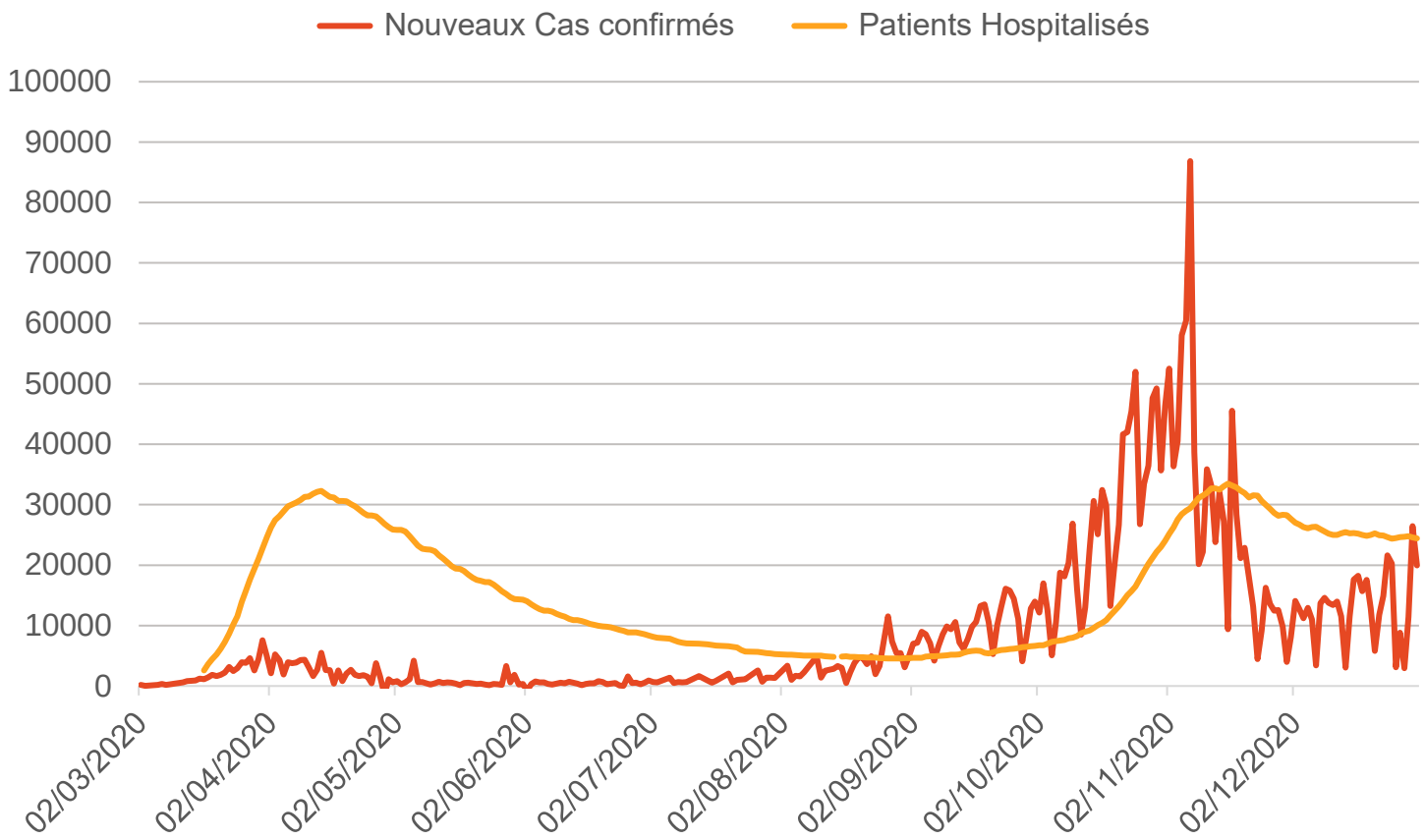
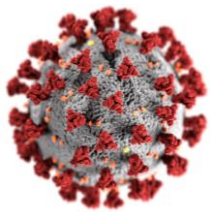
Modélisation d'une épidémie

Mise en évidence des principaux facteurs de la propagation d'un virus



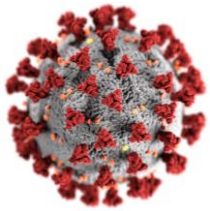
Évolution française du Covid-19 en 2020





Déroulé de la présentation

- ✓ Modélisation sous forme de système d'équations différentielles
- ✓ Résolution numérique avec la méthode d'Euler
- ✓ Modèle discret / « physique » d'une épidémie
- ✓ Étude de la validité des modèles, comparaison des deux
- ✓ Mise en évidence des facteurs favorisant la propagation



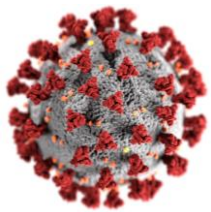
Modèle Sains Infectés Rétablis (SIR)

$$Nombre\ de\ nouveaux\ rétablis = Nombre\ d'infectés * \lambda = \lambda * I$$

$$Nombre\ de\ nouveaux\ cas = Nombre\ d'infectés * \frac{S}{n} * M * P = \frac{IS}{n} * MP = \sigma \frac{IS}{n}$$

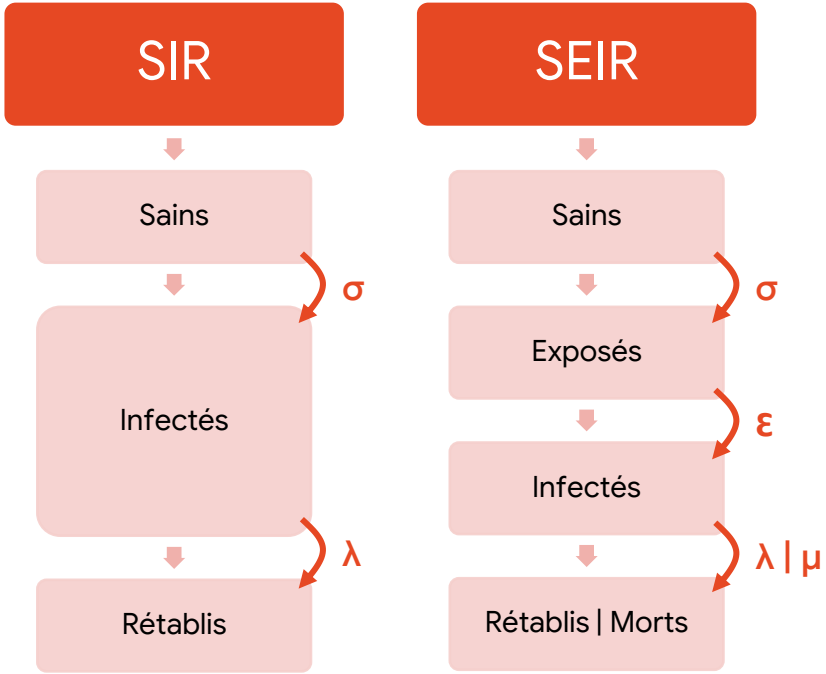
$$\begin{cases} \frac{dS}{dt}(t) = -\sigma * \frac{S(t) * I(t)}{n} \\ \frac{dI}{dt}(t) = \sigma * \frac{S(t) * I(t)}{\mathrm{n}} - \lambda * I(t) \\ \frac{dR}{dt}(t) = \lambda * I(t) \end{cases}$$

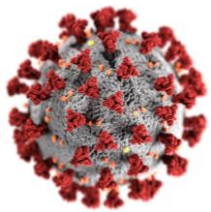
Ne se résout pas à la main



Modèle SEIR

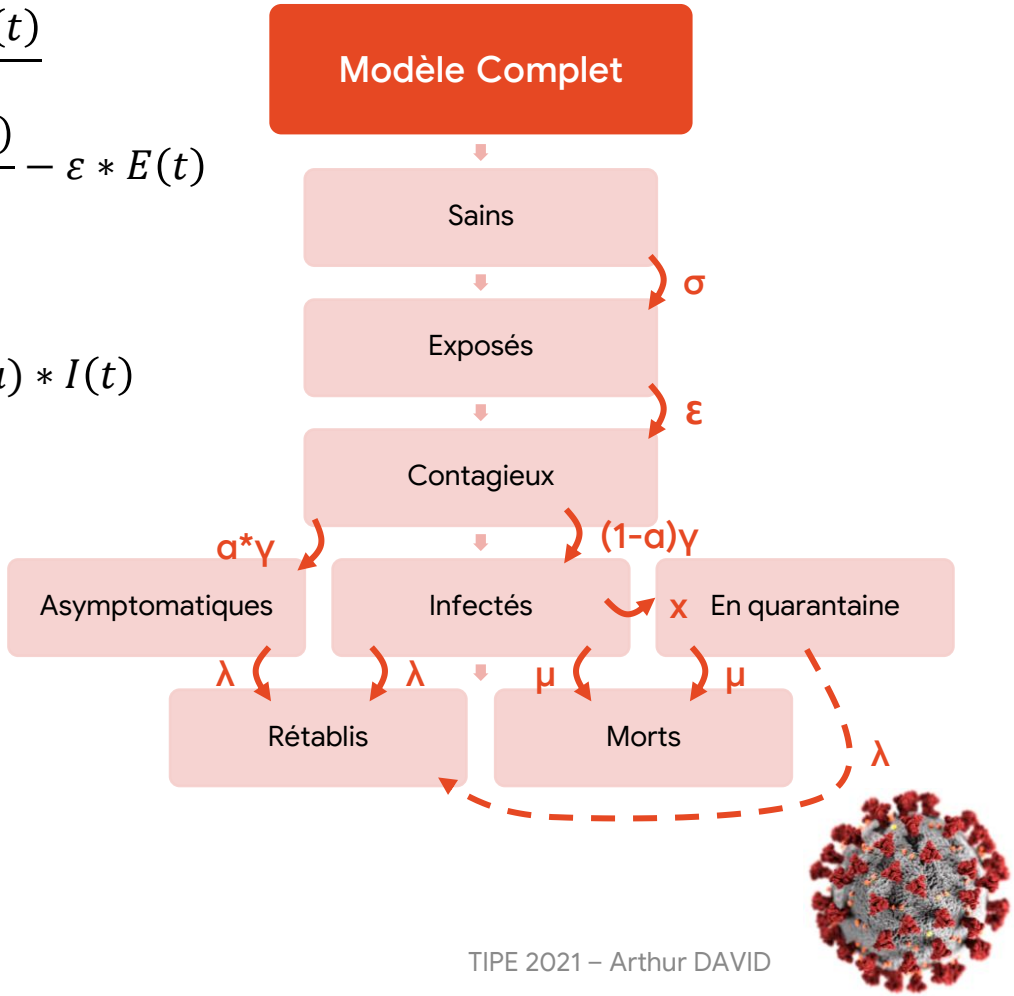
$$\begin{cases} \frac{dS}{dt}(t) = -\sigma * \frac{S(t) * I(t)}{n} \\ \frac{dE}{dt}(t) = \sigma * \frac{S(t) * I(t)}{n} - \boldsymbol{\varepsilon} * \boldsymbol{E(t)} \\ \frac{dI}{dt}(\mathrm{t}) = \boldsymbol{\varepsilon} * \boldsymbol{E(t)} - \lambda * I(t) - \boldsymbol{\mu} * \boldsymbol{I(t)} \\ \frac{dR}{dt}(t) = \lambda * I(t) \\ \frac{\boldsymbol{dM}}{\boldsymbol{dt}}(\mathbf{t}) = \boldsymbol{\mu} * \boldsymbol{I(t)} \end{cases}$$



