Projet de deuxième année

Simulation de modèles d'épidémie

mai 2020

CROGUENNEC Guillaume

DUPONT Ronan

Table des matières

1 Introduction

Lors d'une épidémie ou d'une pandémie, le meilleur moyen d'y faire face est de savoir comment elle va évoluer et comment elle va réagir à chacune des actions que l'on fera à son encontre. Pour savoir tout cela, le meilleur moyen est de la modéliser et de la simuler à l'aide de modèles mathématiques. C'est dans les années 1920 que l'approche des modèles compartimentaux est pensée. C'est une approche selon laquelle la population est divisée en classes selon leur état par rapport au virus. L'approche basique pour une épidémie est de diviser la population en trois classes : les sains, les infectés et les rétablis. Mais elle peut être améliorée, en incluant par exemple d'autres classes. Ce modèle peut également être modifié pour qu'il corresponde à certains types de virus particuliers, comme le virus des zombies, qui est dans l'imaginaire collectif la pire épidémie qui pourrait arriver. C'est pour cela que l'on s'y intéresse. Le but est donc de modéliser et de simuler une épidémie de zombies, mais pour cela il est nécessaire de passer par la modélisation et la simulation d'une épidémie habituelle.

2 Modélisation d'une épidémie

2.1 Modèle SIR

La base de cette modélisation d'épidémie est de diviser la population en trois classes décrites par des fonctions dont les variables sont les coordonnées x et y dans l'espace de notre modélisation et le temps t. On pose donc les fonctions S, I et R qui ont respectivement pour valeurs les nombres d'individus sains, infectés et rétablis aux coordonnées (x ;y) au moment t.
Avant la modélisation, ce qui est su de l'épidémie c'est les valeurs de S, I et R à ce que l'ont appelle l'instant initial. Et le but de la modélisation est de prédire quelles seront ces valeurs après une certaine durée. Pour cela, il faut avoir un schéma qui permette de déduire des valeurs de S, I et R, leur variation à cet instant.
Pour cela, on part du postulat que, premièrement, la variation temporelle d'individus sains à un endroit ne dépend que du nombre d'individus qui ont été infectés depuis l'instant d'avant. Deuxièmement, que la variation temporelle des individus infectés à un endroit ne dépend que du nombre d'individus qui ont été infectés depuis l'instant d'avant et du nombre d'individus qui ont guéri depuis l'instant d'avant. Et troisièmement, que la variation temporelle d'individus rétablis à un endroit ne dépend que du nombre d'individus qui ont guéri depuis l'instant d'avant..

Ce modèle compartimental est schématisé de la manière suivante :

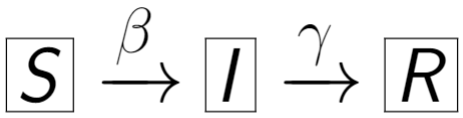


FIGURE 1 – SIR classique

Les paramètres de cette modélisation d'une épidémie sont donc, premièrement, le taux d'infection β du virus, qui est multiplié à un terme S.I dans les équations car le nombre d'infections dépend aussi du nombre de sains et du nombre d'infectés. Et deuxièmement, le taux de guérison γ du virus, qui est multiplié à un terme I dans les équations car le nombre de guérisons dépend aussi du nombre d'infectés.
En prenant en compte premièrement que les infections diminuent le nombre de sains et augmentent le nombre nombre d'infectés. Et deuxièmement que les guérisons diminuent le nombre d'infectés

et augmentent le nombre de rétablis. On obtient le système d'équations suivant :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta SI \\ \dfrac{\partial I}{\partial t} = \beta SI - \gamma I \\ \dfrac{\partial R}{\partial t} = \gamma I \end{cases}$$

On précise que les coefficients dans tous les modèles sont des taux pas jour (sauf β qui est est un taux par jour par personne, et plus loin le taux de diffusion qui est un taux par jour par pas spatial au carré). Donc Les variations calculées sont les variations en un jour (pour un pas temporel unitaire).

2.2 Modèle SEIR

Même si les résultats du modèle SIR présenté ci-dessus se rapprochent de la réalité, ils sont loin d'être similaires. Cela est dû au fait que le modèle est un peu simpliste. En réalité il faudrait sûrement autant de classes de population et de paramètres qu'il y a de personnes pour le rendre exact. Mais en ajoutant seulement une classe de population et trois paramètres, il est possible d'obtenir des résultats déjà beaucoup plus proches de la réalité.
La première chose qui n'a pas encore été prise en compte dans ce modèle SIR est le fait que lorsqu'un individu est infecté par un virus, il n'est pas obligatoirement infectueux. Il faut donc ajouter à notre modèle une classe de population pour les individus infectés non infectueux. On appelle E la fonction qui représente cette classe de population dans notre modèle. Il est schématisé de la manière suivante :

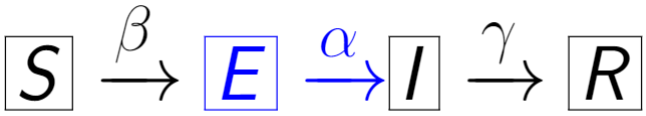


FIGURE 2 – SEIR classique

Dans ce nouvelle modèle, on considère que tous les individus passent par l'étape infectés non infectueux, et qu'ensuite ils deviennent infectueux. Donc les nouveaux infectés ne vont pas s'ajouter à I, mais à E. Et ensuite un certain nombre des individus (suivant le taux d'incubation α) de cette classe va devenir infectueux, et va donc se soustraire à E et s'additionner à I. Ce nouveau terme qui apparaît deux fois dans le système d'équation est donc le taux d'incubation multiplié à E.
Le nouveau système d'équation du modèle est :

$$\begin{cases} \dfrac{dS(t)}{dt} = -\beta S(t)I(t) \\ \dfrac{dE(t)}{dt} = \beta S(t)I(t) - \alpha E(t) \\ \dfrac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \\ \dfrac{dR(t)}{dt} = \gamma I(t) \end{cases}$$

Le deuxième élément qui n'est pas encore pris en compte dans ce modèle est la natalité. En effet, même lors d'une épidémie il y a toujours des naissances.
En considérant que les bébés naissent sains, il suffit d'ajouter à la variation de S un terme

constitué du taux de natalité ν multiplié à la population totale N (valant l'intégrale sur le domaine du problème de S+E+I+R).

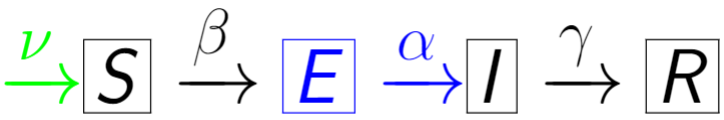


FIGURE 3 – SEIR natalité

On obtient donc le système d'équations suivant :

$$\begin{cases} \frac{dS(t)}{dt} = -\beta S(t)I(t) + \nu N(t) \\ \frac{dE(t)}{dt} = \beta S(t)I(t) - \alpha E(t) \\ \frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \\ \frac{dR(t)}{dt} = \gamma I(t) \end{cases}$$

Enfin, un troisième élément qui n'est pas encore pris en compte dans ce modèle est la mortalité naturelle. Pour l'ajouter à ce modèle il suffit de soustraire à chaque variation de classe le taux de mortalité μ, multiplié au nombre d'individus dans la classe.

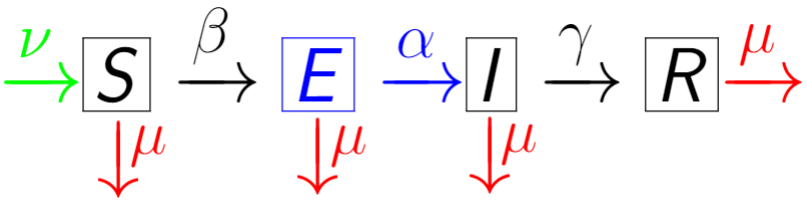


FIGURE 4 – SEIR natalité et mortalité

On obtient donc le système d'équations suivant :

$$\begin{cases} \frac{dS(t)}{dt} = -\beta S(t)I(t) + \nu N(t) - \mu S(t) \\ \frac{dE(t)}{dt} = \beta S(t)I(t) - \alpha E(t) - \mu E(t) \\ \frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) - \mu I(t) \\ \frac{dR(t)}{dt} = \gamma I(t) - \mu R(t) \end{cases}$$

Cette amélioration du modèle SIR s'appelle modèle SEIR. En passant de trois classes de population et deux paramètres à quatre classes de population et cinq paramètres, le modèle parvient déjà à des résultats beaucoup plus proches de la réalité.
On peut voir la différence en comparant les résultats obtenus pour les deux modèles.
Bien sûr, ce modèle SEIR peut encore être améliorer. Par exemple en considérant que lorsqu'un bébé né, il est dans la même classe de population que sa mère.
Dans ce cas, il ne faudrait pas ajouter ν N à la variation de S, mais ajouter à chaque variation de classe ν multiplié au nombre d'individus dans cette classe. Un autre exemple d'amélioration

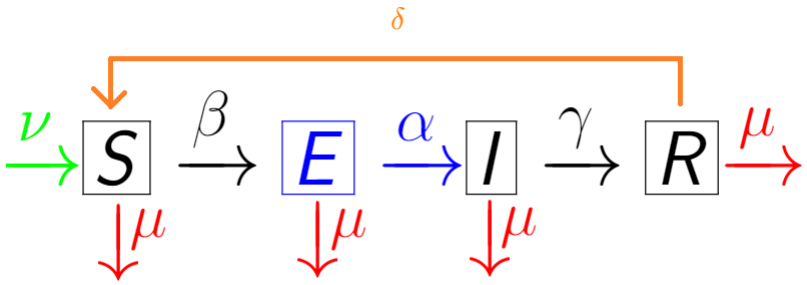


FIGURE 5 – SEIR : Perte d'immunité

serait de prendre en compte le fait parfois, certains rétablis ne sont plus immunisés contre le virus.
Il faudrait donc soustraire ces individus à la classe des rétablis, et les ajouter à la classe des sains (qui représente en fait les individus non infectés mais qui peuvent le devenir).
Avec de telles modifications, le système d'équations du modèle SEIR deviendrait :

$$\begin{cases} \frac{dS(t)}{dt} = -\beta S(t)I(t) + \nu N(t) - \mu S(t) + \delta R(t) \\ \frac{dE(t)}{dt} = \beta S(t)I(t) - \alpha E(t) - \mu E(t) \\ \frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) - \mu I(t) \\ \frac{dR(t)}{dt} = \gamma I(t) - \mu R(t) - \delta R(t) \end{cases}$$

Enfin, une autre modification du modèle serait de prendre en compte l'âge des individus dans la population. En effet le virus n'agit pas de la même manière suivant l'âge de l'individu (en raison de la condition physionomique). Et c'est quelque chose d'important à prendre en compte car les pourcentages des individus d'un certain âge ne sont pas les mêmes suivant les endroits.
Pour prendre en compte cela, ce n'est pas un nouveau paramètre ou une nouvelle classe de population qu'il faut ajouter, mais une nouvelle variable. On passe donc de trois variables (t, x et y) à quatre variables (t, x, y et a qui est l'âge des individus). Ce qui change dans les équations c'est que chaque fonction dépend donc de quatre variables, et que dans chaque équation on additionne au terme de la dérivée temporelle, du même côté de l'égalité, une dérivée par rapport à l'âge.
On a donc le système d'équations suivant :

$$\begin{cases} \frac{\partial S(a,t)}{\partial t} + \frac{\partial S(a,t)}{\partial a} = -\lambda(a,t)S(a,t) - \mu(a)S(a,t) \\ \frac{\partial E(a,t)}{\partial t} + \frac{\partial E(a,t)}{\partial a} = \lambda(a,t)S(a,t) - \alpha(a)E(a,t) - \mu(a)E(a,t) \\ \frac{\partial I(a,t)}{\partial t} + \frac{\partial I(a,t)}{\partial a} = \alpha(a)E(a,t) - \gamma(a)I(a,t) - \mu(a)I(a,t) \\ \frac{\partial R(a,t)}{\partial t} + \frac{\partial R(a,t)}{\partial a} = \gamma(a)I(a,t) - \mu(a)R(a,t) \end{cases}$$

Avec donc $\alpha(a), \gamma(a)$ et $\mu(a)$ respectivement les taux d'incubation, de guérison et de mortalité qui dépendent tous de l'âge.
Et $\lambda(a,t)$ le taux d'infection qui est défini par :

$$\lambda(a,t) = \int_0^{a_{\max}} \beta(u,a)I(u,t)du$$

Où a_{max} est l'âge maximal dans la population et β le taux d'infection qui dépend d'un âge donné. On peut par exemple prendre une gaussienne de la forme suivante :

$$\beta(u,a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(u-a)^2}{2\sigma^2}}$$

et σ dépend du virus donné.

