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

1 : Mines-Pont 2016

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

Partie I. Modèle à compartiments

1) En posant $X = \begin{pmatrix} S \\ I \\ R \\ D \end{pmatrix}$, le système différentiel se réécrit :

$$(I) \left\{ \begin{array}{rcl} \dfrac{\mathrm{d}}{\mathrm{d}t}X_0(t) & = & -rX_0(t)X_1(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}X_1(t) & = & rX_0(t)X_1(t) - (a+b)X_1(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}X_2(t) & = & aX_1(t) \\ \dfrac{\mathrm{d}}{\mathrm{d}t}X_3(t) & = & bX_1(t) \end{array} \right.$$

Posons donc $f : X \in \mathcal{F}(\mathbb{R}, \mathbb{R}^4) \mapsto f(X) = \begin{pmatrix} -rX_0X_1 \\ rX_0X_1 - (a+b)X_1 \\ aX_1 \\ bX_1 \end{pmatrix} \in \mathcal{F}(\mathbb{R}, \mathbb{R}^4)$

2) Conformément au résultat de la question précédente :

```python
def f(X):
    """Fonction definissant l'equation differentielle"""
    global r,a,b
    return X[1]*np.array([-r*X[0],r*X[0]-(a+b),a,b])
```

3) La simulation effectuée pour $N = 7$ a fait 7 pas de boucle en prenant pour la méthode d'Euler $\mathrm{d}t = \dfrac{25}{7}$ (dans l'unité imposée par le problème, qui n'est pas précisée...). Elle effectue donc moins de calcul que la simulation pour $N = 250$ (qui fait 250 pas de boucle !), mais est aussi beaucoup plus approximative que cette seconde pour laquelle $\mathrm{d}t = 10^{-1}$.

4)

```python
def f(X,Itau):
    """
    Fonction definissant l'equation differentielle
    Itau est le valeur de I(t-p*dt)
    """
    global r,a,b
    return [-r*X[0]*Itau,r*X[0]*Itau-(a+b)*X[1],a*X[1],b*X[1]]
```

```python
# Methode d'Euler
for i in range(N):
    t=t+dt
    if i<p:
        X=X+dt*f(X,XX[0][1])
    else:
        X=X+dt*f(X,XX[-p][1])
    tt.append(t)
    XX.append(X)
```

5) On modifie le code de la question précédente ainsi :

```python
# Methode d'Euler
for i in range(N):
    t=t+dt
    if i<p:
        aXX=np.array(XX)
        I=[0.05 for k in range(p-i)]+aXX[:,1].tolist()
        X=X+dt*f(X,I)
    else:
        aXX=np.array(XX)
        I=aXX[-p:,1]
        X=X+dt*f(X,I)
    tt.append(t)
    XX.append(X)
```

où

```python
def f(X,I):
    """
    Fonction definissant l'equation differentielle
    Itau est le valeur de I(t-p*dt)
    """
    global r,a,b,h,dt
    integrale=dt*sum([h(j*dt)*I[-1-j] for j in range(p)])
    return [-r*X[0]*integrale,r*X[0]*integrale-(a+b)*X[1],a*X[1],b*X[1]]
```

Partie II. Modélisation dans des grilles

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

```python
def grille(n):
    M=[]
    for i in range(n):
        L=[]
```

```python
        for j in range(n): L.append(0)
        M.append(L)
    return M
```

Cette fonction initialise la liste `M` à la liste vide puis lui ajoute à chaque pas de la première boucle `for` la liste `L` qui est elle-même composée de `n` zéros.
Cette fonction retourne donc une grille de 0 (tous les individus sont sains).

7)

```python
def init(n):
    M=grille(n)
    i,j=rd.randrange(n),rd.randrange(n)
    M[i][j]=1
    return M
```

8)

```python
def compte(G):
    n0,n1,n2,n3=0,0,0,0
    n=len(G)
    for i in range(n):
        for j in range(n):
            if G[i][j]==0:
                n0+=1
            elif G[i][j]==1:
                n1+=1
            elif G[i][j]==2:
                n2+=1
            else:
                n3+=1
    return [n0,n1,n2,n3]
```

9) Le résultat renvoyé par la fonction `est_exposee` est de type booléen.

10) Les lignes 12 et 23 peuvent être remplacées par :

```python
def est_exposee(G,i,j):
    ...
    elif  i== 0:
        return (G[0][j-1]-1)*(G[1][j-1]-1)*(G[1][j]-1)*\
               (G[1][j+1]-1)*(G[0][j+1]-1)  == 0
    ...
    else:
        return (G[i][j-1]-1)*(G[i+1][j-1]-1)*(G[i+1][j]-1)*\
```

```
                (G[i+1][j+1]-1)*(G[i][j+1]-1)*(G[i-1][j-1]-1)\
                *(G[i-1][j]-1)*(G[i-1][j+1]-1) == 0
```

11)

```
def suivant(G,p1,p2):
    n=len(G)
    H=[[0 for k in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            if G[i][j]==0:
                if est_exposee(G,i,j):
                    H[i][j]=bernoulli(p2)
            elif G[i][j]==1:
                H[i][j]=2+bernoulli(p1)
            elif G[i][j]==2:
                H[i][j]=2
            else:
                H[i][j]=3
    return H
```

12)

```
def simulation(n,p1,p2):
    G=init(n)
    n1=1
    pas=0
    while n1>0:
        G=suivant(G,p1,p2)
        n0,n1,n2,n3=compte(G)
        pas+=1
    return n0/n**2,0,n2/n**2,n3/n**2,pas
```