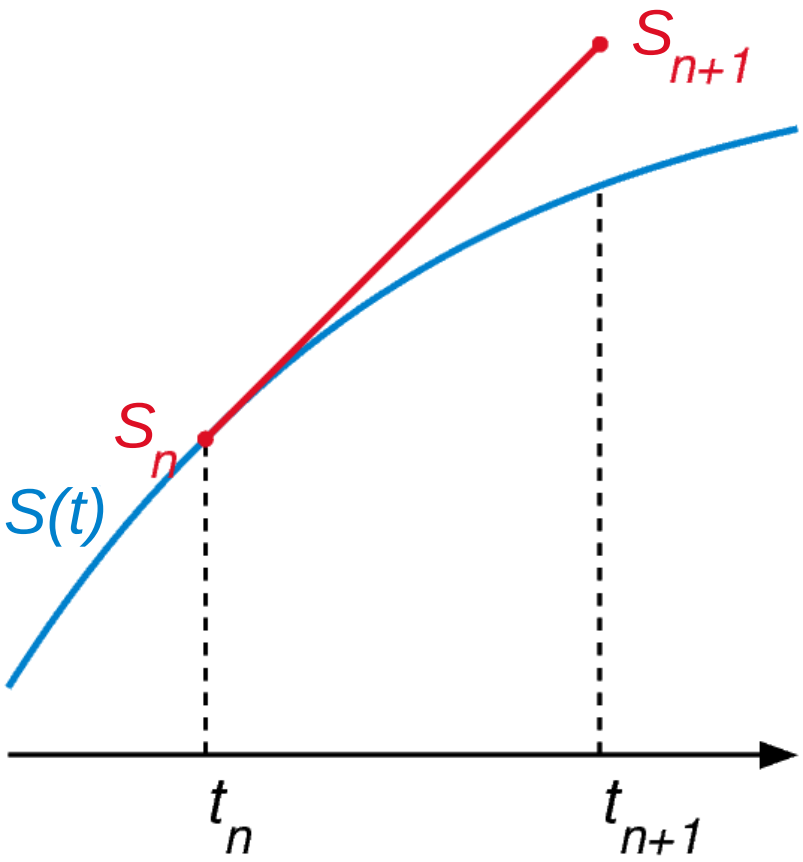


Modèle amélioré

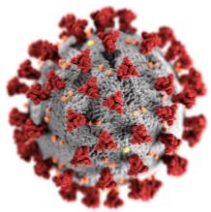
$$
\begin{cases}
\frac{dS}{dt}(t) = -\sigma * S(t) * \frac{I(t)+C(t)+A(t)}{\mathrm{n}} \\
\frac{dE}{dt}(t) = \sigma * S(t) * \frac{I(t)+C(t)+A(t)}{\mathrm{n}} - \varepsilon * E(t) \\
\frac{dC}{dt}(\mathrm{t}) = \varepsilon * E(t) - \gamma * C(t) \\
\frac{dI}{dt}(\mathrm{t}) = (1-\alpha)\gamma * C(t) - (\lambda+\chi+\mu) * I(t) \\
\frac{dA}{dt}(t) = \alpha\gamma * C(t) - \lambda * A(\mathrm{t}) \\
\frac{dQ}{dt}(t) = \chi * I(\mathrm{t}) - (\lambda+\mu) * Q(t) \\
\frac{dR}{dt}(t) = \lambda * (I(t)+A(t)+Q(t)) \\
\frac{dM}{dt}(t) = \mu * (I(t)+Q(t))
\end{cases}
$$



Méthode d'Euler



$$S(t+\Delta t) = S(t) + \Delta t * \frac{dS(t)}{dt}$$
$$= S(t) + \Delta t * \frac{S(t)*I(t)}{n} * \sigma$$

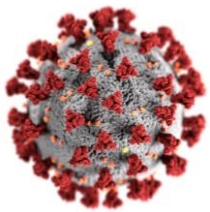


Implémentation en Python

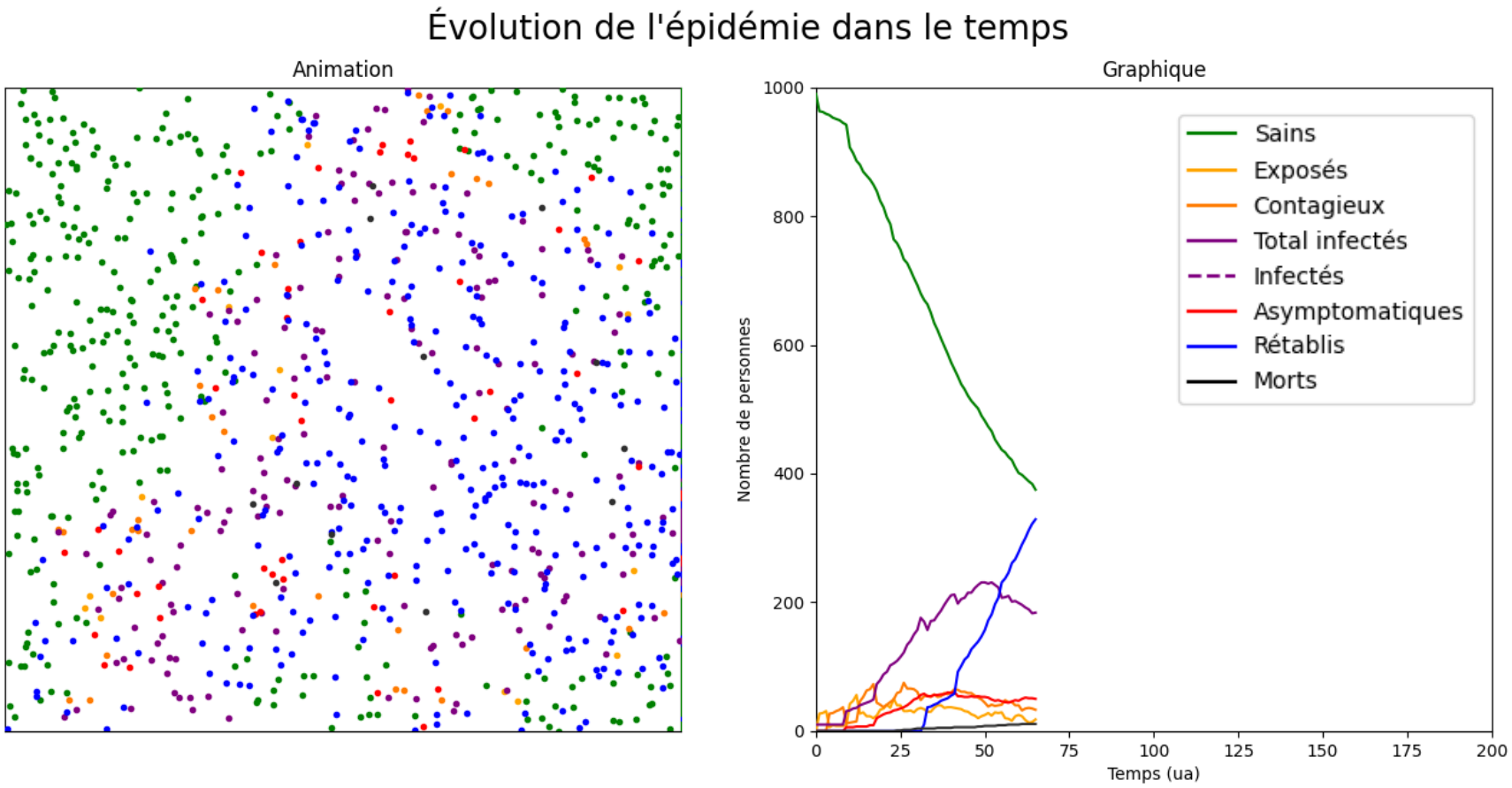
```
def propagation_SEIR(self,σ,ε,λ,μ):
    n=self.n
    dS=σ/n*self.sains*self.infectés*dt
    dE=ε*self.exposés*dt
    dR=λ*self.infectés*dt
    dM=μ*self.infectés*dt
    self.sains += -dS
    self.exposés += dS-dE
    self.infectés += dE-dR-dM
    self.rétablis += dR
    self.morts += dM
```

Calcul des variations

On les somme aux groupes



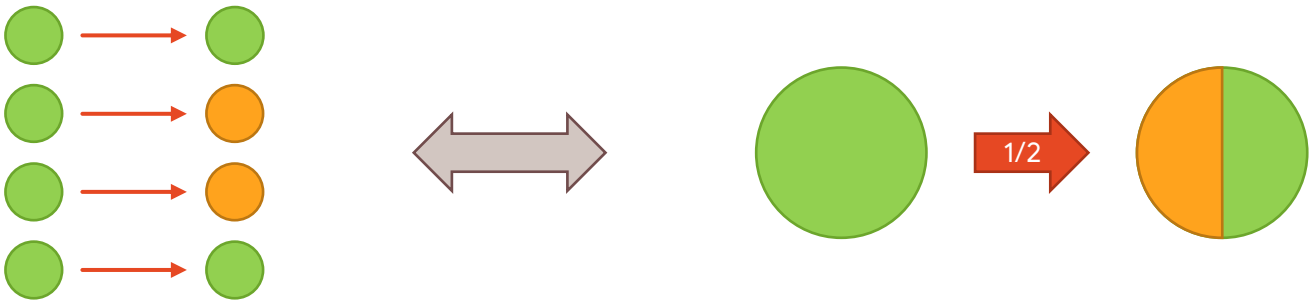
Modèle Discret



Aspect théorique du paramétrage

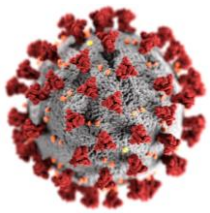
X *compte le nombre de changements d'états*

$X \hookrightarrow B(n;\delta)$ *donc* $E(X) = n * \delta$



Probabilité **δ=1/2**

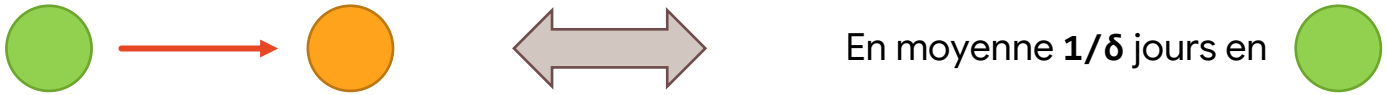
Proportion **δ=1/2**



Aspect théorique du paramétrage

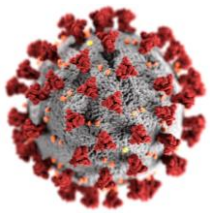
Y *compte le nombre de jours avant le changement d'état*

