TD 3 :
Concours commun Mines-Ponts 2016

PC/PC* - Lycée Thiers

Corrigé-Q10.

On peut choisir $X = (S, I, R, D)$, la fonction $f : \mathbb{R}^4 \longrightarrow \mathbb{R}^4$ est alors donnée par :

$$f(X) = (-rSI,\ rSI - (a+b)I,\ aI,\ bI)$$

Corrigé-Q10.

On peut choisir $X = (S, I, R, D)$, la fonction $f : \mathbb{R}^4 \longrightarrow \mathbb{R}^4$ est alors donnée par :

$$f(X) = (-rSI,\ rSI - (a+b)I,\ aI,\ bI)$$

Corrigé-Q11.

On peut compléter la ligne 4 :

```
    return np.array([-r*X[0]*X[1], r*X[0]*X[1] - (a+b)*X[1],
                                                  a*X[1], b*X[1]])
```

Corrigé-Q10.

On peut choisir $X = (S, I, R, D)$, la fonction $f : \mathbb{R}^4 \longrightarrow \mathbb{R}^4$ est alors donnée par :

$$f(X) = (-rSI,\ rSI - (a+b)I,\ aI,\ bI)$$

Corrigé-Q11.

On peut compléter la ligne 4 :

```
    return np.array([-r*X[0]*X[1], r*X[0]*X[1] - (a+b)*X[1],
                                                 a*X[1], b*X[1]])
```

Corrigé-Q12.

Avec $N = 7$, le pas est très grossier : la valeur approchée obtenue n'est pas une très bonne approximation de la solution. On obtient même des valeurs négatives. Avec $N = 250$, le pas est plus petit : la valeur approchée obtenue est une très bonne approximation de la solution exacte. La méthode d'Euler effectue une évaluation de la fonction f à chaque pas. La simulation avec $N = 250$ a necessité le temps de calcul le plus long.

Corrigé-Q13.

Ligne 7 :

```
    return np.array([-r*X[0]*Itau,r*X[0]*Itau-(a+b)*X[1],
                                                a*X[1],b*X[1]])
```

Ligne 28 :

```
    if i < p:                # si (t-p*dt) est négatif
        Itau = X0[1]            # Condition Initiale
    else:
        Itau = XX[-p-1][1]     # I(t-p*dt)
    X = X + dt * f(X, Itau)
```

Corrigé-Q14.

Seul le schéma d'Euler est à modifier pour, préalablement à l'appel de la fonction `f`, calculer l'intégrale considérée.

```
# Méthode d'Euler
for i in range(N):
    t = t+dt
    sItau = 0.0              # valeur de l'intégrale sur [0, tau]
    for j in range(p):
        if i < j:              # si (t_i-t_j) est négatif
            Itau = X0[1]         # I(0)
        else:
            Itau = XX[-j-1][1]         # I(t_i-t_j)
        sItau = sItau + dt * h(j * dt) * Itau
    X = X + dt * f(X, sItau)
    tt.append(t)
    XX.append(X)
```

Corrigé-Q15.

La fonction retourne une grille de taille $n \times n$ ne contenant que des cases saines, c'est à dire une liste de n listes, chacune de ces dernières contenant n fois le nombre 0.

Corrigé-Q15.

La fonction retourne une grille de taille $n \times n$ ne contenant que des cases saines, c'est à dire une liste de n listes, chacune de ces dernières contenant n fois le nombre 0.

Corrigé-Q16.

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

Corrigé-Q15.

La fonction retourne une grille de taille $n \times n$ ne contenant que des cases saines, c'est à dire une liste de n listes, chacune de ces dernières contenant n fois le nombre 0.

Corrigé-Q16.

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

Corrigé-Q17.

```
def compte(G):
    n = len(G)
    NN = [0, 0, 0, 0]
    for i in range(n):
        for j in range(n):
            NN[G[i][j]] += 1
    return NN
```

Corrigé-Q18.

Booléen.

Corrigé-Q19.

On complète la ligne 12 (il y a cinq voisins pour une case de la ligne 0 qui n'est pas dans un coin) :

```
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
```

et la ligne 20 (il y a huit voisins pour une case qui n'est pas sur le bord) :

```
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
```

Corrigé-Q20.

```
def suivant(G, p1, p2):
    n = len(G)
    new_G = grille(n)    # toutes les cases sont saines
    for i in range(n):
        for j in range(n):
            if G[i][j] == 3:            # si la case est Décédée
                new_G[i][j] = 3
            elif G[i][j] == 0:            # si la case est saine
                if est_exposée(G, i, j):    # si voisine Infect.
                    new_G[i][j] = bernoulli(p2)
            elif G[i][j] == 2 :         # si case rétablie
                new_G[i][j] = 2
            else:          # sinon la case est infectée
                new_G[i][j] = 2 + bernoulli(p1)
    return new_G
```

Corrigé-Q21.

```
def simulation(n, p1, p2):
    G = init(n)
    NN = compte(G)
    while NN[1] != 0:
        G = suivant(G, p1, p2)
        NN = compte(G)
    x0 = NN[0] / n**2
    x1 = NN[1] / n**2
    x2 = NN[2] / n**2
    x3 = NN[3] / n**2
    return [x0, x1, x2, x3]
```

Corrigé-Q21.

```
def simulation(n, p1, p2):
    G = init(n)
    NN = compte(G)
    while NN[1] != 0:
        G = suivant(G, p1, p2)
        NN = compte(G)
    x0 = NN[0] / n**2
    x1 = NN[1] / n**2
    x2 = NN[2] / n**2
    x3 = NN[3] / n**2
    return [x0, x1, x2, x3]
```

Corrigé-Q22.

A la fin d'une simulation, il n'y a plus de case infectée, donc `x1=0`.
Les n^2 cases de la grille sont dans l'un des 4 états donc $N0+N1+N2+N3=n^2$.
puis `x0+x1+x2+x3=1`.
Come il n'y a plus de case infectée à la fin d'une simulation, et qu'une case infectée ne redevient pas saine, on a `x_atteinte = x2+x3 = 1-x0`.

Corrigé-Q23.

```
def seuil(Lp2, Lxa):
    n = len(Lp2)
    imin = 0
    imax = n-1
    while imax-imin > 1:
        im = (imin+imax)//2
        if Lxa[im] < 0.5
            imin = im
        else:
            imax = im
    return [Lp2[imin], Lp2[imax]]
```

Corrigé-Q23.

```
def seuil(Lp2, Lxa):
    n = len(Lp2)
    imin = 0
    imax = n-1
    while imax-imin > 1:
        im = (imin+imax)//2
        if Lxa[im] < 0.5
            imin = im
        else:
            imax = im
    return [Lp2[imin], Lp2[imax]]
```

Corrigé-Q24.

on ne peut pas enlever le test sinon on n'aura pas forcément le bon nombre de vaccinations (certains tirage aléatoires pouvant conduire à vacciner deux fois le même individu). Une autre raison est qu'on ne peut vacciner qu'un individu sain.

Corrigé-Q25.

L'appel renvoie une grille 5×5 contenant exactement cinq personnes vaccinées, une personne infectée et 19 personnes saines (non vacinées).