2.3 Modèle SZR

Pour ce qui est de l'épidémie de zombies, les modèles diffèrent un peu car c'est un virus qui est pour l'instant de la science fiction, et qui est donc le plus "fort" possible. Il faut donc utiliser un modèle analogue au modèle SIR, car un modèle analogue au modèle SEIR serait plus compliqué pour le même résultat. En effet, dans la littérature de science-fiction la période d'incubation du virus est négligeable, souvent de maximum quelques heures, voire moins. Donc ajouter une classe d'infectés non infectueux et un paramètre de taux d'incubation est inutile. De plus on sait d'avance que c'est un virus qui se propage extrêmement vite, et donc que l'on peut négliger les naissances et les morts naturelles sur le court laps de temps qui va nous intéresser pour la simulation.
On se retrouve donc à essayer de trouver un modèle analogue au modèle SIR, mais qui diffère néanmoins en certains point. Déjà, ce nouveau modèle va être appelé SZR, car on ne parle d'infectés, mais de zombies. La différence est qu'un zombi ne peut pas guérir du virus. Par conséquent, R ne représente plus la classe de population des individus rétablis, mais des zombies qui ont été tués par les humains sains qui tentent de se défendre.
En soit, le modèle SZR ne diffère qu'en un point du modèle SIR, outre la lettre Z à la place de la lettre I. Cette différence est due au fait que R ne représente pas les rétablis mais les zombies mort, car cela implique que dans le système d'équation, le terme de guérison qui apparaît deux fois doit être remplacé par un terme d'élimination des zombies. Le coefficient n'est donc plus le même, et il n'est plus multiplié seulement à Z, mais à S.Z. Car le nombre d'élimination de zombies dépend à la fois du nombre de zombies et du nombre d'humains sains qui se défendent.

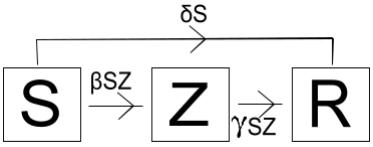


FIGURE 6 – SZR classique

Le système d'équations obtenu pour le modèle SZR est donc le suivant :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta SZ \\ \dfrac{\partial Z}{\partial t} = \beta SZ - \gamma SZ \\ \dfrac{\partial R}{\partial t} = \gamma SZ \end{cases}$$

On peut ensuite compléter ce modèle en ajoutant un taux de résurrection ϵ. L'idée est que même un corps inanimé peut être infecté par le virus. On soustrait donc un ϵR au terme en R et on l'ajoute aux zombies :

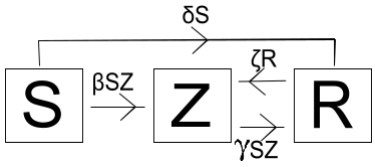


FIGURE 7 – SZR classique avec taux de résurrection ϵ

Le système d'équations obtenu pour le modèle SZR avec taux de résurrection est donc le suivant :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta SZ \\ \dfrac{\partial Z}{\partial t} = \beta SZ - \gamma SZ + \epsilon R \\ \dfrac{\partial R}{\partial t} = \gamma SZ - \epsilon R \end{cases}$$

2.4 Ajout d'un terme de diffusion α_{diff}

Dans certains modèles SIR (et donc égalements dans certains modèles SZR) les termes qui modélisent les déplacements de la population ne sont pas présents. C'est-à-dire que les déplacements au sein de la population sont négligés. Or, les déplacements de la population ne sont plus négligeables de nos jours. Pour prendre en compte les déplacements de chaque classe de population, il faut ajouter à chaque variation de classe le taux de déplacement α_{diff} multiplié au laplacien du la fonction de cette classe. On peut par contre négliger les déplacements des individus rétablis, car dans un modèle SIR classique ces derniers n'influent plus sur la propagation du virus. Par exemple le modèle SIR deviendrait :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta SI + \alpha_{diff}\Delta S \\ \dfrac{\partial I}{\partial t} = \beta SI - \gamma I + \alpha_{diff}\Delta I \\ \dfrac{\partial R}{\partial t} = \gamma I \end{cases}$$

Voici une simulation avec et sans déplacements de population.

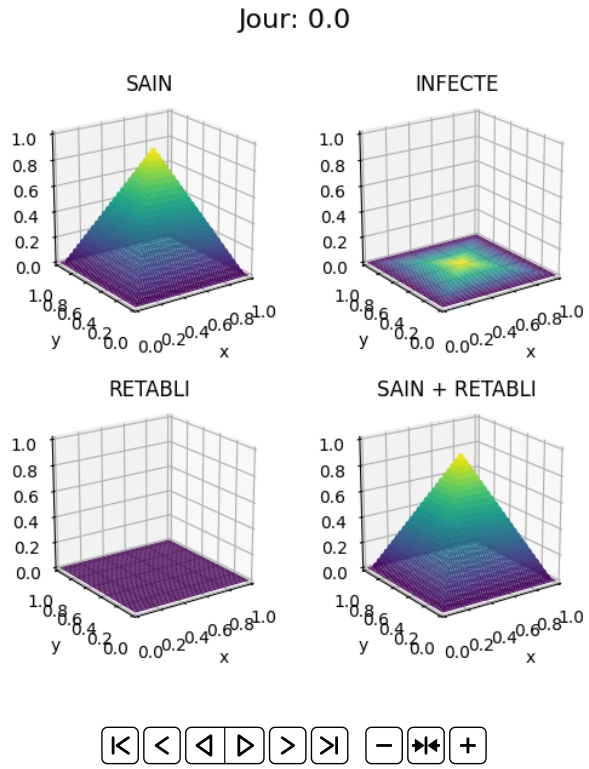


FIGURE 8 – $\alpha_{diff} = 0$
https://drive.google.com/open?id=1fHo_6fZXYxcS6o_sCQ7gGbo5mL60Q_dG

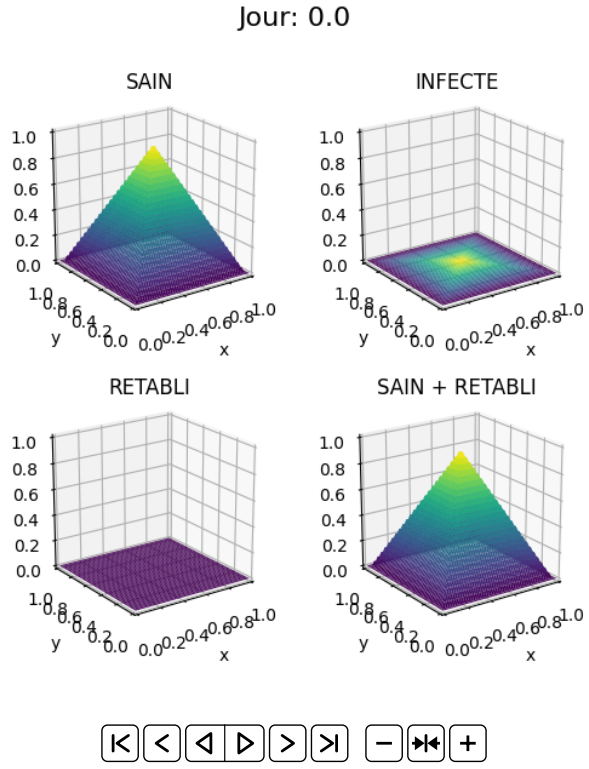


FIGURE 9 – $\alpha_{diff} = 0.001$
https://drive.google.com/open?id=1oRpNrCPQdi76do92a51eSTpbEm9kNWci

En comparant les deux résultats des classes de population infectés, on remarque bien l'influence du facteur de diffusion : pour la figure sans facteur de diffusion on a un pic plus haut et moins étalé, alors qu'en ajoutant un facteur de diffusion ce pic s'arrondi est s'étale plus : les populations se mélangent et s'homégénisent dans la zone entière.
Ce facteur est donc très important car dans la réalité les populations sont en mouvement. Même si cela soulève également un problème. C'est que ce terme représente le taux de déplacement moyen, alors qu'en réalité ce taux devrait être variable sur la topologie du problème. Mais c'est un problème qui sera étudié plus tard.

Néanmoins, dans les deux cas, diffusion ou non, nous observons que le pic de la maladie se situe entre 6.5 et 7 jours. Donc à peu près au même moment. Mais cela ne signifie pas qu'un confinement sans déplacement ne ralenti pas la propagation du virus. C'est en fait dû au fait que le maillage spatial reste toujours grossier pas rapport à la réalité. Donc même si on empêche les déplacements entre les cases, ce modèle n'empêche pas les déplacements à l'intérieur d'une case.

3 Résolution Numérique des modèles

Une fois les modèles numériques obtenus, il faut résoudre les systèmes d'équations afin de pouvoir simuler le phénomène de l'épidémie. On ne résout que les modèles SIR et SZR, car on est intéressé par l'épidémie de zombies et sa comparaison avec une épidémie habituelle. Le modèle SEIR qui n'est pas analogue au modèle SZR est donc caduque dans cette étude.
Les équations des modèles étudiés n'étant pas linéaires, il serait très compliqué de trouver des solutions exactes aux systèmes. C'est donc pour cela qu'une résolution numérique est nécessaire. Dans ce but, on commence par résoudre ces modèles sur une topologie simple. On choisit que l'épidémie se propage dans une zone en une dimension, dont aucun individu ne peut sortir, et dont la population est répartie simplement.

3.1 Méthode explicite

3.1.1 Obtention de la formule itérative

Afin de trouver la bonne méthode de résolution numérique, il est nécessaire de commencer par un seul cas simple. Pour cela, on choisit le modèle SIR, sur une zone en une dimension, allant de 0 à 1.
La méthode la plus simple est celle des différences finies, en utilisant l'approximation centrée sauf pour les dérivées temporelles. Pour cela, il faut discrétiser la zone en subdivisions de taille $h = 1/M$ ainsi que la temporalité en subdivisions de taille Δt.
On note donc : $S(x_i, t_n) = S_i^n = S(i.h, n.dt)$ et $I(x_i, t_n) = I_i^n = I(i.h, n.dt)$. En utilisant le développement de Taylor au premier et second ordre, on se ramène facilement à :

$$\frac{\partial S}{\partial t}(x_i, t_n) \approx \frac{S_i^{n+1} - S_i^n}{dt} \qquad \frac{\partial^2 S}{\partial^2 x}(x_i, t_n) \approx \frac{S_{i+1}^n - 2S_i^n + S_{i-1}^n}{h^2}$$

On peut donc ramener la première équation du sytème à l'équation suivante :

$$\frac{S_i^{n+1} - S_i^n}{dt} = -\beta S_i^n I_i^n + \alpha_{diff} \frac{S_{i+1}^n - 2S_i^n + S_{i-1}^n}{h^2}$$

$$\Rightarrow \; S_i^{n+1} = S_i^n + dt(-\beta S_i^n I_i^n + \frac{\alpha_{diff}}{h^2}(S_{i+1}^n - 2S_i^n + S_{i-1}^n)) \tag{1}$$

De même on peut ramener la deuxième équation du système à l'équation suivante :

$$\Rightarrow \; I_i^{n+1} = I_i^n + dt(-\gamma I_i^n + \beta S_i^n I_i^n + \frac{\alpha_{diff}}{h^2}(I_{i+1}^n - 2I_i^n + I_{i-1}^n)) \tag{2}$$

Enfin, on peut ramener la troisième équation du système à cette équation :

$$\Rightarrow \; R_i^{n+1} = R_i^n + dt.\gamma.I_i^n \tag{3}$$

Avec ces trois nouvelles équations, on peut grâce à de simples itérations obtenir toutes les valeurs de S, I et R à chaque instant.