$\mathrm{Y} \hookrightarrow G(\delta)$ *donc* $E(Y) = 1/\delta$



Probabilité **δ**

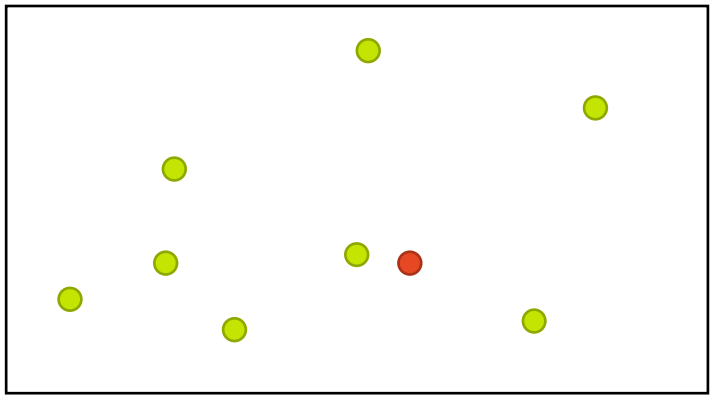
En moyenne **1/δ** jours en

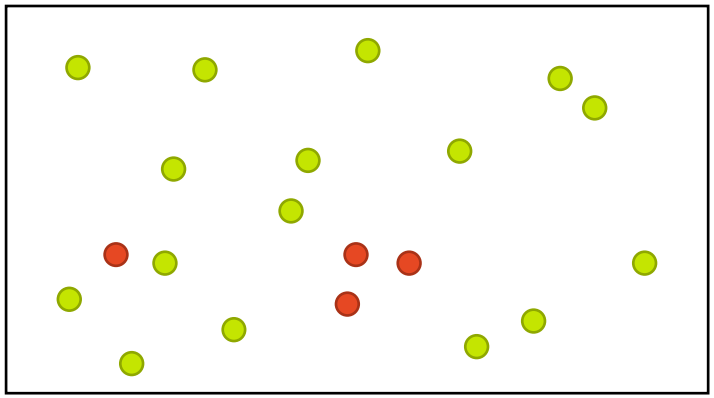


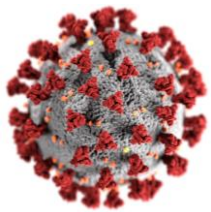
Coefficient de transmission

$$densité = \frac{n_{points} * aire_{point}}{aire_{espace}}$$

$$\sigma = \sigma_0 * densité = \sigma_0 * \frac{n_{points} * aire_{point}}{aire_{espace}}$$

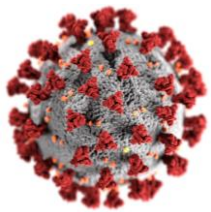




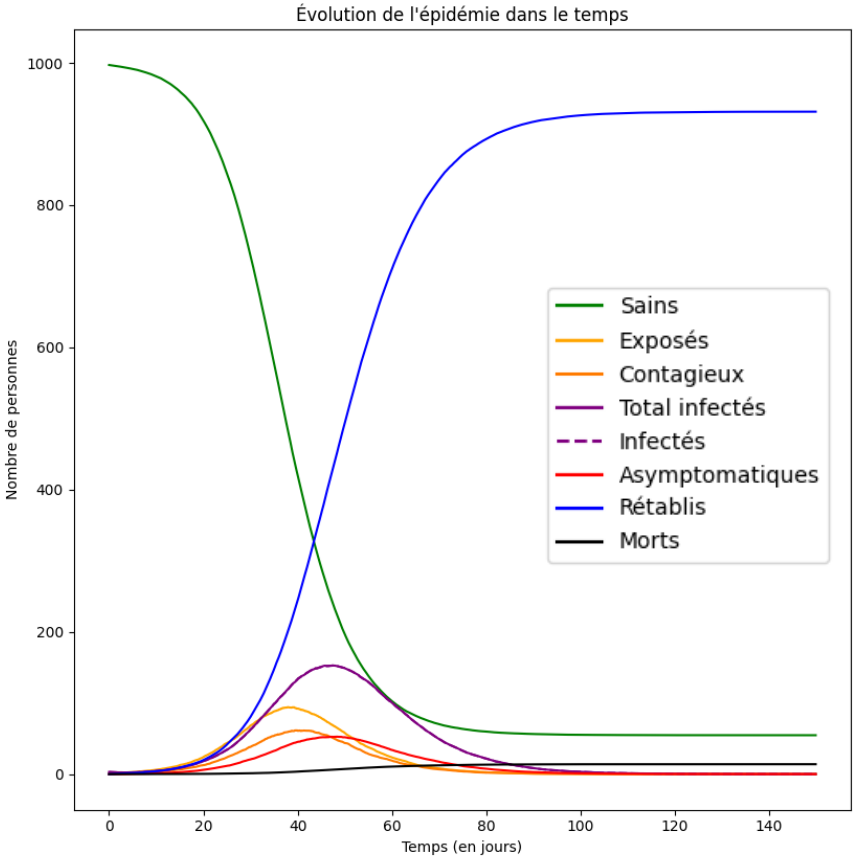


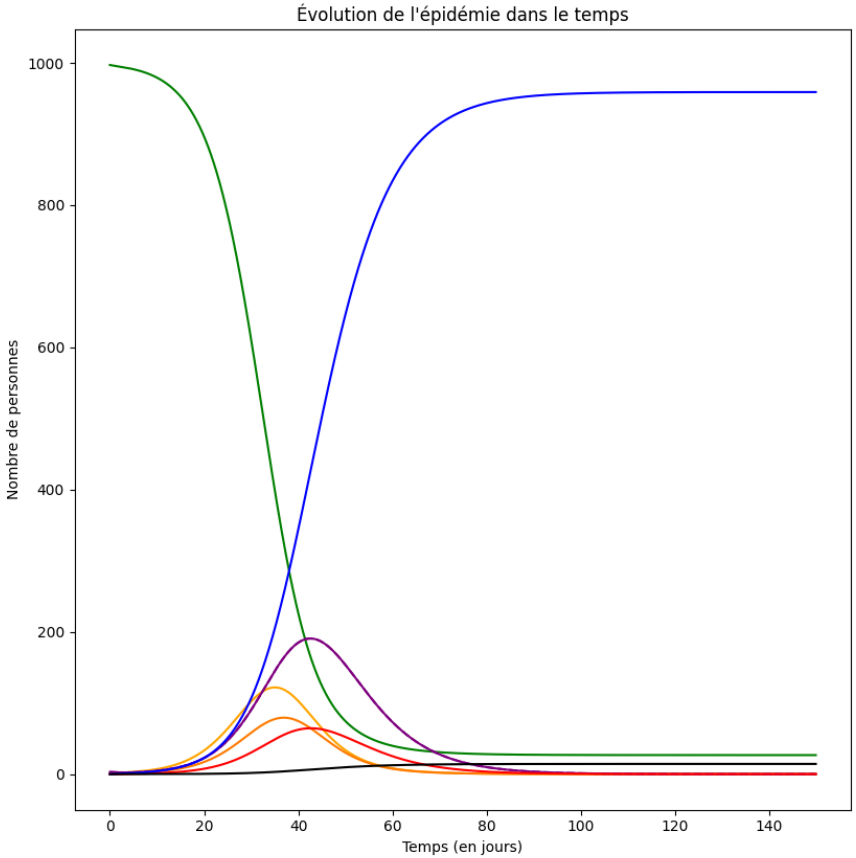
Paramétrage des modèles

Paramètres	Valeurs en France
σ	$R_0 * \lambda = 3/8$
ε	1/3 (3 jours)
γ	1/2 (2 jours)
λ	1/8 (8 jours)
α	0,25
μ	$0{,}02 * \lambda = 0{,}0025$



