Informatique - DS n°1 - Lundi 8 juin 2020, 2 heures

Fonctions, méthode d'Euler

L'étude de la propagation des épidémies joue un rôle important dans les politiques de santé publique. Les modèles mathématiques ont permis de comprendre pourquoi il a été possible d'éradiquer la variole à la fin des années 1970 et pourquoi il est plus difficile d'éradiquer l'apparition d'épidémies de grippe tous les hivers. Aujourd'hui, des modèles de plus en plus complexes et puissants sont développés pour prédire la propagation d'épidémies à l'échelle planétaire telles que le SRAS, le virus H5N1 ou le virus Ebola. Ces prédictions sont utilisées par les organisations internationales pour établir des stratégies de prévention et d'intervention.
Le travail sur ces modèles mathématiques s'articule autour de trois thèmes principaux : traitement de base des données, simulation numérique (par plusieurs types de méthodes), identification des paramètres intervenant dans les modèles à partir de données expérimentales. Ces trois thèmes sont abordés dans le sujet. *Les parties sont indépendantes.*
Dans tout le problème, on peut utiliser une fonction traitée précédemment. On suppose que les bibliothèques `numpy` et `random` ont été importées par :

```
import numpy as np
import random as rd
```

Partie I. Modèle à compartiments

On s'intéresse ici à une première méthode de simulation numérique.

Les modèles compartimentaux sont des modèles déterministes où la population est divisée en plusieurs catégories selon leurs caractéristiques et leur état par rapport à la maladie. On considère dans cette partie un modèle à quatre compartiments disjoints : sains (S, "susceptibles"), infectés (I,"infected"), rétablis(R,"recovered", ils sont immunisés) et décédés (D,"dead"). Le changement d'état des individus est gouverné par un système d'équations différentielles obtenues en supposant que le nombre d'individus nouvellement infectés (c'est-à-dire le nombre de ceux qui quittent le compartiment S) pendant un intervalle de temps donné est proportionnel au produit du nombre d'individus infectés avec le nombre d'individus sains.
En notant $S(t)$, $I(t)$, $R(t)$ et $D(t)$ la fraction de la population appartenant à chacune des quatre catégories à l'instant t, on obtient le système :

$$(I)\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}S(t) &= -rS(t)I(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}I(t) &= rS(t)I(t) - (a+b)I(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}R(t) &= aI(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}D(t) &= bI(t) \end{cases}$$

avec r le taux de contagion, a le taux de guérison et b le taux de mortalité. On suppose qu'à l'instant initial $t = 0$, on a $S(0) = 0,95$, $I(0) = 0,05$ et $R(0) = D(0) = 0$.

1) Préciser un vecteur X et une fonction f (en donnant son domaine de définition et son expression) tels que le système différentiel (I) s'écrive sous la forme

$$\frac{d}{dt}X = f(X)$$

2) Compléter la ligne 4 du code suivant (on précise que `np.array` permet de créer un tableau `numpy` à partir d'une liste donnant ainsi la possibilité d'utiliser les opérateurs algébriques).

```
1  def f(X):
2      """Fonction definissant l'equation differentielle"""
3      global r,a,b
4      # a completer
5
6  # Parametres
7  tmax=25.
8  r=1.
9  a=0.4
10 b=0.1
11 X0=np.array([0.95,0.05,0.,0.])
12
13 N=250
14 dt=tmax/N
15
16 t=0
17 X=X0
18 tt=[t]
19 XX=[X]
20
21 # Methode d'Euler
22 for i in range(N):
23     t=t+dt
24     X=X+dt*f(X)
25     tt.append(t)
26     XX.append(X)
```

3) La figure 1 représente les quatre catégories en fonction du temps obtenues en effectuant deux simulations : la première avec $N = 7$ correspondant aux points (cercle, carré, losange, triangle) et la seconde avec $N = 250$ correspond aux courbes. Expliquer la différence entre ces deux simulations. Quelle simulation a nécessité le temps de calcul le plus long ?

En pratique, de nombreuses maladies possèdent une phase d'incubation pendant laquelle l'individu est porteur de la maladie mais ne possède par de symptômes et n'est pas contagieux. On peut