3.1.2 Conditions limites

Tout d'abord, pour que la modélisation soit fidèle au cas étudié, il faut imposer que personne ne sorte de la zone en imposant :

$$\frac{\partial S}{\partial x}(0, t) = \frac{\partial S}{\partial x}(1, t) = 0$$

$$\frac{\partial I}{\partial x}(0, t) = \frac{\partial I}{\partial x}(1, t) = 0$$

$$\frac{\partial R}{\partial x}(0,t) = \frac{\partial R}{\partial x}(1,t) = 0$$

Une solution simple pour imposer cela est d'utiliser encore une fois la méthode des différences finies centrée. Cela revient à imposer pour n'importe quel n :

$$S_{-1}^n = S_1^n$$
$$S_{M-1}^n = S_{M+1}^n$$
$$I_{-1}^n = I_1^n$$
$$I_{M-1}^n = I_{M+1}^n$$
$$R_{-1}^n = R_1^n$$
$$R_{M-1}^n = R_{M+1}^n$$

En faisant plusieurs simulations, on s'aperçoit que même si la méthode explicite est plus simple, elle n'est pas préférable car pour avoir un maillage spatial assez fin il faut avoir un maillage temporel encore plus fin. En effet, dans cette méthode explicite, la condition de stabilité CFL : $2\Delta t \leq (h)^2$ doit être respectée, sinon la solution du système ne convergera pas. Cela oblige donc à faire beaucoup de calculs, car le pas temporel est très petit, et cela prend donc beaucoup trop de temps.

3.2 Méthode implicite

3.2.1 Modèle SIR 1D

3.2.1.1 Obtention des nouvelles équations

Une seconde approche pour résoudre ce problème est d'utiliser la méthode implicite. Elle est moins facile à mettre en place, mais n'impose aucune condition de stabilité.
Encore une fois, on essaie donc dans un premier temps de résoudre la modèle SIR dans une zone en une dimension. Et pour cela, on utilise encore une fois la méthode des différences finies en discrétisant le problème de la même manière.
Nous posons donc encore une fois : $S(x_i, t_n) = S_i^n$. Puis on utilise encore une fois le développement de Taylor au premier et deuxième ordre, mais cette fois en utilisant la fonction à l'instant d'après, sauf dans la dérivée temporelle qui ne change pas :

$$\frac{\partial S}{\partial t}(x_i, t_n) \approx \frac{S_i^{n+1} - S_i^n}{dt} \qquad \frac{\partial^2 S}{\partial^2 x}(x_i, t_{n+1}) \approx \frac{S_{i+1}^{n+1} - 2S_i^{n+1} + S_{i-1}^{n+1}}{h^2}$$

On peut donc ramener le problème à l'équation suivant :

$$\frac{S_i^{n+1} - S_i^n}{dt} = -\beta S_i^{n+1} I_i^n + \alpha_{diff} \frac{S_{i+1}^{n+1} - 2S_i^{n+1} + S_{i-1}^{n+1}}{h^2}$$

Donc :

$$S_i^{n+1}(1 + dt(\beta I_i^n + 2\frac{\alpha_{diff}}{h^2})) - \alpha_{diff}\frac{dt}{h^2}(S_{i+1}^{n+1} + S_{i-1}^{n+1}) = S_i^n \tag{4}$$

De même avec I :

$$I_i^{n+1}(1 + dt(\gamma - \beta S_i^n + 2\frac{\alpha_{diff}}{h^2})) - \alpha_{diff}\frac{dt}{h^2}(I_{i+1}^{n+1} + I_{i-1}^{n+1}) = I_i^n \tag{5}$$

La troisième équation du système n'est pas changée.
On transforme ensuite ces systèmes d'équations (pour chaque i) en système d'équations matricielles.

On pose :

$$\begin{cases} a_{Si} = 1 + dt(\beta I_i^n + 2\dfrac{\alpha_{diff}}{h^2}) \\ b_i = -\alpha_{diff}\dfrac{dt}{h^2} \end{cases}$$

Et on obtient comme première équation du système : $A_S x_S = b_S$ avec :

$$A_S = \begin{pmatrix} a_{S_0} & b_0 & & & (0) \\ b_0 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ (0) & & & b_{n-1} & a_{S_{n-1}} \end{pmatrix} b_S = \begin{pmatrix} S_0^n \\ \vdots \\ \vdots \\ \vdots \\ S_{n-1}^n \end{pmatrix} x_S = \begin{pmatrix} S_0^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ S_{n-1}^{n+1} \end{pmatrix}$$

Puis on pose :

$$\begin{cases} a_{Ii} = 1 + dt(\gamma - \beta S_i^n + 2\dfrac{\alpha_{diff}}{h^2}) \\ b_i = -\alpha_{diff}\dfrac{dt}{h^2} \end{cases}$$

Et on obtient comme deuxième équation du système : $A_I x_I = b_I$ avec :

$$A_I = \begin{pmatrix} a_{I_0} & b_0 & & & (0) \\ b_0 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ (0) & & & b_{n-1} & a_{I_{n-1}} \end{pmatrix} b_I = \begin{pmatrix} I_0^n \\ \vdots \\ \vdots \\ \vdots \\ I_{n-1}^n \end{pmatrix} x_I = \begin{pmatrix} I_0^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ I_{n-1}^{n+1} \end{pmatrix}$$

La troisième équation du système est donc l'équation matricielle telle que pour chaque i on a :

$$\Rightarrow \; R_i^{n+1} = R_i^n + dt.\gamma.I_i^n \tag{6}$$

3.2.1.2 Conditions limites

Pour respecter le fait qu'aucun individu ne sorte de la zone, on pourrait à chaque résolution du système égaliser le premier terme avec le second ainsi que le dernier avec l'avant dernier. Mais cette méthode n'est pas très précise, elle aplatie sur les bords la courbe de la fonction.

L'autre méthode est d'utiliser d'imposer la dérivée spatiale nulle sur tous les bords grâce à la formule centrée des différences finies. Les deux matrices A deviennent donc :

$$A_S = \begin{pmatrix} a_0 & 2b_0 & & & (0) \\ b_0 & \ddots & b_1 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & \ddots & \ddots \\ (0) & & & 2b_{n-1} & a_{n-1} \end{pmatrix}$$

En faisant plusieurs simulations, on conclue que cette méthode est bien plus pratique que la méthode précédente, et qu'elle est suffisante pour cette étude.

3.2.2 Modèle SIR 2D

3.2.2.1 Obtention des équations matricielles

Maintenant que la bonne méthode a été choisie, il faut l'appliquer à une zone en deux dimensions. Ce qui correspond à la réalité car les populations humaines se déplacent suivant deux dimensions.
Ce qui change par rapport à une seule dimension spatiale est la manière dont on approxime le laplacien :

$$\Delta S_{i,j}^n = \frac{S_{i+1,j}^n + S_{i-1,j}^n - 4S_{i,j}^n + S_{i,j+1}^n + S_{i,j-1}^n}{h^2}$$

On arrive aux équations suivantes :

$$S_{i,j}^{n+1}(1 + dt(\beta I_{i,j}^n + 4\frac{\alpha_{diff}}{h^2})) - \alpha_{diff}\frac{dt}{h^2}(S_{i+1,j}^{n+1} + S_{i-1,j}^{n+1} + S_{i,j+1}^{n+1} + S_{i,j-1}^{n+1}) = S_{i,j}^n \tag{7}$$

Et de même avec I :

$$I_{i,j}^{n+1}(1 + dt(\gamma - \beta S_{i,j}^n + 4\frac{\alpha_{diff}}{h^2})) - \alpha_{diff}\frac{dt}{h^2}(I_{i+1,j}^{n+1} + I_{i-1,j}^{n+1} + I_{i,j+1}^{n+1} + I_{i,j-1}^{n+1}) = I_{i,j}^n \tag{8}$$

Ensuite, il faut faire en sorte d'obtenir un système d'équations matricielles.

Pour cela,on doit poser le problème en deux dimensions comme un problème en une dimension. On doit donc numéroter chaque éléments du maillage de 0 à $M^2 - 1$.
On choisit la numérotation suivante : on partcourt la grille de gauche à droite puis de bas en haut.
On en déduit donc que l'élément i,j de la maille correspond au i.M+j ème élément du vecteur b. Puis on crée donc le vecteur b et la matrice A en fonction de cette notation.

En posant

$$\begin{cases} a_{Ii} = 1 + dt(-\beta S_i^n + 4\frac{\alpha_{diff}}{h^2}) \\ b_i = -\alpha_{diff}\frac{dt}{h^2} \end{cases}$$

On obtient donc comme première équations matricielle du sustème : $A_S x_S = b_S$ avec :

$$A_S = \begin{pmatrix} a_{S_0} & b_i & & & b_i & & (0) \\ b_i & \ddots & \ddots & & & \ddots & \\ & \ddots & \ddots & \ddots & & & b_i \\ & & \ddots & \ddots & \ddots & & \\ b_i & & & \ddots & \ddots & \ddots & \\ & \ddots & & & \ddots & \ddots & b_i \\ (0) & & b_i & & & b_i & a_{S_{n^2-1}} \end{pmatrix}$$

On notera qu'il y a cette fois des M-diagonales pour la dérivée seconde selon la deuxième dimension spatiale.

En posant

$$\begin{cases} a_{Ii} = 1 + dt(\gamma - \beta S_i^n + 4\frac{\alpha_{diff}}{h^2}) \\ b_i = -\alpha_{diff}\frac{dt}{h^2} \end{cases}$$

On obtient comme deuxième équation matricielle du système : $A_I x_S = b_I$ avec :

$$A_I = \begin{pmatrix} a_{I_0} & b_i & & & b_i & & (0) \\ b_i & \ddots & \ddots & & & \ddots & \\ & \ddots & \ddots & \ddots & & & b_i \\ & & \ddots & \ddots & \ddots & & \\ b_i & & & \ddots & \ddots & \ddots & \\ & \ddots & & & \ddots & \ddots & b_i \\ (0) & & b_i & & & b_i & a_{I_{n^2-1}} \end{pmatrix}$$

La troisième équation matricielle du système est la même que en une dimension spatiale, sauf que cette fois la matrice n'est pas de taille M mais de taille M^2.

3.2.2.2 Conditions limites

Une première approche pourrait être comme précédemment d'égaliser les termes du bord avec ceux qui sont à côtés (vers le centre). Cette approche se fait facilement mais elle est moins précise que celle choisie.