Comparaison des deux modèles
sans confinement

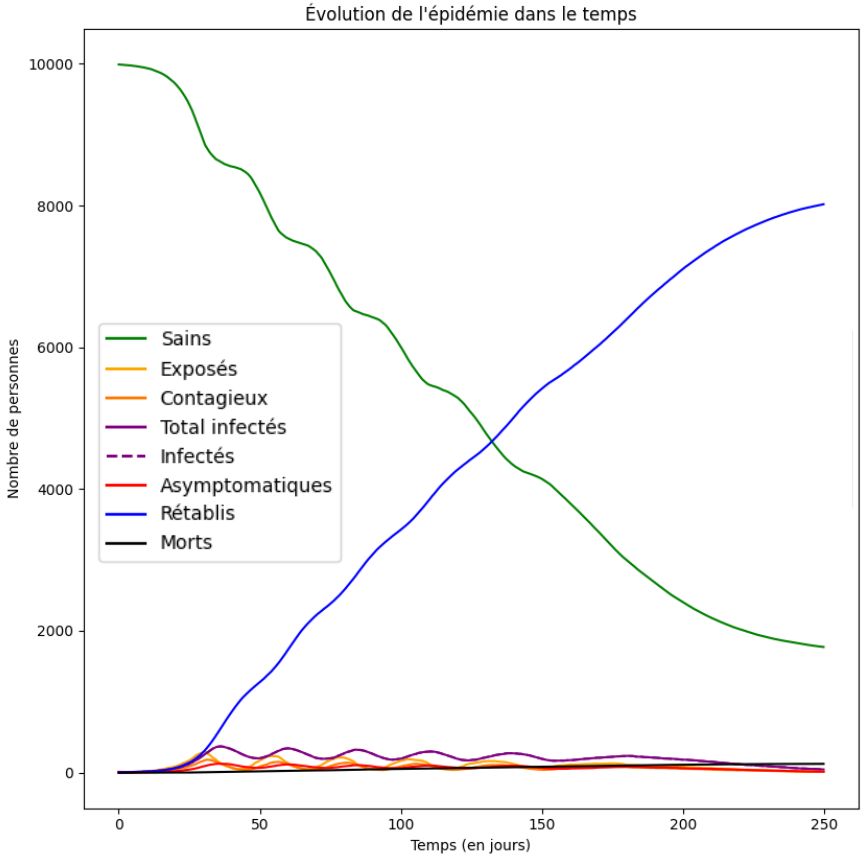




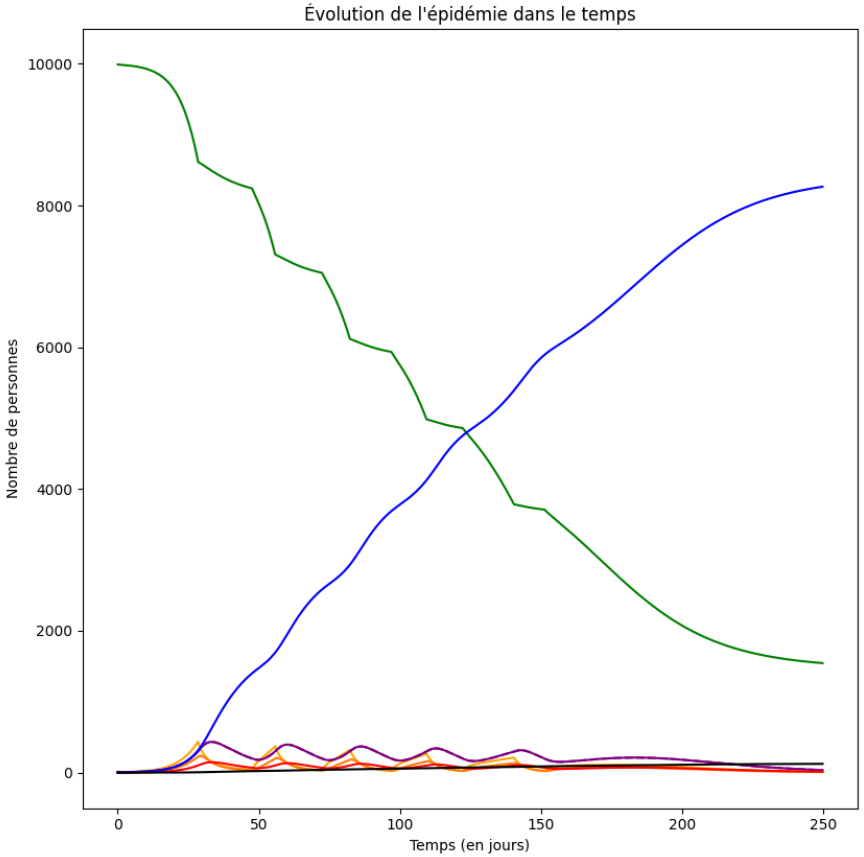
Discret

Mathématique

Comparaison des deux modèles
avec confinement

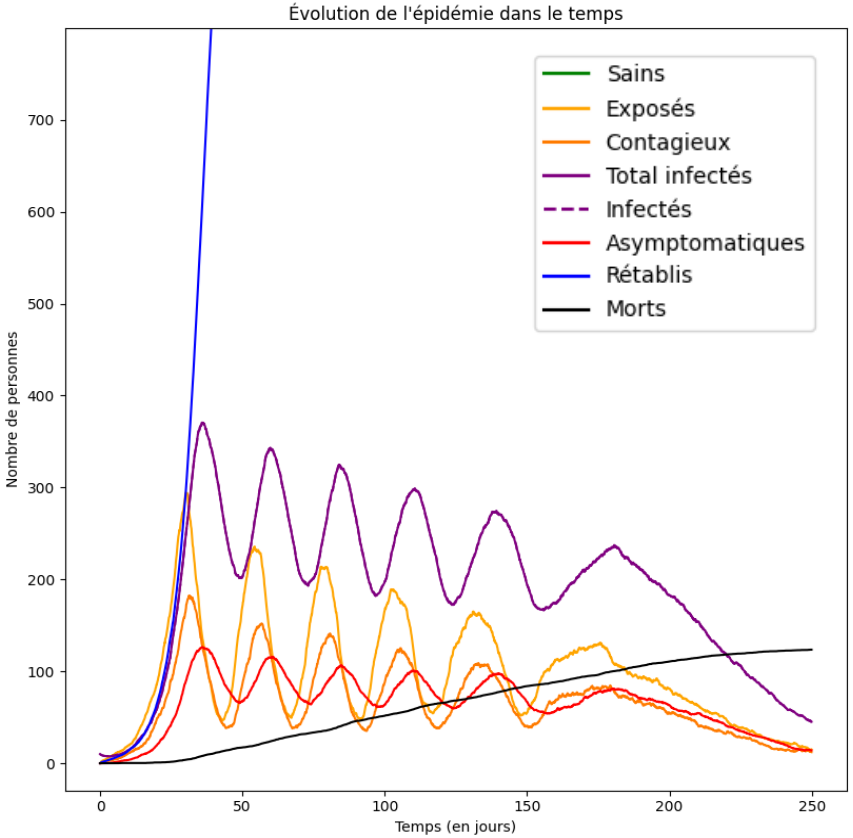


Discret

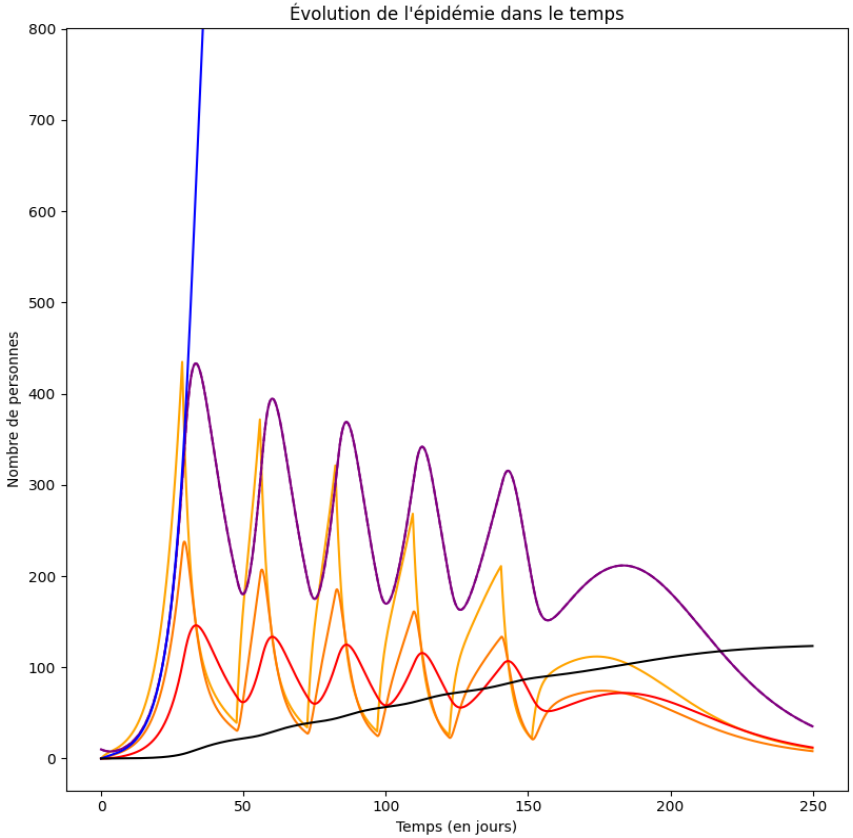


Mathématique

Comparaison des deux modèles
avec confinement



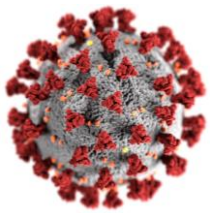
Discret



Mathématique

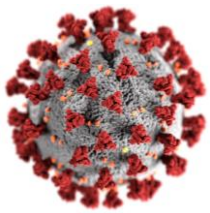
Comparaison des deux modèles

	Modèle Discret
Avantages	❖ Plus de flexibilité (comportements) ❖ Instinctif
Inconvénients	❖ Complexité élevée (mémoire et vitesse) ❖ Long à coder

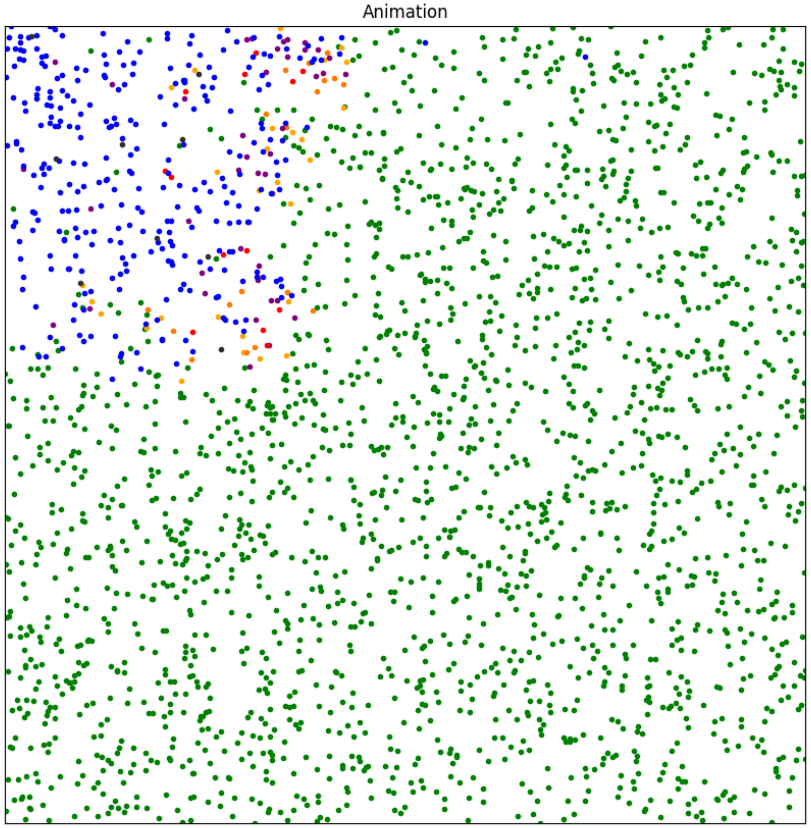


Mesures sanitaires

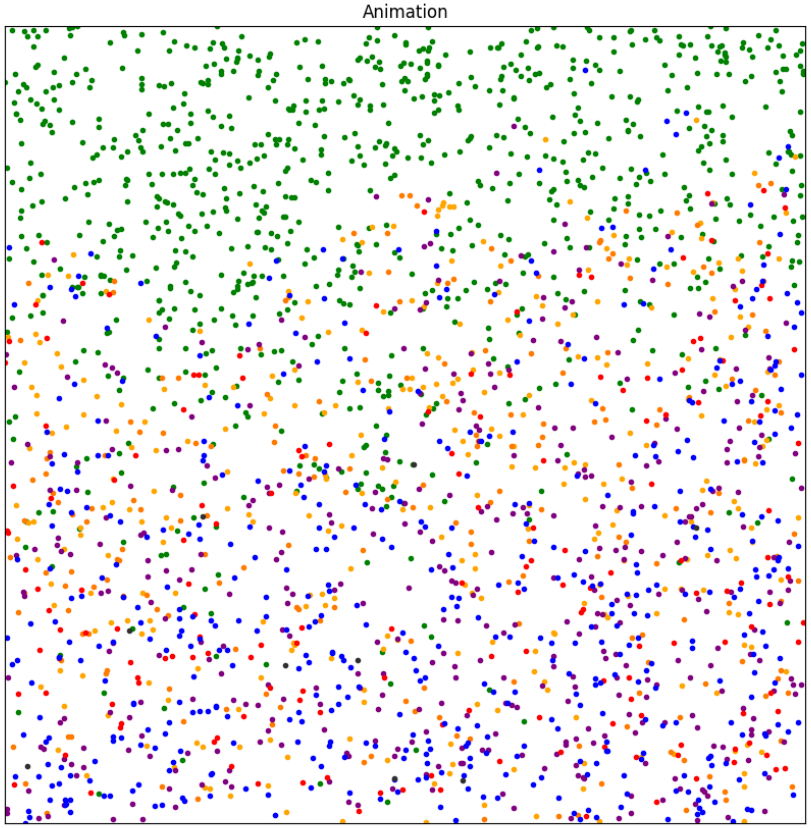
Paramètres	Modèle Discret
`r`	Rayon de propagation entre deux points. Influe sur la densité donc sur σ ; correspond aux gestes barrières.
`pas`	Distance arbitraire parcourue par un point en une journée. Détermine le brassage dans la population.
`PPC,SDC,SFC` (Confinement)	Paramètres du confinement
`χ`	Paramètre de la quarantaine



