TD n°3 : Propagation d'une épidémie

Mars 2024

Nous allons étudier dans ce nouveau TD la propagation d'une épidémie par la méthode *des automates cellulaires*. Vous pouvez évidemment penser au virus de votre choix !

Dans ce qui suit, on appelle **grille** de taille n une liste de n listes de longueur n, où n est un entier strictement positif. Pour prendre en compte la dépendance spatiale de la contagion, on va simuler la propagation de l'épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants :

- Sain, représenté par l'entier 0 ;
- Infecté, représenté par l'entier 1 ;
- Rétabli, représenté par l'entier 2 ;
- Décédé, représenté par l'entier 3.

L'état des cases d'une grille va évoluer au cours du temps selon des règles simples. Dans le modèle considéré, l'état d'une case à l'instant $t+1$ ne dépend que de son état et de celui des cases voisines à l'instant t (une case a huit voisines si elle est placée au milieu de la grille, mais elle n'en a que cinq si elle est placée sur un bord et trois si elle est placée dans un coin). Les règles de transition sont les suivantes :

- une case décédée reste décédée ;
- une case infectée devient soit décédée avec une probabilité p_1, soit rétablie avec une probabilité $1-p_1$;
- une case rétablie reste rétablie (elle est immunisée contre la maladie) ;
- une case saine devient infectée avec une probabilité p_2 si elle a au moins une case infectée dans son voisinage, et elle reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté (elle joue le rôle du pangolin !).

1) Écrire une fonction `grille(n)` qui renvoie une grille de taille n ne contenant que des 0.
2) Écrire une fonction `init(n)` qui construit une grille `G` de taille n ne contenant que des cases saines, choisit aléatoirement une des cases et la transforme en case infectée, et enfin renvoie `G`.

   Remarque : pour tout entier p strictement positif, la fonction `randrange(p)` de la bibliothèque `random` renvoie un entier choisi aléatoirement entre 0 et `p-1` inclus.
3) Écrire une fonction `compte(G)` qui a prend en argument une grille `G` et qui renvoie la liste `[n0,n1,n2,n3]` formée des nombres de cases dans chacun des quatre états.
4) Créer une fonction `voisines(G,i,j)` qui détermine la liste des indices (ce sont des couples) des cases voisines de la case `(i,j)` dans la grille `G`.

   Pour une grille de taille n, les indices de ligne et de colonne doivent rester entre 0 et n-1.

D'après les règles de transition, pour savoir si une case saine peut devenir infectée à l'instant suivant, il faut déterminer si elle est exposée à la maladie, c'est-à-dire si elle possède au moins une case infectée dans son voisinage.

5) Écrire une fonction `est_exposee(G,i,j)` qui renvoie `True` si la case `(i,j)` de la grille `G` est exposée et `False` sinon.

6) La fonction `random()` de la bibliothèque `random` renvoie un nombre aléatoire pris entre 0 et 1, selon une loi uniforme. Utiliser cette fonction pour écrire une fonction `bernoulli(p)` qui simule une loi de Bernoulli de paramètre p : elle renvoie `True` avec une probabilité égale à p et `False` avec une probabilité égale à 1-p.

7) Écrire une fonction `suivant(G,p1,p2)` qui fait évoluer toutes les cases de la grille `G` à l'aide des règles de transition et renvoie une nouvelle grille correspondant à l'instant suivant. Les arguments `p1` et `p2` sont les probabilités qui interviennent dans les règles de transition pour les cases infectées et les cases saines.

Avec les règles de transition du modèle utilisé dans ce TD, l'état de la grille évolue entre les instants t et $t+1$ tant qu'il existe au moins une case infectée.

8) Écrire une fonction `simulation(n,p1,p2)` qui réalise une simulation complète avec une grille de taille n pour les probabilités `p1` et `p2`, et renvoie la liste `[x0,x1,x2,x3]` formée des proportions de cases dans chacun des quatre états à la fin de la simulation (c'est-à-dire lorsque la grille n'évolue plus).

9) Quelle est la valeur de la proportion de cases infectées `x1` à la fin d'une simulation ?