On choisit donc encore l'autre méthode qui est de dire que la valeurs du côté extérieur de la zone est la même que celle du côté intérieur de la zone. Ceci implique donc quelques changements dans la matrice.
Par exemple pour une système avec $n = 4$ on obtiendrait les matrices A suivantes :

$$\begin{pmatrix}
a_i & 2b_i & & & 2b_i & & & & & & & & & & (0) \\
b_i & \ddots & b_i & & & 2b_i & & & & & & & & & \\
 & b_i & \ddots & b_i & & & 2b_i & & & & & & & & \\
 & & 2b_i & \ddots & 0 & & & 2b_i & & & & & & & \\
b_i & & & 0 & \ddots & 2b_i & & & b_i & & & & & & \\
 & b_i & & & b_i & \ddots & b_i & & & b_i & & & & & \\
 & & b_i & & & b_i & \ddots & b_i & & & b_i & & & & \\
 & & & b_i & & & 2b_i & \ddots & 0 & & & b_i & & & \\
 & & & & b_i & & & 0 & \ddots & 2b_i & & & b_i & & \\
 & & & & & b_i & & & b_i & \ddots & b_i & & & b_i & \\
 & & & & & & b_i & & & b_i & \ddots & b_i & & & b_i \\
 & & & & & & & b_i & & & 2b_i & \ddots & 0 & & \\
 & & & & & & & & 2b_i & & & 0 & \ddots & 2b_i & \\
 & & & & & & & & & 2b_i & & & b_i & \ddots & b_i \\
(0) & & & & & & & & & & 2b_i & & & 2b_i & a_i
\end{pmatrix}$$

Il suffit donc de généraliser cela à n'importe quel M.

3.2.3 Modèle SZR 2D

Une fois le modèle SIR résolu numériquement, il est aisé de faire de même avec le modèle SZR car ils sont analogues.
Le modèle SZR est donc résolvable de la même manière que le modèle SIR, et la seule chose qui change est que le système d'équations est le suivant :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta SZ + \alpha_{diff}\Delta S \\ \dfrac{\partial Z}{\partial t} = \beta SZ - \gamma SZ + \alpha_{diff}\Delta Z \\ \dfrac{\partial R}{\partial t} = \gamma SZ \end{cases}$$

Pour la résolution du modèle SZR, il suffit donc de changer un terme dans la matrice diagonale de I (et donc Z) ainsi qu'un terme sur la résolution de R :
Le terme γS devient γSZ dans l'équation de Z et l'équation de R.

4 Simulations et Résultats

Les deux modèles simulés grâce aux résolutions numériques précédentes sont donc ces deux-là :

$$SIR : \begin{cases} \frac{\partial S}{\partial t} = -\beta SI + \alpha_{diff}\Delta S \\ \frac{\partial I}{\partial t} = \beta SI - \gamma I + \alpha_{diff}\Delta Z \\ \frac{\partial R}{\partial t} = \gamma I \end{cases}$$

$$SZR : \begin{cases} \frac{\partial S}{\partial t} = -\beta SZ + \alpha_{diff}\Delta S \\ \frac{\partial Z}{\partial t} = \beta SZ - \gamma SZ + \alpha_{diff}\Delta Z \\ \frac{\partial R}{\partial t} = \gamma SZ \end{cases}$$

Grâce aux simulations visuelles il va donc être possible de comparer ces deux modèles, et d'étudier l'importance des paramètres.

4.1 Modèle SZR

4.1.1 Influence du taux de résurrection ϵ

On rappelle qu'en rajoutant un terme de résurrection dans les équations, le modèle SZR devient :

$$SZR : \begin{cases} \frac{\partial S}{\partial t} = -\beta SZ \\ \frac{\partial Z}{\partial t} = \beta SZ - \gamma SZ + \epsilon R \\ \frac{\partial R}{\partial t} = \gamma SZ - \epsilon R \end{cases}$$

Dans certains modèles SZR, le terme de résurrection est absent. Cela est dû au fait que parfois l'épidémie de zombies est considérée comme n'affectant pas les morts. On simule des cas avec des coefficients de résurrection différents pour voir son influence :

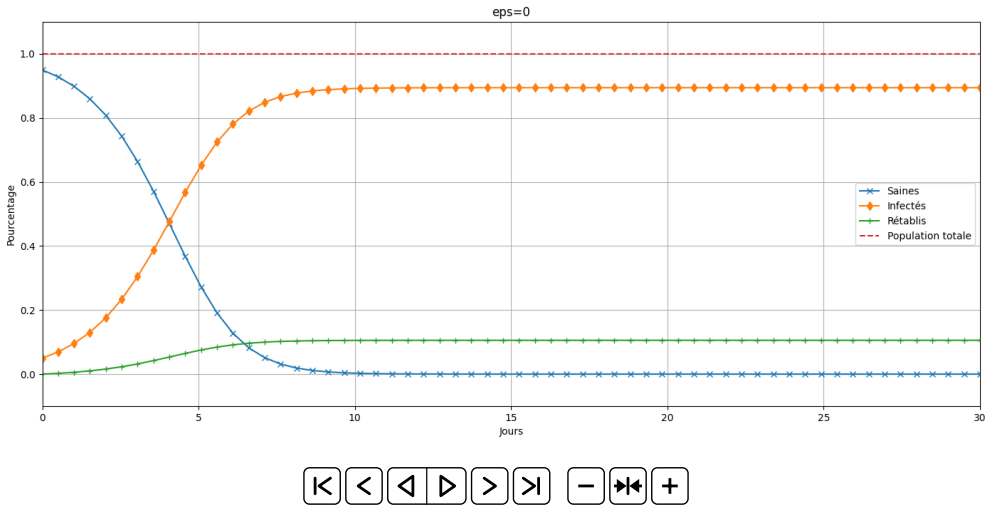


FIGURE 10 – Evolution du modèle SIR en fonction de ϵ
https://drive.google.com/open?id=12rPcPKkdHmfNz218McVNT-r6JQom7MNL

On remarque bien sur la figure (10) qu'avec un $\epsilon = 0$ on n'a jamais le nombre de zombies qui atteint le nombre d'individus de la population de départ.

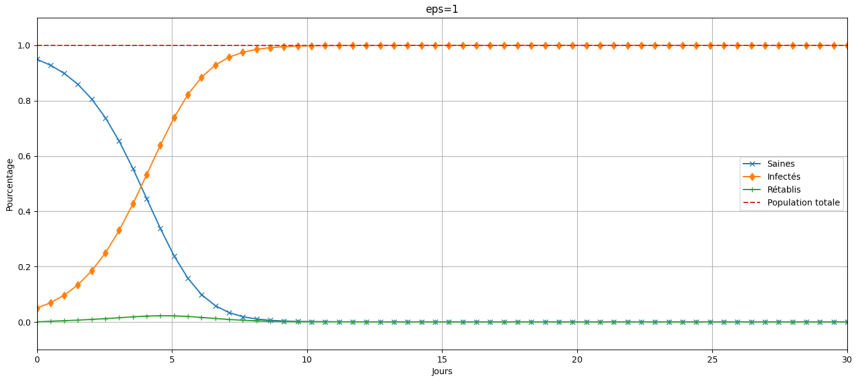


FIGURE 11 – $\epsilon = 1$

Alors que lorsque ϵ augmente et atteint 1 comme sur la figure (11), on a une population de zombies qui peut atteindre la population totale de départ.
Cela signifie donc qu'avec ϵ non nul la population de zombies dans la zone fermée ne pourra jamais décroître. Alors qu'avec ce coefficient nul on peut s'imaginer qu'au bout de très longtemps, avec un zombie qui se tue de temps en temps en déambulant, la zone pourrait redevenir habitable par une nouvelle population humaine.

4.1.2 Influence des Conditions Initiales sur le résultat du modèle SZR

On cherche à étudier la propagation des zombies en fonction du nombre de zombies initial. Dans un premier temps on procède à la simulation du modèle SZR sans déplacements des populations, afin d'avoir le cas le plus optimiste possible. On obtient donc :

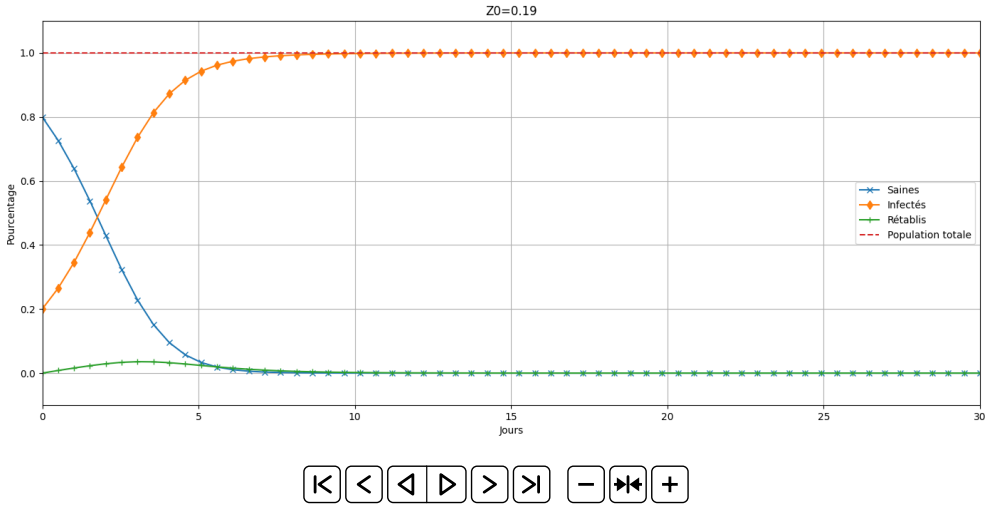


FIGURE 12 – Evolution du modèle SIR en fonction de $Z0$
https://drive.google.com/open?id=10Zx6jjhbOkeCpQ8kSAhwgFynj-YeptDq

On voit que quelque soit le nombre de zombies au départ, ils gagnent toujours avec ce modèle. Diminuer leur nombre retarde juste l'épidémie.

On compare ensuite les simulations avec déplacement des populations : 3D (https://drive.google.com/open?id=1YRFWupkB3YLimLu6PV4dPctuTO5qoq1i).
Les différences ne sont pas très significatives.

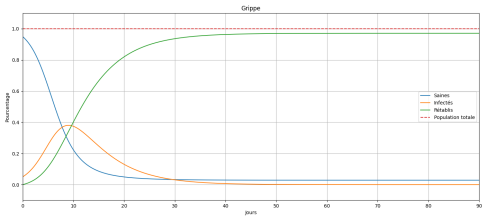
4.2 Modèle SIR

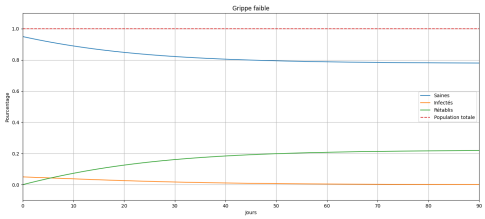
4.2.1 Simulation de maladies

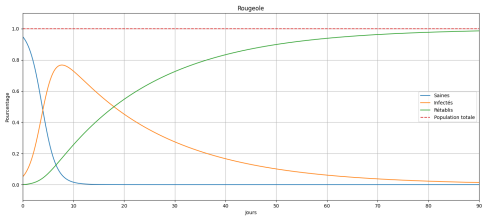
Outre le fait de simuler une propagation de zombies, on peut grâce au modèle SIR également simuler des épidémies de virus notoires.
Pour cela, on utilise les paramètres obtenus dans le document [2] :

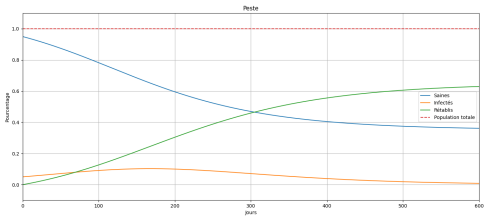
Maladie	β	γ	R_0
Grippe	0.6	1/6	3.6
Grippe faible	0.15	1/6	0.9
Rougeole	0.8	1/20	16
Peste	0.0273	1/56.2	1.53

En traçant pour chacun d'entre eux l'évolution du nombre d'individus dans chaque classe au cours du temps on obtient :





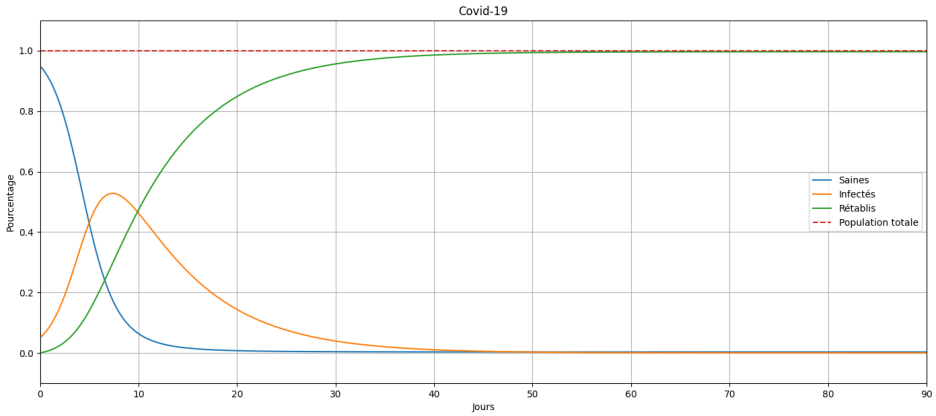




On peut voir que tous les virus évoluent différemment et que c'est bien sûr dû aux paramètres β et γ sur chaque maladies. Par exemple, le coefficient β de la peste est très petit (57.2) et on voit bien sur la courbe que le nombre d'infectés augmente lentement au début. De plus le coefficient γ est également très faible, et c'est pour cela que le nombre d'infectés décroît également lentement par la suite.