Pas de déplacement

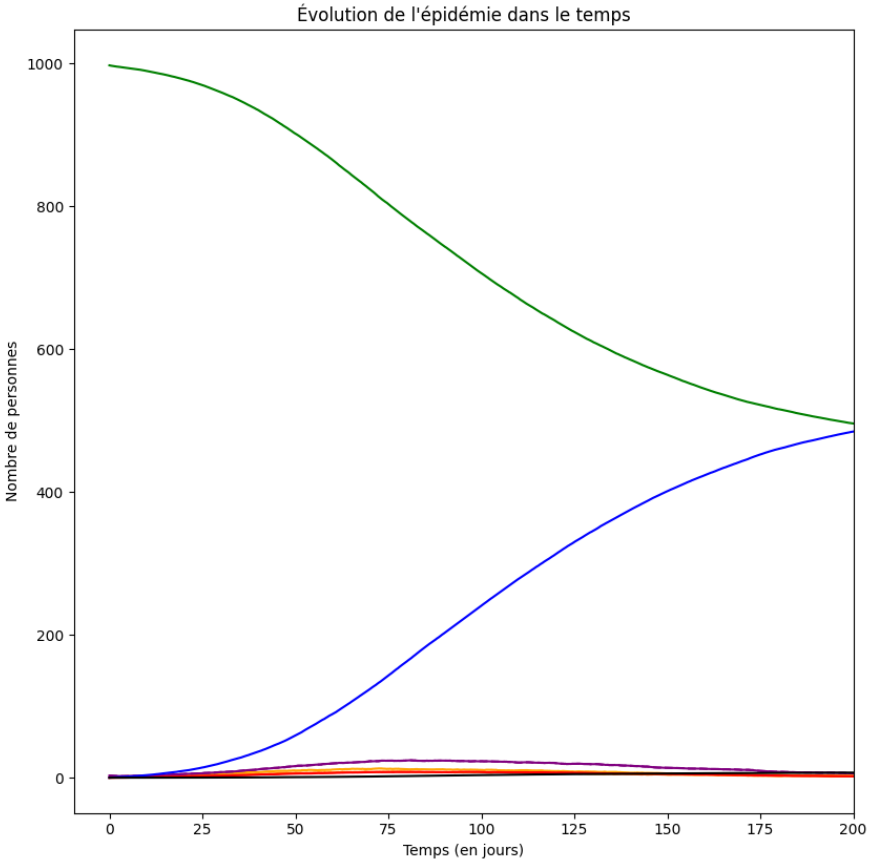


Pas = 0,5

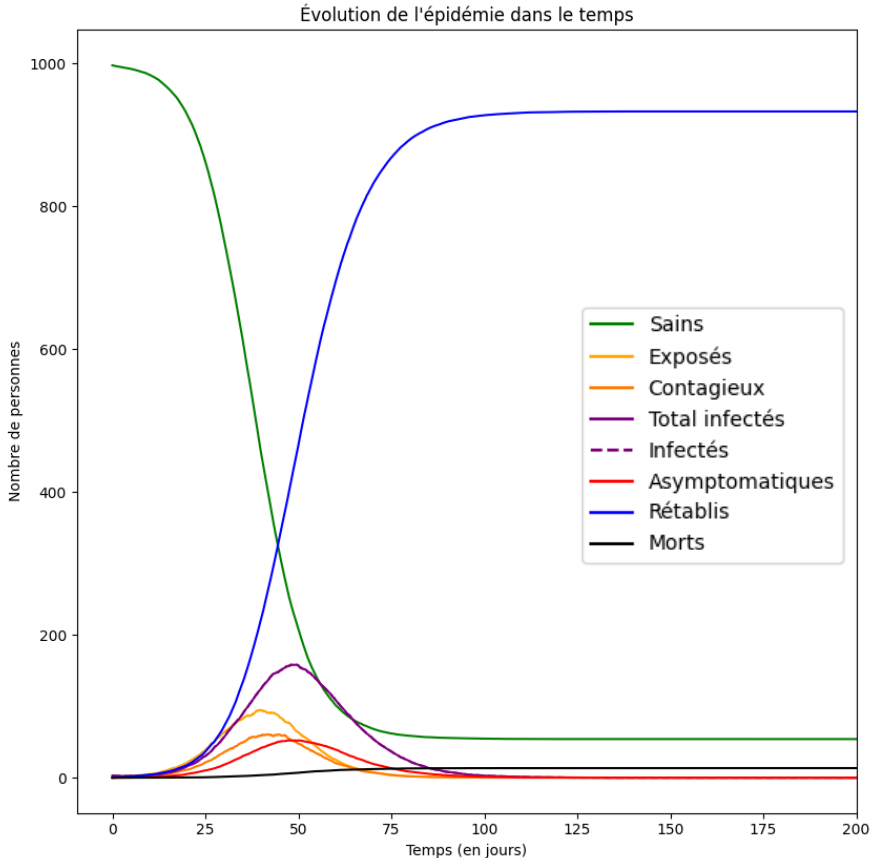


Pas = 5

Pas de déplacement

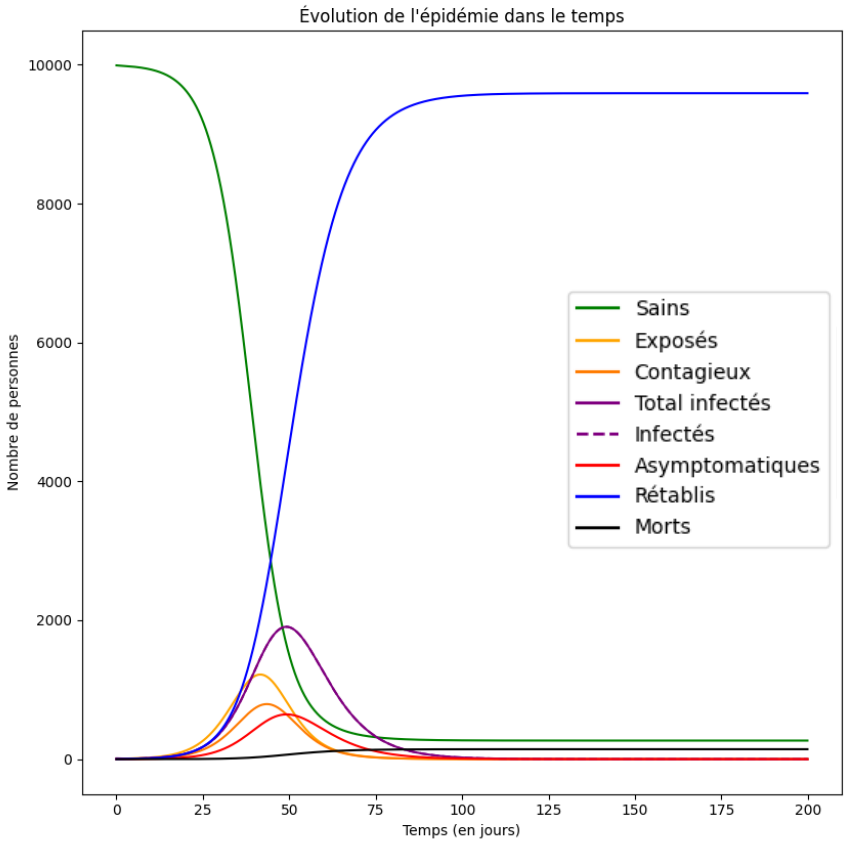


Pas = 0,5

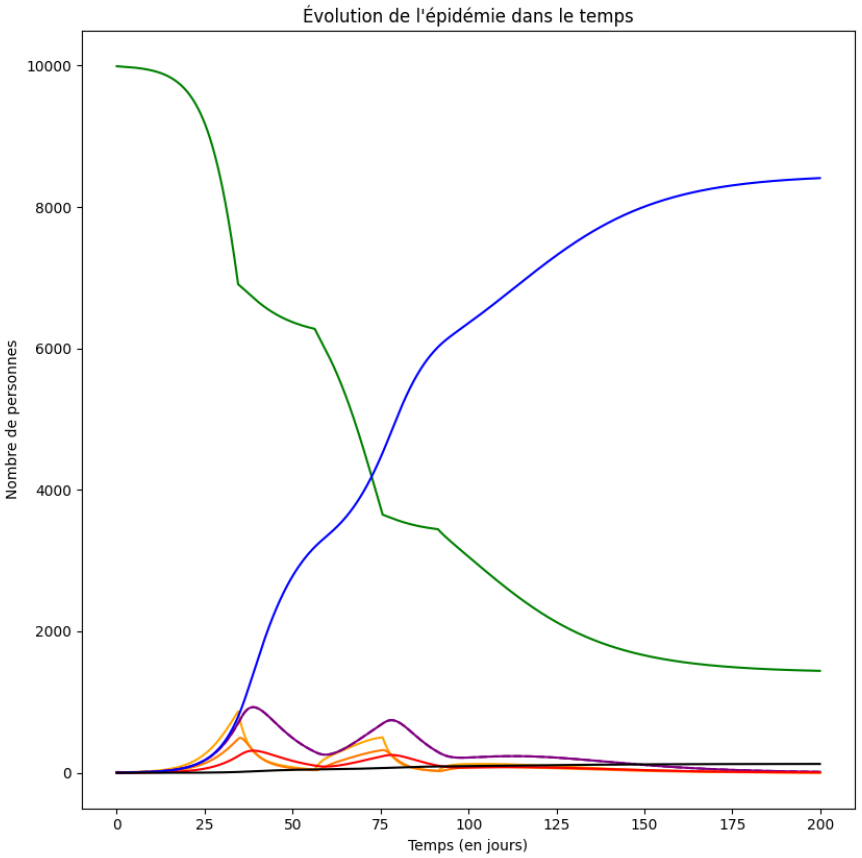


Pas = 5

Confinement

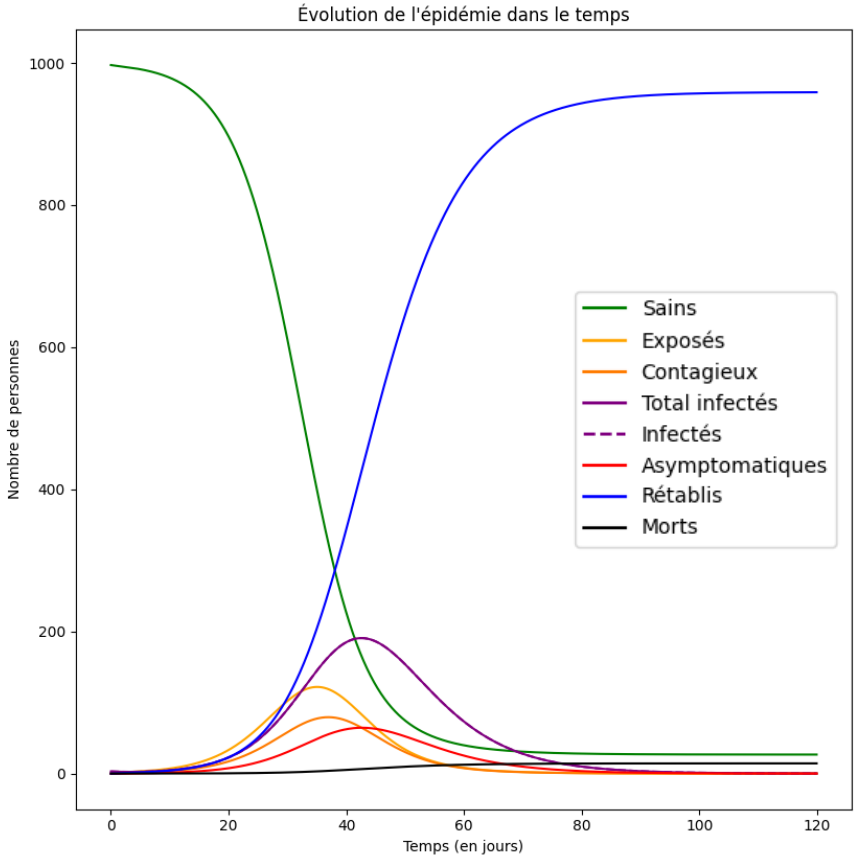


PPC = 0

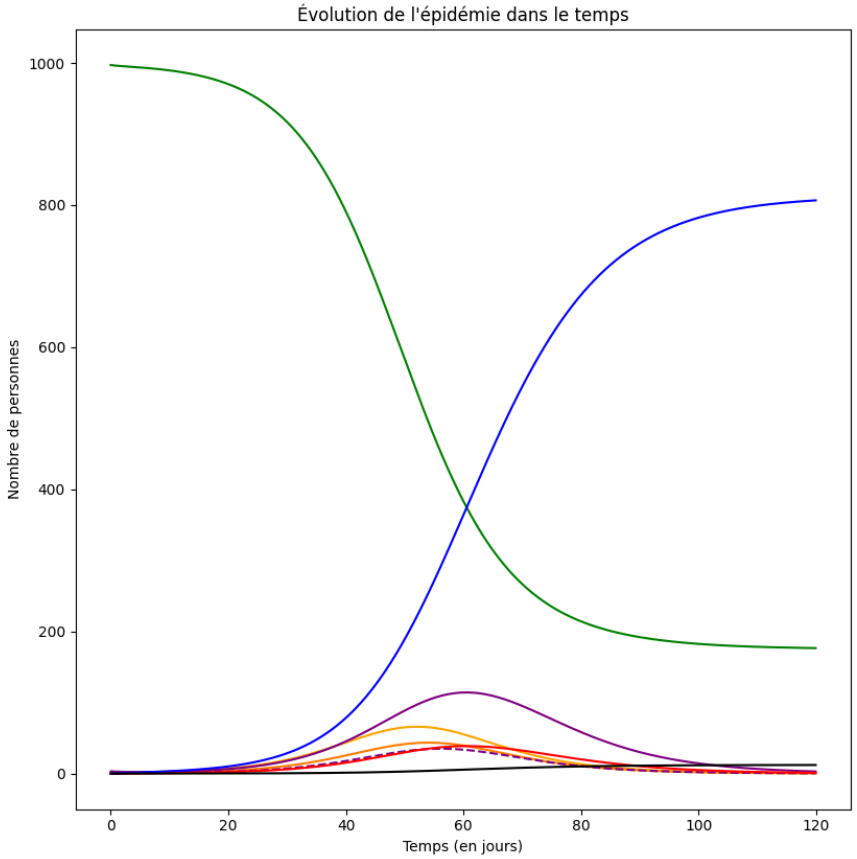


PPC = 2/3

Quarantaine



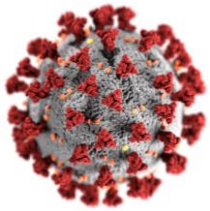
χ=0



χ=2/3

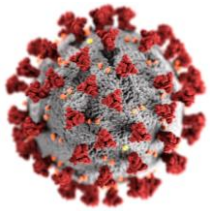
Conclusion

- ✓ Mesures anti-brassage
- ✓ Modèles à très court terme
- ✓ Limités par le manque de variables extérieures



Bibliographie

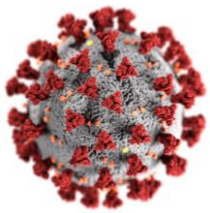
- Photo cellule covid-19 : CDC/ Alissa Eckert, MSMI; Dan Higgins, MAMS
- Schéma méthode d'Euler : Wikipédia - Méthode d'Euler
- Données du covid-19 en France : data.gouv.fr



Annexes

- Programme du modèle discret
  1) Classe population
  2) Fonctions de simulation
  3) Fonctions d'exploitation des résultats