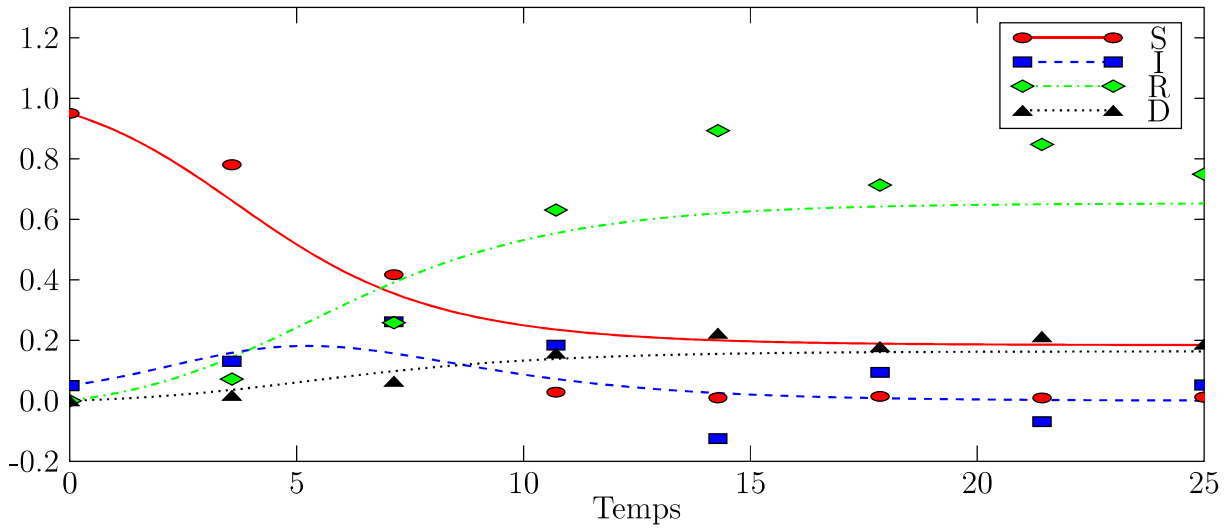

Figure 1 - Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes)

prendre en compte cette phase d'incubation à l'aide du système à retard suivant :

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}S(t) &= -rS(t)I(t-\tau) \\ \frac{\mathrm{d}}{\mathrm{d}t}I(t) &= rS(t)I(t-\tau) - (a+b)I(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}R(t) &= aI(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}D(t) &= bI(t) \end{cases}$$

où τ est le temps d'incubation. On suppose alors que pour tout $t \in [-\tau, 0]$, $S(t) = 0,95$, $I(t) = 0,05$ et $R(t) = D(t) = 0$.

En notant $tmax$ la durée des mesures et N un entier donnant le nombre de pas, on définit le pas de temps $dt = tmax/N$. On suppose que $\tau = p \times dt$ où p est un entier ; ainsi p est le nombre de pas de retard. Pour résoudre numériquement ce système d'équations différentielles à retard (avec `tmax=25, N=250` et `p=50`), on a écrit le code suivant :

```
1  def f(X,Itau):
2      """
3      Fonction definissant l'equation differentielle
4      Itau est le valeur de I(t-p*dt)
5      """
6      global r,a,b
7      # a completer
8
9  # Parametres
10 r=1.
11 a=0.4
12 b=0.1
13 X0=np.array([0.95,0.05,0.,0.])
14
```

```
15 tmax=25.
16 N=250
17 dt=tmax/N
18 p=50
19
20 t=0
21 X=X0
22 tt=[t]
23 XX=[X]
24
25 # Methode d'Euler
26 for i in range(N):
27     t=t+dt
28     # a completer
29     tt.append(t)
30     XX.append(X)
```

4) Compléter les lignes 7 et 28 du code précédent (utiliser autant de lignes que nécessaire)

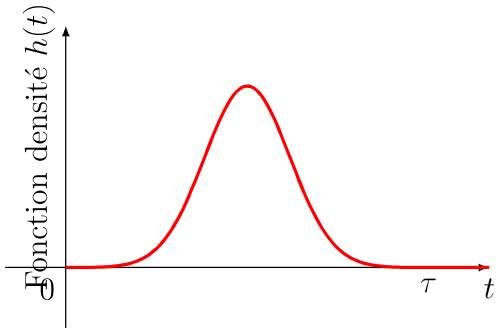

Figure 2 - Exemple d'une fonction de densité

On constate que le temps d'incubation de la maladie n'est pas nécessairement le même pour tous les individus. On peut modéliser cette diversité à l'aide d'une fonction positive d'intégrale unitaire (dite de densité) $h : [0, \tau] \to \mathbb{R}_+$ telle que représentée sur la figure 2. On obtient alors le système intégro-différentiel :

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}S(t) &= -rS(t)\int_0^\tau I(t-s)h(s)\mathrm{d}s \\ \frac{\mathrm{d}}{\mathrm{d}t}I(t) &= rS(t)\int_0^\tau I(t-s)h(s)\mathrm{d}s - (a+b)I(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}R(t) &= aI(t) \\ \frac{\mathrm{d}}{\mathrm{d}t}D(t) &= bI(t) \end{cases}$$

On supposera à nouveau que pour tout $t \in [-\tau, 0]$, $S(t) = 0,95$, $I(t) = 0,05$ et $R(t) = D(t) = 0$. Pour j entier compris entre 0 et N, on pose $t_j = j \times dt$. Pour un pas de temps dt donné, on peut calculer numériquement l'intégrale à l'instant t_i $(0 \leq i \leq N)$ à l'aide de la méthode des rectangles à gauche en utilisant l'approximation :

$$\int_0^\tau I(t_i - s)h(s)\mathrm{d}s \approx dt \times \sum_{j=0}^{p-1} I(t_i - t_j)h(t_j).$$

5) On suppose que la fonction h a été écrite en Python. Expliquer comment modifier le programme de la question précédente pour résoudre ce système intégro-différentiel (on explicitera les lignes de code nécessaires)

Partie II. Modélisation dans des grilles

On s'intéresse ici à une seconde méthode de simulation numérique (dite *par automates cellulaires*).