Quant au coefficient R_0 dans le tableau, il représente la force de propagation du virus. Plus il est élevé et plus le virus va se propager. Il s'obtient pas le calcul $R_0 = \beta/\gamma$.
Dans le contexte actuel, l'épidémie la plus préoccupante est le Covid-19. D'après le document [3] son coefficient R_0 vaut 5.7, car ses coefficients β et γ valent respectivement 0.76 et 1/7.5. C'est donc une épidémie qui se propage mieux que la peste ou la grippe, mais moins bien que la rougeole.

Avec ces données, on peut donc tracer l'évolution temporelle du nombre d'individus dans chaque classe de population lors d'une épidémie de Covid-19 :



On remarque que comme pour la grippe et la rougeole, lors du pic de l'épidémie, la classe de population qui compte le plus d'individus est celle des infectés. Mais le laps de temps durant lequel cette classe est la plus nombreuse est plus long que pour la grippe, et moins long que pour la rougeole.

On simule ensuite cette même épidémie, mais cette fois sur une zone en deux dimensions, et toujours en considérant qu'il n'y a pas de déplacement (https://drive.google.com/open?id=1az7HP6P2eYSyWex3UJn1dmFrAujtIneI). On observe que le pic avec ou sans déplacements de la population le pic d'infectés de l'épidémie a toujours lieu au même moment. Mais cela est due à la raison évoquée dans la partie 2, l'imprécision du maillage. Ce qui pourrait en revanche changer le moment du pic de l'épidémie serait d'alterner entre les deux cas, avec et sans déplacements.

5 Modèle topologique de la simulation

Jusqu'à présent, les simulations on été faites sur des topologies carrées où la population est répartie de manière linéaire. Mais pour bien modéliser une épidémie, il faut aussi prendre en compte la topologie sur laquelle elle se propage. C'est-à-dire avoir la vraie forme de la zone, avec la vraie densité de population.

5.1 Le programme dont on s'inspire

Pour commencer, on reprend un programme (8.4) qui modélise et simule la propagation d'une épidémie sur la topologie de la Suède. C'est aussi le modèle SIR qui en est la base :

$$SIR : \begin{cases} \dfrac{\partial S}{\partial t} = -\beta SI \\ \dfrac{\partial I}{\partial t} = \beta SI - \gamma I \\ \dfrac{\partial R}{\partial t} = \gamma I \end{cases} \tag{9}$$

Sauf que les déplacements ont également été ajoutés, mais d'une autre manière qu'avec un laplacien. Là chaque "case" est autant affectée par chacun de ses voisins que par elle-même.
Cela donne donc :

$$\begin{cases} \dfrac{\partial S}{\partial t} = -\beta(S_{i,j}I_{i,j} + S_{i-1,j}I_{i-1,j} + S_{i+1,j}I_{i+1,j} + S_{i,j-1}I_{i,j-1} + S_{i,j+1}I_{i,j+1}) \\ \dfrac{\partial I}{\partial t} = \beta(S_{i,j}I_{i,j} + S_{i-1,j}I_{i-1,j} + S_{i+1,j}I_{i+1,j} + S_{i,j-1}I_{i,j-1} + S_{i,j+1}I_{i,j+1}) - \gamma(I_{i,j} + I_{i-1,j} + I_{i+1,j}I_{i,j-1} + I_{i,j+1}) \\ \dfrac{\partial R}{\partial t} = \gamma(I_{i,j} + I_{i-1,j} + I_{i+1,j}I_{i,j-1} + I_{i,j+1}) \end{cases}$$

Cependant, ce modèle semble peu réaliste car il fait comme si quatre personnes sur cinq changeait de "case" chaque jour. Et il rend impossible la modification du taux de déplacement.
Mais le point fort de ce programme est qu'il utilise une vraie carte de densité de la population. Cela rend le modèle très proche e la réalité. Dans le cas de la France, une carte de densité de la population ressemble à cela :

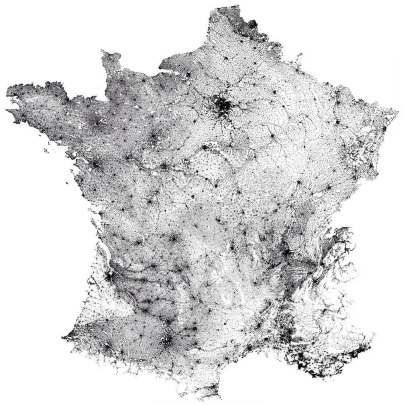


FIGURE 13 – Carte de densité de population française

Ensuite, dans le programme suédois, à l'aide du traitement d'image l'image est redimentionnée et inversée en couleurs afin que les zones où il y a de la population soient en blanc (pour ensuite y faire apparaître du rouge pour montrer la présence d'infectés) :

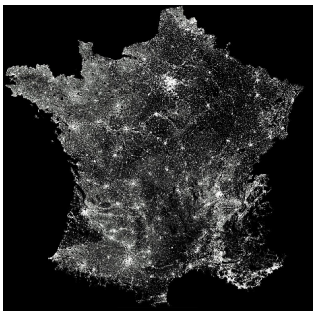
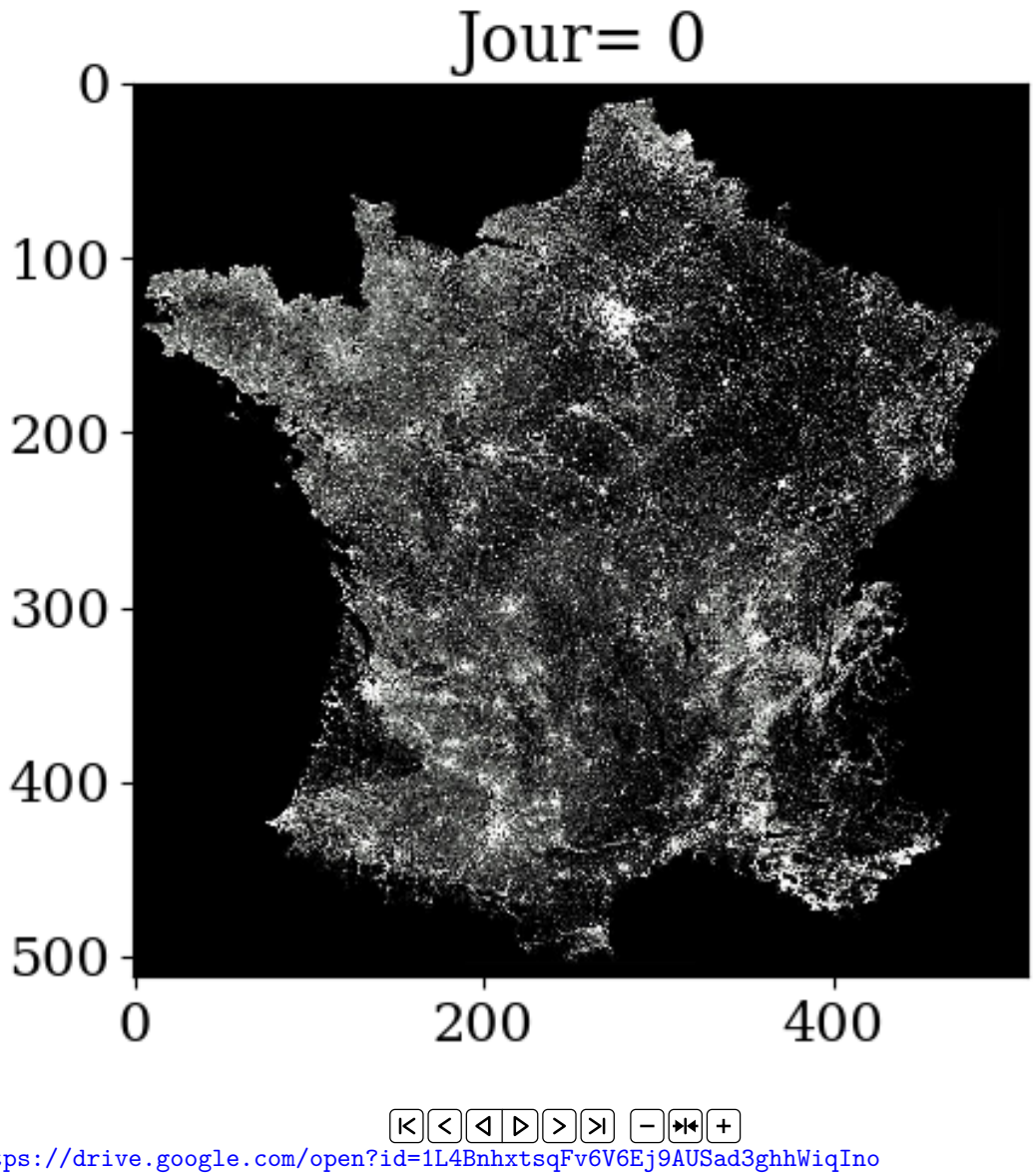


FIGURE 14 – France couleurs inversés

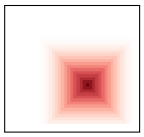
Ensuite la matrice de couleur est extraite de cette image (tableau n x n avec des valeurs $\in [0,255]$ et est affectée à la populations saine. Puis des seuils sont placés (les valeurs au dessus de 255 sont mises égales à 255 et celles en dessous de 0 à 0) afin d'avoir une visibilité plus nette (sans cela il y aurait des pics dans les grandes villes et on ne verrait pas que les autres endroits ont été infectés à cause de l'échelle de couleur) pour observer l'épidémie.
En reprenant ce modèle, en l'appliquant à la topologie de la France, et en faisant partir l'épidémie de Seatech on obtient cela :



https://drive.google.com/open?id=1L4BnhxtsqFv6V6Ej9AUSad3ghhWiqIno

Le résultat est que la totalité de la France se retrouve infectée au bout de 1500 jours.

Mais cela est dû à la manière dont les déplacements sont modélisés. En réalité l'épidémie est loin de se propager aussi bien. Le second problème de ce modèle est qu'il ne pose aucune condition limite. En effet, l'épidémie se propage dans la mer, comme les populations, mais on ne le voit pas car un filtre noir est posé autour de la zone qui est intéressante. Voici la propagation depuis Seatech si on enlève l'image derrière. On voit qu'elle se propage sans vraiment prendre en compte la topologie.