- Programme du modèle mathématique
  1) Classe population
  2) Simulation et graphique



```python
"""Modèle Discret"""

import random as rd
import csv
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from math import pi,cos,sin,ceil,sqrt

class population(object):
    def __init__(self,n=0,r=0):
        """Crée une population de n individus, de direction donnée, dans un espace carré de coté r."""
        pos=[[rd.random()*r for _ in range(n)],[rd.random()*r for _ in range(n)]] #Positions random
        vx,vy=[],[]
        for _ in range(n):
            direction=rd.random()*2*pi #Directions aléatoires ...
            vx+=[cos(direction)] #... selon x
            vy+=[sin(direction)] #... selon y
        self.n = n #Nombre d'individus
        self.r = r #Longueur (ua) du coté du carré de l'étude
        self.x = pos[0] #Coordonnées sur x
        self.y = pos[1] #Coordonnées sur y
        self.vx = vx #direction selon x
        self.vy = vy #direction selon y
        self.sains = [i for i in range(n)] #Indices des individus ... sains
        self.exposés = [] #... contaminés mais pas contagieux (et sans symptômes)
        self.contagieux = [] #... contagieux mais pas encore de symptômes apparents
        self.infectés = [] #... infectés
        self.asymptomatiques = [] #... infectés mais sans symptômes
        self.enQuarantaine = [] #... infectés mais identifiés et isolés
        self.rétablis = [] #... rétablis
        self.morts = [] #... décédés
        self.confinés = [] #... confinés (repectent le confinement).
        self.seDéplacent = [True for _ in range(n)] #Booléen indiquant si l'individu se déplace ou pas
.
```

```
    def infecter(self,n=1):
        """Contamine n éléments sains de la population (aléatoire car leur position l'est)."""
        i=0
        for _ in range(n):
            while not i in self.sains:
                i+=1
            self.infectés+=[i]
            self.sains.remove(i)
            i-=1


    def immuniser(self,n=0):
        """Immunise n éléments dans la population."""
        i=0
        for _ in range(n):
            while not i in self.sains:
                i+=1
            self.rétablis+=[i]
            self.sains.remove(i)
            i-=1


    def confiner(self,p):
        """Définit la partie de la population qui sera confinée et celle qui ne respectera pas le conf
inement."""
        pool=[-1] #Liste des éléments déjà confinés pour pas contaminer deux fois la même personne
        for i in range(p):
            random=-1 #Un élément déjà dans pool pour entrer dans la boucle while
            while random in pool: #On vérifie que l'élément choisi n'a pas déjà été confiné
                random=rd.randint(0,self.n) #Élément aléatoire
            pool+=[random]
            self.confinés+=[i] #Confinement
```

```
def propagation(self,dt,pas,r,σ,ε,γ,λ,α,μ,χ,confinement):
    """Déplace chaque élément d'une distance pas et propage le virus à l'instant suivant."""
    b=self.r
    X,Y,vX,vY=self.x,self.y,self.vx,self.vy
    dS,dE,dC,dQ,dRi,dRq,dRa,dMi,dMq=[False for _ in range(self.n)],[],[],[],[],[],[],[],[]

    #On détermine quels éléments se déplacent.
    if confinement :
        for p in self.confinés:
            self.seDéplacent[p]=False
    else :
        for p in self.confinés:
            self.seDéplacent[p]=True

    for p in self.morts: self.seDéplacent[p]=False
    for p in self.enQuarantaine: self.seDéplacent[p]=True

    #On déplace chaque élément qui est dans seDéplacent, rebond si besoin.
    for p in [p for p in range(self.n) if self.seDéplacent[p]]:
        X[p]+=pas*dt*vX[p]
        Y[p]+=pas*dt*vY[p]

        if X[p] > b :       X[p] , vX[p]  =  2*b-X[p] , -vX[p]
        elif X[p] < 0 :     X[p] , vX[p]  =  -X[p] , -vX[p]
        if Y[p] > b :       Y[p] , vY[p]  =  2*b-Y[p] , -vY[p]
        elif Y[p] < 0 :     Y[p] , vY[p]  =  -Y[p] , -vY[p]
```

```
        #On détermine quels individus passent à l'état d'après selon certains paramètres.
        for p in self.exposés:
            if rd.random()<ε*dt: dE+=[p]

        for p in self.contagieux:
            if rd.random()<γ*dt: dC+=[p]
            if self.seDéplacent[p]:
                for s in self.sains:
                    if self.seDéplacent[s] and rd.random()<σ*dt and sqrt((X[p]-X[s])**2 +(Y[p]-
Y[s])**2)<r : dS[s]=True

        for p in self.infectés:
            if rd.random()<μ*dt: dMi+=[p]
            elif rd.random()<λ*dt: dRi+=[p]
            elif rd.random()<χ*dt: dQ+=[p]
            if self.seDéplacent[p]: #Si une personne saine trop proche de p, proba σ d'être infectée.
                for s in self.sains:
                    if self.seDéplacent[s] and rd.random()<σ*dt and sqrt((X[p]-X[s])**2+(Y[p]-
Y[s])**2)<r : dS[s]=True

        for p in self.enQuarantaine:
            if rd.random()<μ*dt: dMq+=[p]
            elif rd.random()<λ*dt: dRq+=[p]

        for p in self.asymptomatiques:
            if rd.random()<λ*dt: dRa+=[p]
            if self.seDéplacent[p]:
                for s in self.sains:
                    if self.seDéplacent[s] and rd.random()<σ*dt and
sqrt((X[p]-X[s])**2 +(Y[p]-Y[s])**2)<r :
                        dS[s]=True
```

```
        #On les fait changer d'état.
        for p in [p for p in range(self.n) if dS[p]]:
            self.exposés+=[p]
            self.sains.remove(p)

        for p in dE:
            self.contagieux+=[p]
            self.exposés.remove(p)

        for p in dC:
            if rd.random()<α: self.asymptomatiques+=[p]
            else: self.infectés+=[p]
            self.contagieux.remove(p)

        for p in dQ:
            self.enQuarantaine+=[p]
            self.infectés.remove(p)

        for p in dRi:
            self.rétablis+=[p]
            self.infectés.remove(p)

        for p in dRq:
            self.rétablis+=[p]
            self.enQuarantaine.remove(p)

        for p in dRa:
            self.rétablis+=[p]
            self.asymptomatiques.remove(p)

        for p in dMi:
            self.morts+=[p]
            self.infectés.remove(p)

        for p in dMq:
            self.morts+=[p]
            self.enQuarantaine.remove(p)
```

```
"""Simulations dans la mémoire ou dans des fichiers csv."""«

nomsParDéfaut=['x.csv','y.csv','s.csv','e.csv','c.csv','i.csv','q.csv','a.csv','r.csv','m.csv']
def copy(objet): return [e for e in objet] #Copie d'une liste


def simulation(P,duree,dt,pas,r,σ,Ɛ,γ,λ,α=0,μ=0,χ=0,SDC=1,SFC=1):
    """Fait la simulation de la population P sur une certaine duree selon certaines paramètres."""
    attributs=[P.x,P.y,P.sains,P.exposés,P.contagieux,P.infectés,P.enQuarantaine,P.asymptotiques,P.rét
ablis,P.morts]
    #On copie les données de la population dans des listes pour toute la simulation
    X,Y,S,E,C,I,Q,A,R,M=[[copy(a)] for a in attributs]
    confinement=False
    for t in [dt*t for t in range(ceil(duree/dt))]:
        nInfectés,nVivants=len(P.infectés),P.n-len(P.morts)
        confinement=nInfectés > SDC*nVivants or (nInfectés > SFC*nVivants and confinement)
        P.propagation(dt,pas,r,σ,Ɛ,γ,λ,α,μ,χ,confinement) #On passe à l'instant d'après

        #On enregistre la position des individus et leur état
        X+=[copy(P.x)]
        Y+=[copy(P.y)]
        S+=[copy(P.sains)]
        E+=[copy(P.exposés)]
        C+=[copy(P.contagieux)]
        I+=[copy(P.infectés)]
        Q+=[copy(P.enQuarantaine)]
        A+=[copy(P.asymptomatiques)]
        R+=[copy(P.rétablis)]
        M+=[copy(P.morts)]
        print(str(100*(t+dt)/duree)+"%")

    return X,Y,S,E,C,I,Q,A,R,M #Résultats de la simulation
```

```