   Quelle relation vérifient à tout instant les nombres `x0`, `x1`, `x2` et `x3` ?

   Comment obtenir à l'aide de `x0`, `x1`, `x2` et `x3` la valeur `x_atteinte` de la proportion de cases qui ont été atteintes par la maladie au cours de la simulation ?

10) Écrire une fonction `pop_atteinte(n,p1,p2,N)` qui effectue `N` simulations complètes avec les paramètres n, `p1` et `p2`, et qui renvoie la valeur moyenne de `x_atteinte` pour ces `N` simulations.

On veut maintenant représenter la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité `p2`. Pour cela, on va fixer n à 10, `p1` à 0.5 et on calcule à l'aide de la fonction `pop_atteinte` la valeur moyenne de `x_atteinte` sur `N` simulations pour différentes valeurs de `p2` entre 0 et 1 (par exemple 100 valeurs). On obtient alors deux listes de même longueur :

- `Lp2` : liste des valeurs de `p2` utilisées
- `Lxa` : liste des valeurs moyennes de `x_atteinte` associées

ce qui permet de représenter graphiquement la proportion de la population atteinte en fonction de la valeur de la probabilité `p2`.

11) Écrire une fonction `simulation_globale(N)` qui renvoie les deux listes `Lp2` et `Lxa` obtenues pour une valeur du paramètre `N`.

    Écrire une procédure `representation(Lp2,Lxa)` qui utilise ces listes pour effectuer la représentation graphique évoquée ci-dessus.

Recommandation : avant de lancer la simulation pour de grandes valeurs de `N`*, il faut estimer le temps de calcul avec la fonction* `time()` *de la bibliothèque* `time`*. Par exemple, si l'on tape :*

```
t1=time.time()
Opérations diverses et variées
t2=time.time()
d=t2-t1
```

la variable `d` *nous donne le temps de calcul nécessaire. Ceci permet de voir jusqu'à quelle valeur de* `N` *on peut raisonnablement aller.*

Si l'on lance une simulation qui tourne pendant plusieurs heures, on va obtenir des données plus précises. Évidemment, on ne va pas relancer 5 heures de calcul à chaque fois ! Il va donc falloir sauvegarder les résultats de la simulation dans un fichier texte, et être capable de récupérer ces résultats à partir du fichier de sauvegarde.

12) Écrire une fonction `Sauvegarde(nom,Lp2,Lxa)` qui écrit les contenus des listes `Lp2` et `Lxa` dans le fichier texte `nom`.

    Les éléments de `Lp2` seront sur la première ligne, ceux de `Lxa` sur la seconde ligne, et les différents éléments de chaque liste seront séparés par des espaces

13) Écrire une fonction `Lecture(nom)` qui récupère les listes `Lp2` et `Lxa` à partir d'un fichier texte créé comme ci-dessus.

On appelle **seuil critique de pandémie** la valeur de `p2` à partir de laquelle plus de la moitié de la population a été atteinte par la maladie à la fin de la simulation. On suppose que les valeurs de `p2` et `x_atteinte` calculées lors de la question 11 sont stockées dans deux listes de même longueur `Lp2` et `Lxa`.

14) Écrire une fonction `seuil(Lp2,Lxa)` qui détermine par dichotomie un encadrement `[p2cmin,p2cmax]` du seuil critique de pandémie avec la plus grande précision possible. On rappelle que la liste `Lp2` croît de 0 à 1 et que la liste `Lxa` des valeurs correspondantes de `x_atteinte` est également croissante.

Pour étudier l'effet d'une campagne de vaccination, on immunise au hasard à l'instant initial une fraction q de la population.

15) Écrire une fonction `init_vac(n,q)` qui modifie la fonction `init(n)` de sorte que $q \times n^2$ cases soient dans l'état 2.

    Attention à ne pas immuniser deux fois la même case.

16) Refaire les simulations précédentes (représentation de `x_atteinte` en fonction de `Lp2` et calcul du seuil critique) avec une population vaccinée.

    On adaptera les fonctions `simulation`, `pop_atteinte` et `simulation_globale` et l'on prendra n=10, p1=0.5 et q=0.2 pour la simulation globale.