Le résultat est donc plus esthétique mais bien moins réaliste.

5.2 Résultat en incluant la topologie

On reprend donc l'idée du programme suédois, mais en changeant le modèle de l'épidémie (ici le modèle SZR), et surtout les conditions limites. [4].
Pour ce qui est de la population saine, on reprend la matrice de couleur mais divisée par 255 pour avoir des populations comprises entre [0,1] sur chaque éléments du maillage. Et de même que précédemment, on place initialement les zombies à SeaTech.

quant aux conditions limites, on impose encore un dérivée spatiale nulle sur chaque frontière du domaine Pour se faire, on parcourt l'ensemble du maillage et pour chaque cellule où il y a de la population, s'il y a une cellule vide qui est adjacente, on modifie la ligne qui lui correspond dans chaque matrice (nommées A dans la partie 3) suivant de quel côté est la cellule vide.

Cependant, pour les cas où il y a des cellules vides de part et d'autre, le programme va simplement modifier deux fois la ligne correspondante, et donc considérer qu'il y a une seule des deux cellules autour qui est vide. On pourrait donc différencier ces cas, mais le nombre est négligeable sur la topologie de la France, donc cela ferait beaucoup de calculs pour quelque chose de négligeable. Il est donc bien plus simple de fixer vides toutes les cellules qui le sont initialement.
On a donc une condition de Neuman presque partout sur la frontière du domaine, sauf à quelques endroits négligeables où il y a une condition de Dirichlet.
On obtient donc un résultat (figure (15) à peine moins proche de la réalité, mais avec beaucoup moins de calculs.

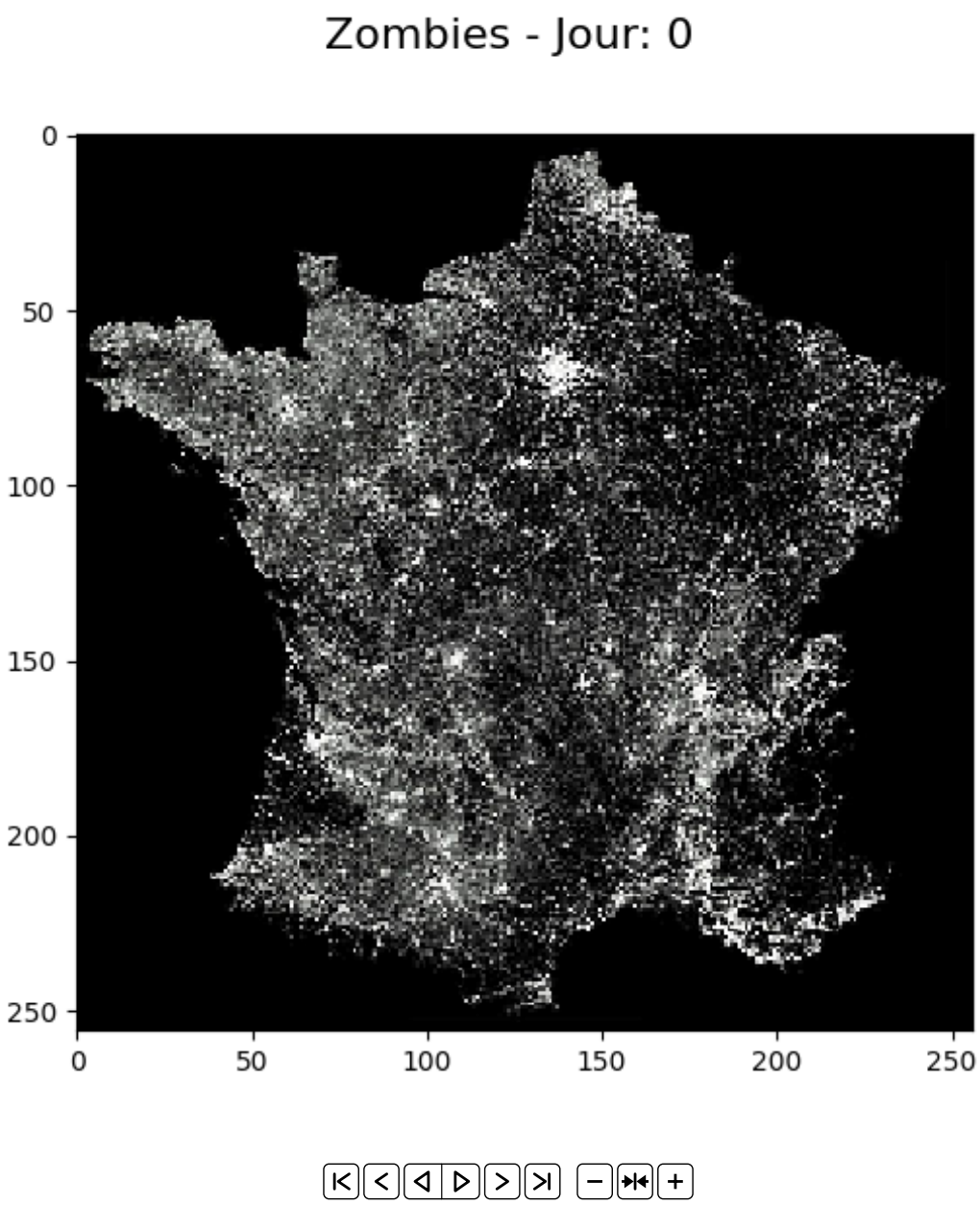


FIGURE 15 – https://drive.google.com/open?id=1NNzV9f94pma79vCAno6blZKrDoQcB-Fl

Bien sûr, il est encore possible d'améliorer la modélisation de la topologie. On pourrait par exemple ajouter les flux principaux de transport. Pour modéliser le fait qu'il y ait beaucoup d'échanges de population entre deux endroits, on pourrait comme avec le laplacien, imposer que les variations à ces endroits ne dépendent pas que de l'endroit et d'autour, mais aussi de l'autre endroit. Mais cela n'est pas forcément judicieux dans la modélisation d'une épidémie de zombies, car les flux de transports serait coupés par mesure de sécurité.

6 Conclusion

Ainsi, comme on a pu le vérifier, la méthode consistant à séparer la population en classes donne de bons résultats pour la modélisation d'une épidémie. Bien sûr il existe des méthodes plus précises, mais elles sont bien plus complexes. Celle-ci est donc un bon compromis entre précision et simplicité.
On a vu au travers de plusieurs modèles que ce qui est décisif pour la précision de la méthode est la manière dont les paramètres influent sur les classes. De plus, même si plus il y a de classes et de paramètres plus le modèle va être précis, on a pu constater que certaines classes et certains paramètres sont plus judicieux que d'autres, car ils apportent beaucoup plus de précision.
Ensuite, après avoir tenté une résolution numérique explicite, on a utilisé une résolution numérique implicite. Et on a pu constater que même si pour des "petits" systèmes d'équations la première est préférable, ce n'est pas le cas pour des grands systèmes car il n'y a pas de condition de stabilité qui impose de prendre un pas de temps très petit (inversement proportionnel au carré de la taille de la matrice de résolution). On a donc besoin de beaucoup moins d'itération pour obtenir le résultat que l'on veut.
Enfin, on a pu s'apercevoir que les simulations obtenues grâce à ces modèles peuvent être très utiles pour prédire l'évolution de l'épidémie, et donc réfléchier à comment la stopper. Mais pour pouvoir l'utiliser dans des cas réels, il est bien sûr nécessaire de prendre en compte la topologie du problème, car c'est analogue à des conditions initiales.
Il est bien sûr possible d'améliorer les modèles et les programmes utilisés, mais au vu de la puissance de calcul disponible ce qui est obtenu paraît très suffisant.

7 Bibliographie

Références

[1] W. Chinviriyasit, *Numerical modelling of an SIR epidemic model with diffusion*, 2010

[2] Gloria Facanoni, *Etude des modèles SIR/SZR en temporel*, 2020

[3] Coralie Lemke, *Covid-19 : chaque malade pourrait-il vraiment en contaminer 5,7 autres ?*, Avril 2020
https://www.sciencesetavenir.fr/sante/covid-19-chaque-personne-en-contaminerait-5-7-autres-en-moy
143573

[4] Max Berggren, *Model of a zombie outbreak in Sweden, Norway and Finland (Denmark is fine)*, Nov 27, 2014

[5] Corentin Bayette, *Modélisation d'une épidémie, partie 2*, April 23, 2020
https://images.math.cnrs.fr/Modelisation-d-une-epidemie-partie-2.html

8 Annexe

8.1 Résolution SIR diffusion 1D explicite

```
from matplotlib.pylab import * # importe aussi numpy sans alias

#Paramètres
beta = 0.9  # si =0  les saines restent sains
lamb = 10
nbj=40
alpha=0.001
S_0=0.95
I_0=1-S_0

#Spatial
n=40
X=linspace(0,1,n)
h=X[1]-X[0]

#Temporel
dt=.9*h**2/2
Tf=10

#Conditions initiales
S0=lambda z:2*S_0*z*(0<=z<0.5)-2*S_0*(z-1)*(0.5<=z<=1)
I0=lambda z:2*I_0*z*(0<=z<0.5)-2*I_0*(z-1)*(0.5<=z<=1)

#Initialisation
bS=array([S0(x) for x in X])
bI=array([I0(x) for x in X])
R=zeros(n)

#Affichage à T=0
plot(X,bS,'x-',label='Saines')
plot(X,bI,'d-',label='Infectés')
plot(X,R,'+-',label='Rétablis')
legend()
title("Jour=0")
show()

#Résolution
for t in range(int(Tf//dt)): #Boucle de temps

        #Copie des vecteurs
        Sc=bS.copy()
        Ic=bI.copy()
        Rc=R.copy()

        #Calcul sur les bords:
        j=0
        Sxx=(2*Sc[j+1]-2*Sc[j])/h**2
        Ixx=(2*Ic[j+1]-2*Ic[j])/h**2

        bS[j]=(-beta*Sc[j]*Ic[j]+alpha*Sxx)*dt+Sc[j]
        bI[j]=(-1/lamb*Ic[j]+beta*Sc[j]*Ic[j]+alpha*Ixx)*dt+Ic[j]
        R[j]=(1/lamb*Ic[j])*dt+Rc[j]

        j=n-1
```

```
        Sxx=(-2*Sc[j]+2*Sc[j-1])/h**2
        Ixx=(-2*Ic[j]+2*Ic[j-1])/h**2

        bS[j]=(-beta*Sc[j]*Ic[j]+alpha*Sxx)*dt+Sc[j]
        bI[j]=(-1/lamb*Ic[j]+beta*Sc[j]*Ic[j]+alpha*Ixx)*dt+Ic[j]
        R[j]=(1/lamb*Ic[j])*dt+Rc[j]

        #Calcul sans les bords:
        for j in range(1,n-1): #Boucle des x

                Sxx=(Sc[j+1]-2*Sc[j]+Sc[j-1])/h**2
                Ixx=(Ic[j+1]-2*Ic[j]+Ic[j-1])/h**2


                bS[j]=(-beta*Sc[j]*Ic[j]+alpha*Sxx)*dt+Sc[j]
                bI[j]=(-1/lamb*Ic[j]+beta*Sc[j]*Ic[j]+alpha*Ixx)*dt+Ic[j]
                R[j]=(1/lamb*Ic[j])*dt+Rc[j]



#Affichage à Tf
plot(X,bS,'x-',label='Saines')
plot(X,bI,'d-',label='Infectés')
plot(X,R,'+-',label='Rétablis')
legend()
title("Jour "+str(Tf))
show()
```

