Devoir d'informatique 4 – Corrigé

D.Malka – MPSI 2018-2019 – Lycée Jeanne d'Albret

19.04.2019

1 Modèle à compartiments pour la propagation d'une épidémie

1. Vecteur $X = \begin{pmatrix} S(t) \\ I(t) \\ R(t) \\ D(t) \end{pmatrix}$

   En posant :

$$f : X = (S(t), I(t), R(t), D(t)) \underset{\mathbb{R}_+^4}{} \to (-rS(t)I(t), rS(t)I(t) - (a+b)I(t), aI(t), bI(t)) \underset{\mathbb{R}_- \times \mathbb{R}\mathbb{R}_+^2}{}$$

   Le système d'équations différentielles s'écrit :

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

2. La ligne 25 montre que l'on doit pouvoir utiliser des opérations algébriques sur X et $f(X)$ donc il faut renvoyer un objet de type `ndarray` et non pas de type `list`.

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

3. La méthode d'Euler est d'autant plus précise que le pas `dt` est petit c'est-à-dire, à intervalle fixé, que le nombre de point `N` est élevé. La figure 1 montre, par la dispersion des points autour des courbes, que

pour $N = 7$ l'erreur locale et l'erreur de convergence globale est plus importante que pour $N = 250$. La méthode d'Euler étant de complexité linéaire, le temps de calcul nécessaire à la simulation pour $N = 250$ est plus grand que pour $N = 7$.

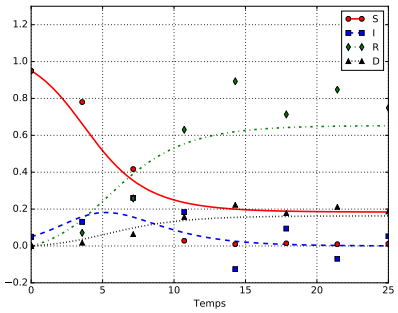

FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

4. Prise en compte de l'incubation.

```
def f(X,Itau):
    """
    Fonction definissant l'equation differentielle
    Itau est la valeur de I(t-p*dt)
    """
    global r,a,b
    S=X[0];I=X[1];R=X[2];D=X[3]
    rhsS=-r*S*Itau
    rhsI=r*S*Itau-(a+b)*I
    rhsR=a*I
    rhsD=b*I
    return np.array([rhsS,rhsI,rhsR,rhsD])

#Parametres
r=1.
a=0.4
b=0.1
X0=np.array([0.95,0.05,0.,0.])

tmax=25.
N=250
dt=tmax/N
p=50

t=0
X=X0
tt=[t]
XX=[X]

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

5. `Itau` se calcule à chaque itération à l'aide de la fonction `rectangle`. En l'état, la complexité de l'algorithme est $O(Np)$ ce qui est peu efficace. En effet, on recalcule l'intégrale complètement à chaque itération i alors qu'il suffirait d'ajouter le terme `dt*XX[i-j][1]*h(j*dt)` et de soustraire `dt*XX[i-j-p][1]*h((j-p)*dt)`

à la valeur de `Itau` calculé à l'itération $i-1$. On gagnerait en complexité à prendre cet élément en compte mais on perdrait en clarté du programme.

```
def rectangle(XX,h,i,j,p,dt):
    Itau=0
    for j in range(p):
        if i-j<p:
            Itau+=dt*XX[0][1]*h(j*dt)
        else:
            Itau=dt*XX[i-j][1]*h(j*dt)
    return Itau

def f(X,Itau):
    """
    Fonction definissant l'equation differentielle
    Itau est la valeur de I(t-p*dt)
    """
    global r,a,b
    S=X[0];I=X[1];R=X[2];D=X[3]
    rhsS=-r*S*Itau
    rhsI=r*S*Itau-(a+b)*I
    rhsR=a*I
    rhsD=b*I
    return np.array([rhsS,rhsI,rhsR,rhsD])

#Parametres
r=1.
a=0.4
b=0.1
X0=np.array([0.95,0.05,0.,0.])

tmax=25.
N=250
dt=tmax/N
p=50

t=0
X=X0
tt=[t]
XX=[X]

#Methode d'Euler
for i in range(N):
    t=t+dt
    Itau=rectangle(XX,h,i,j,p,dt)
    X=X+f(X,Itau)*dt
    tt.append(t)
    XX.append(X)
```

2 Méthode d'optimisation : recherche du minimum d'une fonction

Le but de cet exercice est de proposer un algorithme inspiré de la l'algorithme de dichotomie pour approcher le minimum d'une fonction convexe sur un segment de $\mathbb{R}$.

1. Dichotomie

1.1 Fonction `bisect` :