def simulationDansCsv(P,duree,dt,pas,r,σ,ε,γ,λ,α=0,μ=0,χ=0,SDC=1,SFC=1,noms=nomsParDéfaut):
    """Fait la simulation de la population P puis l'enregistre dans des fichiers csv."""
    fichiers=[open(nom,'w',newline="") for nom in noms]
    écriveurs=[csv.writer(f) for f in fichiers]

    confinement=False
    for t in [dt*t for t in range(ceil(duree/dt))]:
        L=[[round(x,2) for x in P.x],[round(y,2) for y in P.y],P.sains,P.exposés,P.contagieux,P.infect
és,P.enQuarantaine,P.asymptomatiques,P.rétablis,P.morts]
        for i in range(10): écriveurs[i].writerow(L[i])
        nInfectés,nVivants=len(P.infectés+P.enQuarantaine),P.n-len(P.morts)
        confinement=nInfectés > SDC*nVivants or (nInfectés > SFC*nVivants and confinement)
        P.propagation(dt,pas,r,σ,ε,γ,λ,α,μ,χ,confinement)
        print(str(100*(t+1)/duree)+"%")
    for f in fichiers: f.close()


def multiSimulationDansCsv(indiceDébut,indiceFin,n,dim,infectés,confinés,duree,dt,pas,r,σ,ε,γ,λ,α=0,μ=
0,χ=0,SDC=1,SFC=1):
    """Fait plusieurs simulations et les enregistre dans fichiers csv indépendants."""
    for indice in range(indiceDébut,indiceFin):
        P=population(n,dim)
        P.infecter(infectés)
        P.confiner(confinés)
        noms=[lettre+str(indice)+'.csv' for lettre in ['x','y','s','e','c','i','q','a','r','m']]
        simulationDansCsv(P,duree,dt,pas,r,σ,ε,γ,λ,α,μ,χ,SDC,SFC,noms)
```

```
def simulationDepuisCsv(noms=nomsParDéfaut):
    """Récupère une simulation enregistrée dans des fichiers csv."""
    fichiers = [open(nom) for nom in noms]
    lecteurs = [csv.reader(f) for f in fichiers]

    X,Y,S,E,C,I,Q,A,R,M=[],[],[],[],[],[],[],[],[],[]
    for i in range(10):
        reader=lecteurs[i]
        L=[X,Y,S,E,C,I,Q,A,R,M][i]
        n_type=[float,float,int,int,int,int,int,int,int,int][i]
        for row in reader:
            L+=[[n_type(n) for n in row]]
        print(str(i+1)+'/10')
    for f in fichiers: f.close()
    return X,Y,S,E,C,I,Q,A,R,M


def simDepuisCsvLight(noms=nomsParDéfaut):
    """Idem mais sans les positions et ne conserve que les longueurs de liste (pas d'animation)."""
    fichiers = [open(nom) for nom in noms]
    lecteurs = [csv.reader(f) for f in fichiers]
    _,_,S,E,C,I,Q,A,R,M=[],[],[],[],[],[],[],[],[],[]
    for i in range(8):
        reader=lecteurs[i]
        L=[S,E,C,I,Q,A,R,M][i]
        for row in reader:
            L+=[len(row)]
    for f in fichiers: f.close()
    return _,_,S,E,C,I,Q,A,R,M
```

```
"""Exploitation des Résultats""«

#Graphs
def graphique(simulation,dt):
    """Affiche la courbe des différents états des individus en fonction du temps."""
    _,_,S,E,C,I,Q,A,R,M=simulation
    duree=len(S)
    tI=[I[t]+Q[t] for t in range(duree)]

    T=[t for t in range(duree)]
    temps=[dt*t for t in T]
    plt.plot(temps,[len(S[t]) for t in T],color="green",label="Sains")
    plt.plot(temps,[len(E[t]) for t in T],color="orange",label="Exposés")
    plt.plot(temps,[len(C[t]) for t in T],color="#ff7b00",label="Contagieux")
    plt.plot(temps,[len(tI[t]) for t in T],color="purple",label="Total infectés")
    plt.plot(temps,[len(I[t]) for t in T],'--',color="purple",label="Infectés")
    plt.plot(temps,[len(A[t]) for t in T],color="red",label="Asymptomatiques")
    plt.plot(temps,[len(R[t]) for t in T],color="blue",label="Rétablis")
    plt.plot(temps,[len(M[t]) for t in T],color="#303030",label="Morts")
    S,E,C,I,Q,A,R,M=[],[],[],[],[],[],[],[]

    plt.title("Évolution de l'épidémie dans le temps")
    plt.xlabel('Temps (en jours)')
    plt.ylabel('Nombre de personnes')
    plt.legend()

    plt.show()
```

```python
def graphiqueLight(sim,dt):
    """Idem mais avec simDepuisCsvLight."""
    _,_,S,E,C,I,Q,A,R,M=sim
    duree=len(S)
    tI=[I[t]+Q[t] for t in range(duree)]
    temps=[dt*t for t in range(duree)]
    plt.plot(temps,S,color="green",label="Sains")
    plt.plot(temps,E,color="orange",label="Exposés")
    plt.plot(temps,C,color="#ff7b00",label="Contagieux")
    plt.plot(temps,tI,color="purple",label="Total infectés")
    plt.plot(temps,I,'--',color="purple",label="Infectés")
    plt.plot(temps,A,color="red",label="Asymptomatiques")
    plt.plot(temps,R,color="blue",label="Rétablis")
    plt.plot(temps,M,color="#303030",label="Morts")
    S,E,C,I,Q,A,R,M=[],[],[],[],[],[],[],[]

    plt.title("Évolution de l'épidémie dans le temps")
    plt.xlabel('Temps (en jours)')
    plt.ylabel('Nombre de personnes')
    plt.legend()

    plt.show()
```

```python
def multiGraphiqueDepuisCsv(indiceDébut,indiceFin,dt):
    """Trace le graphique d'une' moyenne sur plusieurs simulations."""
    lettres=['x','y','s','e','c','i','q','a','r','m']
    moyenneSimulations=[]
    #On récupère la première simulation
    simulation=simulationDepuisCsv([l+str(indiceDébut)+'.csv' for l in lettres])
    moyenneSimulations+=[[len(L) for L in simulation[i]] for i in range(2,10)]]
    duree=len(moyenneSimulations[0])
    temps=[t for t in range(duree)]
    N=indiceFin-indiceDébut
    for indice in range(indiceDébut+1,indiceFin):
        #On fait la somme du nombre de personnes dans chaque état pour chaque simulation ...
        simulation=simulationDepuisCsv([l+str(indice)+'.csv' for l in lettres])
        simulation=[[len(L) for L in simulation[i]] for i in range(2,10)]
        for i in range(len(moyenneSimulations)):
            for t in temps:
                moyenneSimulations[i][t]+=simulation[i][t]
        print(str(100*indice/N)+'%')
    #... pour en faire la moyenne
    for categorie in moyenneSimulations:
        for t in temps:
            categorie[t]/=N
    #Tracé du graphique
    graphiqueLight([None,None]+moyenneSimulations,dt)
```

```python
#Animation
def animer(simulation,taille=0,frequence=1/60,dt=1):
    """Dessine une animation de la propagation à partir d'une simulation existante."""
    X,Y,S,E,C,I,Q,A,R,M=simulation
    #On remet les listes sur une échelle de temps en jours
    for L in [X,Y,S,E,C,I,Q,A,R,M]: L=[L[i] for i in range(int(len(L)*dt))]

    duree=len(X)
    #Total des infectés avec les personnes en quarantaine.
    tI=[I[t]+Q[t] for t in range(duree)]
    if taille==0: taille=max([max(X[0]),max(Y[0])])

    fig,_=plt.subplots()
    fig.set_size_inches(21.6,10.8)
    plt.suptitle("Évolution de l'épidémie dans le temps",fontsize=20)
    plt.subplots_adjust(left=0.05,right=0.95,bottom=0.08)

    #Graphique de gauche : animation du déplacement des individus en fonction du temps
    plt.subplot(1,2,1)
    anim_sains,=plt.plot([],[],'.',color="green",label="Sains")
    anim_exposés,=plt.plot([],[],'.',color="orange",label="Exposés")
    anim_contagieux,=plt.plot([],[],'.',color="#ff7b00",label="Contagieux")
    anim_infectés,=plt.plot([],[],'.',color="purple",label="Infectés")
    anim_asymptomatiques,=plt.plot([],[],'.',color="red",label="Asymptomatiques")
    anim_rétablis,=plt.plot([],[],'.',color="blue",label="Rétablis")
    anim_morts,=plt.plot([],[],'.',color="#303030",label="Morts")

    plt.title('Animation')
    plt.xlim(0,taille)
    plt.ylim(0,taille)
    plt.tick_params(axis='both',which='both’,
left=False,right=False,bottom=False,top=False,labelbottom=False,labelleft=False)
```

```
    #Graphique de droite : animation de la courbe représentant le nombre d'individus dans chaque état
en fonctiopn du temps
    plt.subplot(1,2,2)
    graph_sains,=plt.plot([],[],color="green",label="Sains")
    graph_exposés,=plt.plot([],[],color="orange",label="Exposés")
    graph_contagieux,=plt.plot([],[],color="#ff7b00",label="Contagieux")
    graph_tInfectés,=plt.plot([],[],color="purple",label="Total infectés")
    graph_infectés,=plt.plot([],[],'--',color="purple",label="Infectés")