8.2 Résolution SIR diffusion 2D implicite

```
from matplotlib.pylab import * # importe aussi numpy sans alias
from scipy.sparse.linalg import spsolve
from scipy.sparse import spdiags
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
#Paramètres
beta = 0.9  # si =0  les saines restent sains
lamb = 10
nbj=40
alpha=0.001
S_0=0.95
I_0=1-S_0

#Spacial
n=30
M=n*n
X=linspace(0,1,n)
h=X[1]-X[0]

#Temporel
dt=.25
Tf=40



plt.ion()
fig = plt.figure()
ax = fig.gca(projection='3d')
```

```
#fig = plt.figure(figsize=plt.figaspect(0.5))


#Quelques fonctions utiles pour manier matrices-vecteurs

posA=lambda i,j: i*n+j #Position dans matrice sur vecteur
posR=lambda i: (int(i//n),int(i%n)) #Position dans vecteur sur matrice

#Passage matrice n x n à vecteur n^2
mat_v=lambda A: array([A[i,j]  for i in range(n) for j in range(n)])

def v_mat(v): #Passage vecteur n^2 à matrice n x n
        A=zeros((n,n))
        for k in range(M):
                i,j=posR(k)
                A[i,j]=v[k]
        return A

def ci_mat(A): #Condition initiale dx=0 et dy=0
        for i in range(1,n-1):
                A[0,i]=A[1,i]
                A[i,0]=A[i,1]
                A[n-1,i]=A[n-2,i]
                A[i,n-1]=A[i,n-2]
                A[0,0],A[0,n-1],A[n-1,0],A[n-1,n-1]=(A[1,0]+A[0,1])/2,(A[1,n-1]+A[1,n-2])/2,(A[n-2,0]
        return A


def tour(A,val,ind,n): #Remplissage de matrice par couronne
        A[ind,ind:n-ind]=val
        A[ind:n-ind,ind]=val
        A[n-1-ind,ind:n-ind]=val
        A[ind:n-ind,n-1-ind]=val
        return A

def plot_v(v): #plot un vecteur en 2d avec bon remplissage
        Av=v_mat(v)
        imshow(Av)
        plt.colorbar()
        show()

#Conditions initiales
S0=lambda z:2*S_0*z*(0<=z<0.5)-2*S_0*(z-1)*(0.5<=z<=1)
I0=lambda z:2*I_0*z*(0<=z<0.5)-2*I_0*(z-1)*(0.5<=z<=1)

#Initialisation

#Matrices des conditions initiales
A0S=zeros((n,n))
A0I=zeros((n,n))
for i in range(int((n+1)//2)):
        A0S=tour(A0S,S0(X[i]),i,n)
        A0I=tour(A0I,I0(X[i]),i,n)

#Vecteurs b
bS=mat_v(A0S)
bI=mat_v(A0I)
```

```
R=zeros(M)

for t in range(int(Tf/dt)): #Boucle de temps
        ax.clear()
        R=(bI/lamb)*dt+R #Solve R

        D0S=1*ones(M)+dt*(beta*bI+4*alpha/h**2*ones(M)) #Midle diag
        D0I=1*ones(M)+dt*(ones(M)*(1/lamb+4*alpha/h**2)-beta*bS)  #Midle diag

        D1p=-alpha*dt/h**2*ones(M) #Up/low diag

        D1m=-alpha*dt/h**2*ones(M) #Up/low diag

        DNp=-alpha*dt/h**2*ones(M)
        DNm=-alpha*dt/h**2*ones(M)
        DNp[:n]=2*DNp[:n]
        DNm[M-n-n-1:M-n]=2*DNm[M-n-1-n:M-n]


        D1p[n-1::n]=0 #correction des termes de bord
        D1m[n-1::n]=0 #correction des termes de bord

        D1p[n::n]=D1p[n::n]*2 #correction des termes de bord
        D1m[n-2::n]=D1m[n-2::n]*2 #correction des termes de bord



        AS=diag(D0S,0)+diag(D1p[0:M-1],1)+diag(D1m[0:M-1],-1)+diag(DNm[0:M-n],-n)+diag(DNp[0:M-n],n)
        AI=diag(D0I,0)+diag(D1p[0:M-1],1)+diag(D1m[0:M-1],-1)+diag(DNm[0:M-n],-n)+diag(DNp[0:M-n],n)


        AS=spdiags(D0S,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M

        AI=spdiags(D0I,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M

        AS[0,1]=2*AS[0,1]
        AS[M-1,M-2]=2*AS[M-1,M-2]
        AI[0,1]=2*AI[0,1]
        AI[M-1,M-2]=2*AI[M-1,M-2]

        bS=spsolve(AS, bS) #Solve AS
        bI=spsolve(AI, bI) #Solve AI

        # representation graphique de la solution
#         ax = fig.add_subplot(1, 2, 1, projection='3d')
        uu=reshape(bI,(n,n))
        uubord=0*ones((n+2,n+2));
        uubord[1:n+1,1:n+1]=uu;
        #print(A.toarray())
        X, Y = meshgrid(linspace(0,1,n+2),linspace(0,1,n+2))
        ax.set_zlim(0,1)
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        #ax.contour3D(X, Y, uubord, 50, cmap='binary')
        surf=ax.plot_surface(X, Y, uubord, rstride=1, cstride=1,
                             cmap='viridis', edgecolor='none',linewidth=0, antialiased = True);
        ax.set_xlabel('x')
        ax.set_ylabel('y')
```

```
        ax.set_zlabel('Population totale');
        ax.view_init(20, -125)
        ax.set_title('SAIN')

        show()
        pause(0.01)
```

8.3 Résolution SZR diffusion 2D implicite

```
from matplotlib.pylab import * # importe aussi numpy sans alias
from scipy.sparse.linalg import spsolve
from scipy.sparse import spdiags
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
#Paramètres
beta = 0.9  # si =0  les saines restent sains
lamb = 10
nbj=40
alpha=0.001
S_0=0.95
I_0=1-S_0

#Spacial
n=30
M=n*n
X=linspace(0,1,n)
h=X[1]-X[0]

#Temporel
dt=.25
Tf=40



plt.ion()
fig = plt.figure()
ax = fig.gca(projection='3d')

#fig = plt.figure(figsize=plt.figaspect(0.5))


#Quelques fonctions utiles pour manier matrices-vecteurs

posA=lambda i,j: i*n+j #Position dans matrice sur vecteur
posR=lambda i: (int(i//n),int(i%n)) #Position dans vecteur sur matrice

#Passage matrice n x n à vecteur n^2
mat_v=lambda A: array([A[i,j]  for i in range(n) for j in range(n)])

def v_mat(v): #Passage vecteur n^2 à matrice n x n
        A=zeros((n,n))
        for k in range(M):
                i,j=posR(k)
                A[i,j]=v[k]
        return A

def ci_mat(A): #Condition initiale dx=0 et dy=0
        for i in range(1,n-1):
```

```
                A[0,i]=A[1,i]
                A[i,0]=A[i,1]
                A[n-1,i]=A[n-2,i]
                A[i,n-1]=A[i,n-2]
                A[0,0],A[0,n-1],A[n-1,0],A[n-1,n-1]=(A[1,0]+A[0,1])/2,(A[1,n-1]+A[1,n-2])/2,(A[n-2,0]
        return A


def tour(A,val,ind,n): #Remplissage de matrice par couronne
        A[ind,ind:n-ind]=val
        A[ind:n-ind,ind]=val
        A[n-1-ind,ind:n-ind]=val
        A[ind:n-ind,n-1-ind]=val
        return A

def plot_v(v): #plot un vecteur en 2d avec bon remplissage
        Av=v_mat(v)
        imshow(Av)
        plt.colorbar()
        show()

#Conditions initiales
S0=lambda z:2*S_0*z*(0<=z<0.5)-2*S_0*(z-1)*(0.5<=z<=1)
I0=lambda z:2*I_0*z*(0<=z<0.5)-2*I_0*(z-1)*(0.5<=z<=1)

#Initialisation

#Matrices des conditions initiales
A0S=zeros((n,n))
A0I=zeros((n,n))
for i in range(int((n+1)//2)):
        A0S=tour(A0S,S0(X[i]),i,n)
        A0I=tour(A0I,I0(X[i]),i,n)

#Vecteurs b
bS=mat_v(A0S)
bI=mat_v(A0I)
R=zeros(M)

for t in range(int(Tf/dt)): #Boucle de temps
        ax.clear()
        R=(bI*bS/lamb)*dt+R #Solve R

        D0S=1*ones(M)+dt*(beta*bI+4*alpha/h**2*ones(M)) #Midle diag
        D0I=1*ones(M)+dt*(ones(M)*(bS/lamb+4*alpha/h**2)-beta*bS)  #Midle diag

        D1p=-alpha*dt/h**2*ones(M) #Up/low diag

        D1m=-alpha*dt/h**2*ones(M) #Up/low diag

        DNp=-alpha*dt/h**2*ones(M)
        DNm=-alpha*dt/h**2*ones(M)
        DNp[:n]=2*DNp[:n]
        DNm[M-n-n-1:M-n]=2*DNm[M-n-1-n:M-n]


        D1p[n-1::n]=0 #correction des termes de bord
        D1m[n-1::n]=0 #correction des termes de bord
```

```
        D1p[n::n]=D1p[n::n]*2 #correction des termes de bord
        D1m[n-2::n]=D1m[n-2::n]*2 #correction des termes de bord



        AS=diag(D0S,0)+diag(D1p[0:M-1],1)+diag(D1m[0:M-1],-1)+diag(DNm[0:M-n],-n)+diag(DNp[0:M-n],n)
        AI=diag(D0I,0)+diag(D1p[0:M-1],1)+diag(D1m[0:M-1],-1)+diag(DNm[0:M-n],-n)+diag(DNp[0:M-n],n)


        AS=spdiags(D0S,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M

        AI=spdiags(D0I,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M

        AS[0,1]=2*AS[0,1]
        AS[M-1,M-2]=2*AS[M-1,M-2]
        AI[0,1]=2*AI[0,1]
        AI[M-1,M-2]=2*AI[M-1,M-2]

        bS=spsolve(AS, bS) #Solve AS
        bI=spsolve(AI, bI) #Solve AI

        # representation graphique de la solution
#        ax = fig.add_subplot(1, 2, 1, projection='3d')
        uu=reshape(bI,(n,n))
        uubord=0*ones((n+2,n+2));
        uubord[1:n+1,1:n+1]=uu;
        #print(A.toarray())
        X, Y = meshgrid(linspace(0,1,n+2),linspace(0,1,n+2))
        ax.set_zlim(0,1)
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        #ax.contour3D(X, Y, uubord, 50, cmap='binary')
        surf=ax.plot_surface(X, Y, uubord, rstride=1, cstride=1,
                             cmap='viridis', edgecolor='none',linewidth=0, antialiased = True);
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('Population totale');
        ax.view_init(20, -125)
        ax.set_title('SAIN')

        show()
        pause(0.01)
#plot_v(bS)
#plot_v(bI)
#plot_v(R)
```