Dans ce qui suit, on appelle grille de taille $n \times n$ une liste de n listes de longueur n, où n est un entier strictement positif.
Pour mieux prendre en compte la dépendance spatiale de la contagion, il est possible de simuler la propagation d'une épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants : saine, infectée, rétablie, décédée. On choisit de représenter ces quatre états par les entiers :

0 (Sain), 1 (Infecté), 2 (Rétabli) et 3 (Décédé)

L'état des cases d'une grille évolue au cours du temps selon des règles simples. On considère un modèle où l'état d'une case à l'instant $t+1$ ne dépend que de son état à l'instant t et de l'état de ses huit cases voisines à l'instant t (une case du bord n'a que 5 cases voisines et trois pour une case d'un coin). Les *règles de transition* sont les suivantes :

- une case décédée reste décédée ;
- une case infectée devient décédée avec une probabilité p_1 ou rétablie avec une probabilité $(1-p_1)$;
- une case rétablie reste rétablie ;
- une case saine devient infectée avec une probabilité p_2 si elle a au moins une case voisine infectée et reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté.

6) On a écrit en Python la fonction `grille(n)` suivante :

```
1  def grille(n):
2      M=[]
3      for i in range(n):
4          L=[]
5          for j in range(n): L.append(0)
6          M.append(L)
7      return M
```

Décrire ce que retourne cette fonction.

On pourra dans la question suivante utiliser la fonction `randrange(p)` de la bibliothèque `random` qui, pour un entier positif p, renvoie un entier choisi aléatoirement entre 0 et $p-1$ inclus.

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

8) Écrire en Python une fonction `compte(G)` qui a pour argument une grille G et renvoie la liste `[n0,n1,n2,n3]` formée des nombres de cases dans chacun des quatre états.

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. Pour cela, on écrit en Python la fonction `est_exposee(G,i,j)` suivante :

```
1  def est_exposee(G,i,j):
2      n=len(G)
3      if i == 0 and  j == 0:
4          return (G[0][1]-1)*(G[1][1]-1)*(G[1][0]-1) == 0
5      elif i == 0 and j == n-1:
6          return (G[0][n-2]-1)*(G[1][n-2]-1)*(G[1][n-1]-1) == 0
7      elif i == n-1 and j == 0:
8          return (G[n-1][1]-1)*(G[n-2][1]-1)*(G[n-2][0]-1) == 0
9      elif i == n-1 and j == n-1:
10         return (G[n-1][n-2]-1)*(G[n-2][n-2]-1)*(G[n-2][n-1]-1) == 0
11     elif  i== 0:
12         # a completer
13     elif i == n-1:
14         return (G[n-1][j-1]-1)*(G[n-2][j-1]-1)*(G[n-2][j]-1)*\
15                (G[n-2][j+1]-1)*(G[n-1][j+1]-1) == 0
16     elif j == 0:
17         return (G[i-1][0]-1)*(G[i-1][1]-1)*(G[i][1]-1)*\
18                (G[i+1][1]-1)*(G[i+1][0]-1) == 0
19     elif j == n-1:
20         return (G[i-1][n-1]-1)*(G[i-1][n-2]-1)*(G[i][n-2]-1)*\
21                (G[i+1][n-2]-1)*(G[i+1][n]-1) == 0
22     else:
23         # a completer
```

9) Quel est le type du résultat renvoyé par la fonction `est_exposee` ?

10) Compléter les lignes 12 et 23 de la fonction `est_exposee`.

11) 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. On pourra utiliser la fonction `bernoulli(p)` suivante qui simule une variable aléatoire de Bernoulli de paramètre p : `bernoulli(p)` vaut 1 avec la probabilité p et 0 avec la probabilité $1-p$.

```
1  def bernoulli(p):
2      x=rd.random()
3      if x <=p:
4          return 1
5      else:
6          return 0
```

On reproduit ci-dessous le descriptif de la documentation Python concernant la fonction `random` de la bibliothèque `random` :

```
random.random()
    random() -> x in the interval [0, 1).
```

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

12) Écrire en Python une fonction `simulation(n,p1,p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ 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 (une simulation s'arrête lorsque la grille n'évolue plus).