    graph_asymptomatiques,=plt.plot([],[],color="red",label="Asymptomatiques")
    graph_rétablis,=plt.plot([],[],color="blue",label="Rétablis")
    graph_morts,=plt.plot([],[],color="#303030",label="Morts")

    plt.title('Graphique')
    plt.xlabel('Temps (en jours)')
    plt.ylabel('Nombre de personnes')
    plt.legend() #Affiche la légende
    plt.xlim(0,duree*dt) #Fixe la limite du temps
    plt.ylim(0,len(X[0])) # Fixe le nombre maximal d'individus
```

```
    #Fonction de génération de chaque image
    def gen_frame(f):
        anim_sains.set_data([X[f][p] for p in S[f]],[Y[f][p] for p in S[f]])
        anim_exposés.set_data([X[f][p] for p in E[f]],[Y[f][p] for p in E[f]])
        anim_contagieux.set_data([X[f][p] for p in C[f]],[Y[f][p] for p in C[f]])
        anim_infectés.set_data([X[f][p] for p in I[f]],[Y[f][p] for p in I[f]])
        anim_asymptomatiques.set_data([X[f][p] for p in A[f]],[Y[f][p] for p in A[f]])
        anim_rétablis.set_data([X[f][p] for p in R[f]],[Y[f][p] for p in R[f]])
        anim_morts.set_data([X[f][p] for p in M[f]],[Y[f][p] for p in M[f]])

        T=[t for t in range(f)]
        temps=[t*dt for t in T] #On crée la liste des tous les instants jusqu'à f de la simulation
        graph_sains.set_data(temps,[len(S[t]) for t in T])
        graph_exposés.set_data(temps,[len(E[t]) for t in T])
        graph_contagieux.set_data(temps,[len(C[t]) for t in T])
        graph_tInfectés.set_data(temps,[len(tI[t]) for t in T])
        graph_infectés.set_data(temps,[len(I[t]) for t in T])
        graph_asymptomatiques.set_data(temps,[len(A[t]) for t in T])
        graph_rétablis.set_data(temps,[len(R[t]) for t in T])
        graph_morts.set_data(temps,[len(M[t]) for t in T])
        return anim_sains,anim_exposés,anim_contagieux,anim_infectés,anim_asymptomatiques,anim_rétabli
s,anim_morts,graph_sains,graph_exposés,graph_contagieux,graph_infectés,graph_asymptomatiques,graph_rét
ablis,graph_morts

    ani=animation.FuncAnimation(fig=fig, func=gen_frame, frames=range(duree), interval=frequence*1000,
 blit=True)
    # ani.save('simvid.mp4',writer='ffmpeg') --> possibilité de l'enregistrer
    plt.show()
```

```
"""Aide au Calcul de densité de population"""
def densité(n,dim,r): return (pi*r**2*n)/dim**2

def dim(n,r,densité): return sqrt(pi*r**2*n/densité)




"""Exemple d'utilisation du programme"""
r=dim(1000,1,1) #r=56
P=population(1000,r)
P.infecter(5)
S=simulation(P,150,1/10,5,1,3/8,1/3,1/2,1/8,0.25,0.02/8)
animer(S,56,1/120,1/10)
```

```python
"""Modèle Mathématique"""

import matplotlib.pyplot as plt

class population(object):
    def __init__(self,n=100,infectés=1,immunisés=0):
        """Crée une population de n individus"""
        self.n = n #Nombre d'individus...
        self.sains = n-immunisés-infectés #... sains
        self.exposés = 0 #... contaminés mais pas contagieux (et sans symptômes)
        self.contagieux = 0 #... contagieux mais pas encore de symptômes apparents
        self.infectés = infectés #... infectés
        self.asymptomatiques = 0 #... infectés mais sans symptôme
        self.enQuarantaine = 0 #infectés mais indentifié et isolé
        self.rétablis = immunisés #... rétablis ou déjà immunisés sans y avoir été exposés
        self.morts = 0 #... décédés


    def propagation_SIR(self,β,γ):
        """Propage le virus à l'instant selon le modèle SIR"""
        dS=int(self.sains*β*self.infectés)
        dI=int(self.infectés*γ)
        self.sains += -dS
        self.infectés += dS-dI
        self.rétablis += dI
```

```python
def propagation_SEIR(self,σ,ε,λ,μ):
    """Propage le virus à l'instant selon le modèle SEIR"""
    n=self.n
    dS=σ/n*self.sains*self.infectés
    dE=ε*self.exposés
    dR=λ*self.infectés
    dM=μ*self.infectés
    self.sains += -dS
    self.exposés += dS-dE
    self.infectés += dE-dR-dM
    self.rétablis += dR
    self.morts += dM


def propagation_complete(self,dt,σ,ε,γ,λ,α,μ,χ=0):
    """Propage le virus à l'instant selon le modèle complet"""
    n=self.n
    dS=σ/n*self.sains*(self.infectés+self.contagieux+self.asymptomatiques)*dt
    dE=ε*self.exposés*dt
    dC=γ*self.contagieux*dt
    dQ=χ*self.infectés*dt
    dRi=λ*self.infectés*dt
    dRq=λ*self.enQuarantaine*dt
    dRa=λ*self.asymptomatiques*dt
    dMi=μ*self.infectés*dt
    dMq=μ*self.enQuarantaine*dt
    self.sains += -dS
    self.exposés += dS-dE
    self.contagieux += dE-dC
    self.infectés += (1-α)*dC-dRi-dMi-dQ
    self.asymptomatiques += α*dC-dRa
    self.enQuarantaine += dQ-dRq-dMq
    self.rétablis += dRi+dRa+dRq
    self.morts += dMi+dMq
```

```python
"""Simulations"""

def simulation_SIR(P,duree,dt,β,γ):
    """Fait la simulation de l'épidémie avec P selon le modèle SIR"""
    S,I,R=[P.sains],[P.infectés],[P.rétablis]
    for _ in range(int(duree/dt)-1):
        P.propagation_SIR(β,γ)
        S+=[P.sains]
        I+=[P.infectés]
        R+=[P.rétablis]
    return [S,I,R]


def simulation_SEIR(P,duree,dt,σ,ε,λ,μ):
    """Fait la simulation de l'épidémie avec P selon avec le modèle SEIR"""
    S,E,I,R=[P.sains],[P.exposés],[P.infectés],[P.rétablis]
    for _ in range(int(duree/dt)-1):
        P.propagation_SEIR(σ,ε,λ,μ)
        S+=[P.sains]
        E+=[P.exposés]
        I+=[P.infectés]
        R+=[P.rétablis]
    return [S,E,I,R]
```

```
def simulation_complete(P,duree,dt,σ,ε,γ,λ,α,μ,χ,SDC=1,PPC=0,SFC=1):
    """Fait la simulation de l'épidémie avec P selon avec le modèle complet"""
    S,E,C,I,A,Q,R,M=[P.sains],[P.exposés],[P.contagieux],[P.infectés],[P.asymptomatiques],[P.enQuarant
aine],[P.rétablis],[P.morts]
    conf=False
    temps=[t for t in range(1,int(duree/dt))]
    for _ in temps:
        vivants=P.n-P.morts
        if P.infectés+P.enQuarantaine > SDC*vivants or (P.infectés+P.enQuarantaine > SFC*vivants and c
onf):
            P.propagation_complete(dt,σ*(1-PPC)**2,ε,γ,λ,α,μ,χ)
            conf=True
        else:
            P.propagation_complete(dt,σ,ε,γ,λ,α,μ,χ)
            conf=False
        S+=[P.sains]
        E+=[P.exposés]
        C+=[P.contagieux]
        I+=[P.infectés]
        A+=[P.asymptomatiques]
        Q+=[P.enQuarantaine]
        R+=[P.rétablis]
        M+=[P.morts]
    return [S,E,C,I,A,Q,R,M]
```

```
"""Graphique"""

def graphique_complet(simulation,mod,dt):
    """Affiche l'évolution de l'épidémie en fonction du temps"""
    if mod=="SIR": S,I,R=simulation
    elif mod=="SEIR": S,E,I,R=simulation
    elif mod=="SEIR+": S,E,C,I,A,Q,R,M=simulation
    temps=[t*dt for t in range(len(S))]
    totalInfectés=[I[t]+Q[t] for t in range(len(S))]

    plt.plot(temps,S,color="green",label="Sains")
    if mod=="SEIR+":
        plt.plot(temps,E,color="orange",label="Exposés")
        plt.plot(temps,C,color="#ff7b00",label="Contagieux")
        plt.plot(temps,totalInfectés,color="purple",label="Total infectés")
        plt.plot(temps,I,'--',color="purple",label="Infectés")
        plt.plot(temps,A,color='red',label="Asymptomatiques")
        plt.plot(temps,R,color="blue",label="Rétablis")
        plt.plot(temps,M,color="black",label="Morts")
    else:
        if mod=="SEIR": plt.plot(temps,E,color="orange",label="Exposés")
        plt.plot(temps,I,color="purple",label="Infectés")
        plt.plot(temps,R,color="blue",label="Rétablis")

    plt.title("Évolution de l'épidémie dans le temps")
    plt.xlabel('Temps (en jours)')
    plt.ylabel('Nombre de personnes')
    plt.legend()

    plt.show()
```

```
"""Exemple d'utilisation du code"""
P=population(1000,infectés=5)
S=simulation_complete(P,150,1/10,3/8,1/3,1/2,1/8,0.25,0.02/8)
graphique_complet(S,"SEIR+",1/10)
```