8.4 Propagation de zombies dans les pays du nord de [4]

```
import numpy as np
import math
import matplotlib.pyplot as plt
#%matplotlib inline
from matplotlib import rcParams
import matplotlib.image as mpimg
rcParams['font.family'] = 'serif'
rcParams['font.size'] = 16
rcParams['figure.figsize'] = 12, 8
```

```
from PIL import Image

beta = 0.010
gamma = 1

def euler_step(u, f, dt):
    return u + dt * f(u)

def f(u):
    S = u[0]
    I = u[1]
    R = u[2]

    new = np.array([-beta*(S[1:-1, 1:-1]*I[1:-1, 1:-1] + \
                           S[0:-2, 1:-1]*I[0:-2, 1:-1] + \
                           S[2:, 1:-1]*I[2:, 1:-1] + \
                           S[1:-1, 0:-2]*I[1:-1, 0:-2] + \
                           S[1:-1, 2:]*I[1:-1, 2:]),
                     beta*(S[1:-1, 1:-1]*I[1:-1, 1:-1] + \
                           S[0:-2, 1:-1]*I[0:-2, 1:-1] + \
                           S[2:, 1:-1]*I[2:, 1:-1] + \
                           S[1:-1, 0:-2]*I[1:-1, 0:-2] + \
                           S[1:-1, 2:]*I[1:-1, 2:]) - gamma*I[1:-1, 1:-1],
                     gamma*I[1:-1, 1:-1]
                    ])

    padding = np.zeros_like(u)
    padding[:,1:-1,1:-1] = new
    padding[0][padding[0] < 0] = 0
    padding[0][padding[0] > 255] = 255
    padding[1][padding[1] < 0] = 0
    padding[1][padding[1] > 255] = 255
    padding[2][padding[2] < 0] = 0
    padding[2][padding[2] > 255] = 255

    return padding

from PIL import Image
img = Image.open('popdens2.png')
img = img.resize((int(img.size[0]/2),int(img.size[1]/2)))
img = 255 - np.asarray(img)
imgplot = plt.imshow(img)
imgplot.set_interpolation('nearest')

S_0 = img[:,:,1]
I_0 = np.zeros_like(S_0)
I_0[309,170] = 1 # patient zero

R_0 = np.zeros_like(S_0)

T = 900                          # final time
dt = 1                           # time increment
N = int(T/dt) + 1                # number of time-steps
t = np.linspace(0.0, T, N)       # time discretization

# initialize the array containing the solution for each time-step
u = np.empty((N, 3, S_0.shape[0], S_0.shape[1]))
u[0][0] = S_0
```

```
u[0][1] = I_0
u[0][2] = R_0

import matplotlib.cm as cm
theCM = cm.get_cmap("Reds")
theCM._init()
alphas = np.abs(np.linspace(0, 1, theCM.N))
theCM._lut[:-3,-1] = alphas

for n in range(N-1):
    u[n+1] = euler_step(u[n], f, dt)

keyFrames = []
frames = 60.0

for i in range(0, N-1, int(N/frames)):
    imgplot = plt.imshow(img, vmin=0, vmax=255)
    imgplot.set_interpolation("nearest")
    imgplot = plt.imshow(u[i][1], vmin=0, cmap=theCM)
    imgplot.set_interpolation("nearest")
    filename = "outbreak" + str(i) + ".png"
    plt.savefig(filename)
    keyFrames.append(filename)

images = [Image.open(fn) for fn in keyFrames]
gifFilename = "outbreak.gif"
writeGif(gifFilename, images, duration=0.3)
plt.clf()
```

8.5 Résolution SZR 2D sur la France

```
from matplotlib.pylab import * # importe aussi numpy sans alias
from scipy.sparse.linalg import spsolve
from scipy.sparse import spdiags
import imageio
import os
import shutil
import matplotlib.cm as cm
from PIL import Image
# Colormap rouge
theCM = cm.get_cmap("Reds")
theCM._init()
alphas = abs(linspace(0, 1, theCM.N))
theCM._lut[:-3,-1] = alphas


###############
# Paramètres #
###############

#Spatial
coef=4 #Coefficient de division de taille d'image: initialement de taille 1024 x 1024
n=int(Image.open('france2.jpg').size[0]/coef) #Longueur du côté du maillage.
M=n*n #Taille du maillage total
X=linspace(0,1,n)
h=X[1]-X[0]

#Temporel
```

```
dt=2
Tf=60
nt=int(Tf/dt)

#Epidemie
beta = 0.9  # si =0  les saines restent sains
lamb = 10   # nombre de jours pour redevenir sain
size_pic=3  # taille du pic d'infecté
alpha=0.001 # coefficient de diffusion
eps=0       #Coefficient de résurection

#image
x_seatech=int(950/coef)
y_seatech=int(810/coef)
images = []
filenames=[]

#Sauvegarde solutions
uS=zeros((M,nt+1))
uI=zeros((M,nt+1))
uR=zeros((M,nt+1))

#Sauvegarde images
nom="SZR_256_F"
chemin="img_"+nom



######################
# FIN des Paramètres #
######################

#Création de dossier image
if os.path.exists(chemin):
        shutil.rmtree(chemin)
        os.mkdir(chemin)
else:
        os.mkdir(chemin)

print(nom)

############################
# Quelques fonctions utiles #
############################


posA=lambda i,j: i*n+j #Position dans matrice sur vecteur
posR=lambda i: (int(i//n),int(i%n)) #Position dans vecteur sur matrice

def v_mat(v): #Passage vecteur n^2 à matrice n x n
        A=zeros((n,n))
        for k in range(M):
                i,j=posR(k)
                A[i,j]=v[k]
        return A

def cc(ci,i,nbtr):
        return ci*(1-i/(nbtr+1))
```

```
def tr(A,i,j,ci,nbtr): # ci et tr permettent de créer une pyramide avec un "rayon"=nbtr
        A[i,j]=ci           # et une hauteur=ci
        for k in range(1,nbtr+1):
                A[i-k:i+k+1,j+k]=cc(ci,k,nbtr)*ones(1+2*k)
                A[i-k:i+k+1,j-k]=cc(ci,k,nbtr)*ones(1+2*k)
                A[i-k,j-k:j+k+1]=cc(ci,k,nbtr)*ones(1+2*k)
                A[i+k,j-k:j+k+1]=cc(ci,k,nbtr)*ones(1+2*k)
        return A

#Passage matrice n x n à vecteur n^2
mat_v=lambda A: array([A[i,j]  for i in range(n) for j in range(n)])

def plot_v(v): #plot un vecteur en 2d avec bon remplissage
        Av=v_mat(v)
        imshow(Av)
        plt.colorbar()
        show()


#####################################################
# Importation d'image, redimenssion et inversion #
#####################################################

fig = plt.figure(figsize=(6,6))
img = Image.open('france2.jpg')
img = img.resize((int(img.size[0]/coef),int(img.size[1]/coef)))
img = 255 - np.asarray(img)
imgplot = plt.imshow(img)
imgplot.set_interpolation('nearest')

#Sauvegarde de la première image
i=0
# representation graphique de la solution
imgplot = plt.imshow(img)
imgplot.set_interpolation("nearest")
imgplot = imshow(v_mat(zeros(M)), vmin=0, cmap=theCM)
imgplot.set_interpolation("nearest")

fig.suptitle("Zombies - Jour: "+str(i*dt), fontsize=16)
fig.savefig(chemin+"/"+"image-"+str(i)+".png")
S_0=img[:,:,1]
print(n)
for i in range(n):
        for j in range(n):
                if S_0[i,j]==0:
                        S_0[i,j]=0
                else:
                        S_0[i,j]=1
np.savetxt('Fr.dat', S_0) #exportation d'un fichier indiquant mer ou pas
b = loadtxt('Fr.dat') #charger ce fichier



########################
# Conditions initiales #
########################
```

```
A0S=copy(S_0/255) #pop saine: densité de population € [0,1]

A0I=zeros((n,n))
A0I=tr(A0I,x_seatech,y_seatech,1,6) # pop infectée: une pyramide d'infecté à SeaTech

#Vecteurs b: transformation matrices en vecteur
bS=mat_v(A0S)
bI=mat_v(A0I)
R=zeros(M)



# Préparation des plot
fig = plt.figure(figsize=(6,6))
img = Image.open('france2.jpg')
img = img.resize((int(img.size[0]/coef),int(img.size[1]/coef)))
img = 255 - np.asarray(img)


############################
# Résolution du problème #
############################

for t in range(nt): #Boucle de temps

        R=(bI*bS/lamb-eps*R)*dt+R #Solve R

        D0S=1*ones(M)+dt*(beta*bI+4*alpha/h**2*ones(M)) #Midle diag
        D0I=1*ones(M)+dt*(ones(M)*(bS/lamb+4*alpha/h**2+R*eps)-beta*bS)  #Midle diag

        D1p=-alpha*dt/h**2*ones(M) #Up_1 diag
        D1m=-alpha*dt/h**2*ones(M) #Down_1 diag
        DNp=-alpha*dt/h**2*ones(M) #Up_n diag
        DNm=-alpha*dt/h**2*ones(M) #Down_n diag

        AS=spdiags(D0S,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M
        AI=spdiags(D0I,[0],M,M)+spdiags(D1p,[1],M,M)+spdiags(D1m,[-1],M,M)+spdiags(DNm[0:M-n],[-n],M

        Mb = loadtxt('Fr.dat')
        # Condition de bord
        for i in range(n-1):
                for j in range(n-1):
                        if (Mb[i,j]==1) and (Mb[i+1,j]==0): #mer en bas
                                AI[posA(i,j),posA(i,j)+n]=0
                                AI[posA(i,j),posA(i,j)-n]=2*AI[posA(i,j)+n,posA(i,j)]
                                AS[posA(i,j),posA(i,j)+n]=0
                                AS[posA(i,j),posA(i,j)-n]=2*AS[posA(i,j)+n,posA(i,j)]

                        if (Mb[i,j]==1) and (Mb[i-1,j]==0): #mer en haut
                                AI[posA(i,j),posA(i,j)-n]=0
                                AI[posA(i,j),posA(i,j)+n]=2*AI[posA(i,j)+n,posA(i,j)]
                                AS[posA(i,j),posA(i,j)-n]=0
                                AS[posA(i,j),posA(i,j)+n]=2*AS[posA(i,j)+n,posA(i,j)]

                        if (Mb[i,j]==1) and (Mb[i,j+1]==0): #mer à droite
                                AI[posA(i,j),posA(i,j)+1]=0
                                AI[posA(i,j),posA(i,j)-1]=2*AI[posA(i,j)+n,posA(i,j)]
                                AS[posA(i,j),posA(i,j)+1]=0
```

```
                                AS[posA(i,j),posA(i,j)-1]=2*AS[posA(i,j)+n,posA(i,j)]

                        if (Mb[i,j]==1) and (Mb[i,j-1]==0): #mer à gauche
                                AI[posA(i,j),posA(i,j)-1]=0
                                AI[posA(i,j),posA(i,j)+1]=2*AI[posA(i,j)+n,posA(i,j)]
                                AS[posA(i,j),posA(i,j)-1]=0
                                AS[posA(i,j),posA(i,j)+1]=2*AS[posA(i,j)+n,posA(i,j)]

        bS=spsolve(AS, bS) #Solve AS
        bI=spsolve(AI, bI) #Solve AI

        #Passage de matrice
        mS=v_mat(bS)
        mI=v_mat(bI)

        #Condition de population nulle
        for i in range(n):
                for j in range(n):
                        if b[i,j]==0:
                                mS[i,j]=0
                                mI[i,j]=0

        # Passage en vecteur
        bS=mat_v(mS)
        bI=mat_v(mI)

        #Enregistrement de solution
        uS[:,t+1]=bS
        uI[:,t+1]=bI
        uR[:,t+1]=R

        imgplot = plt.imshow(img, vmin=0, vmax=255)
        imgplot.set_interpolation("nearest")
        imgplot = plt.imshow(v_mat(uI[:,t]), vmin=0, cmap=theCM)
        imgplot.set_interpolation("nearest")
        fig.suptitle("Zombies - Jour: "+str(t*dt), fontsize=16)
        fig.savefig(chemin+"/"+"image-"+str(t)+".png")

        print(str(t)+"/"+str(nt-1))        #Avancée des calculs
print("Cacluls OK")

# Liste des chemins vers les images
for i in range(0,nt):
        filenames.append(chemin+"/"+"image-"+str(i)+".png")

# Création de gif
for filename in filenames:
    images.append(imageio.imread(filename))
imageio.mimsave(nom+'.gif', images)
```