```
def bisect(f,a,b,e):
    assert (f(a)*f(b)<0 and a<b),'a doit etre > b et f doit changer de signe sur [a,b]'
    g=a;d=b
    while (d-g)>e:
        x=(g+d)/2
        if f(x)*f(g)>0:
            g=x
        elif f(x)*f(g)<0:
```

```
            d=x
        else:
            return x
    return (d+g)/2,(d-g)
```

1.2 Montrons la terminaison de la fonction `bisect`. Pour cela, considérons que la suite $v_i = d_i - g_i$ et montrons qu'elle devient inférieure à e partir d'un certain rang n. La suite v_i est définie par $v_i = d_i - g_i$. Or à chaque itération soit $f(x_i) = 0$ et l'algorithme termine de manière triviale soit $d_{i+1} = \dfrac{d_i + g_i}{2}$ et $g_{i+1} = g_i$, soit $g_{i+1} = \dfrac{d_i + g_i}{2}$ et $d_{i+1} = d_i$. Dans ces deux derniers cas : $v_{i+1} = \dfrac{d_i - g_i}{2} = \dfrac{v_i}{2}$. Comme $v_0 = b - a$, on montre par récurrence que $v_i = \dfrac{b-a}{2^i}$. Ainsi :
— v_i est une suite strictement décroissante,
— $\lim_{i\to\infty} = 0$.
Donc, il existe un certain rang n tel que $v_n = d_n - g_n < e$ et l'algorithme termine.

1.3 Pour montrer que la fonction `bisect` est correcte, il faut montrer que la proposition $\mathcal{P}_i : x_i \in [g_i, d_i]$ est un invariant de boucle. Voir cours.

2. Minimum d'une fonction convexe

2.1 A l'aide d'une illustration graphique dire si chacune des conjectures suivantes vous paraît raisonnable :
— $p_0 \leq p_1 \leq p_2 \leq 0 \Rightarrow \mu \in [x_2, b]$: fig.2a.
— $p_0 \leq p_1 \leq 0 \leq p_2 \Rightarrow \mu \in [x_1, b]$: fig.2b.
— $p_0 \leq 0 \leq p_1 \leq p_2 \Rightarrow \mu \in [a, x_2]$: fig.2c.
— $0 \leq p_0 \leq p_1 \leq p_2 \Rightarrow \mu \in [a, x_1]$: fig.2d.
Il n'y a aucun autre cas possible car, par définition de la convexité de f, $\forall x \in [a, b]$, $f'(x)$ croît et, attendu que la pente de la sécante au même point a une pente plus grande que la dérivée, les pentes des sécantes successives ne peuvent que croître quels que soient x_2 et x_1 (fig.**??**). Soit $p_2 > p_1 > p_0$.

2.2 Fonction `min_function` :

```
pente=lambda f,u,v:(f(v)-f(u))/(v-u);

def min_function(f,a,b,e):
    l=1/3
    g,d=a,b
    while (d-g)>e:
        m=(d+g)/2
        x1=m-l*(d-g);x2=m+l*(d-g)
        p0=pente(f,g,x1); p1=pente(f,x1,x2); p2=pente(f,x2,d)
        if  p0<=0 and p1<=0 and p2<=0:
            g=x2
        elif p0<=0 and p1<=0 and p2>=0:
            g=x1
        elif p0<=0 and p1>=0 and p2>=0:
            d=x2
        else:# p0>=0 and p1>=0 and p2>=0:
            d=x1
    return m
```

2.3 Exprimons la complexité, dans le pire des cas, de la fonction `min_function`. Le nombre d'itérations donc l'ordre de grandeur du nombre de comparaisons est une fonction de l'évolution de la longueur de l'intervalle de recherche $l = d - g$. On note $(l_i)_{i\in\mathbb{N}}$ associée avec $l_{i+1} = d_{i+1} - g_{i+1}$.
— dans le cas 1 : $g_{i+1} = x_2 = \dfrac{d_i + g_i}{2} - \lambda(d_i - g_i)$ et $d_{i+1} = d_i$ d'où :

$$l_{i+1} = d_i - \frac{d_i + g_i}{2} + \lambda(d_i - g_i) = \frac{d_i - g_i}{2} + \lambda(d_i - g_i) = \left(\lambda + \frac{1}{2}\right) l_i$$

— dans le cas 2 : $g_{i+1} = x_1 = \dfrac{d_i + g_i}{2} - \lambda(d_i - g_i)$ et $d_{i+1} = d_i$ d'où :

$$l_{i+1} = d_i - \frac{d_i + g_i}{2} - \lambda(d_i - g_i) = \frac{d_i - g_i}{2} - \lambda(d_i - g_i) = \left(\lambda - \frac{1}{2}\right) l_i$$

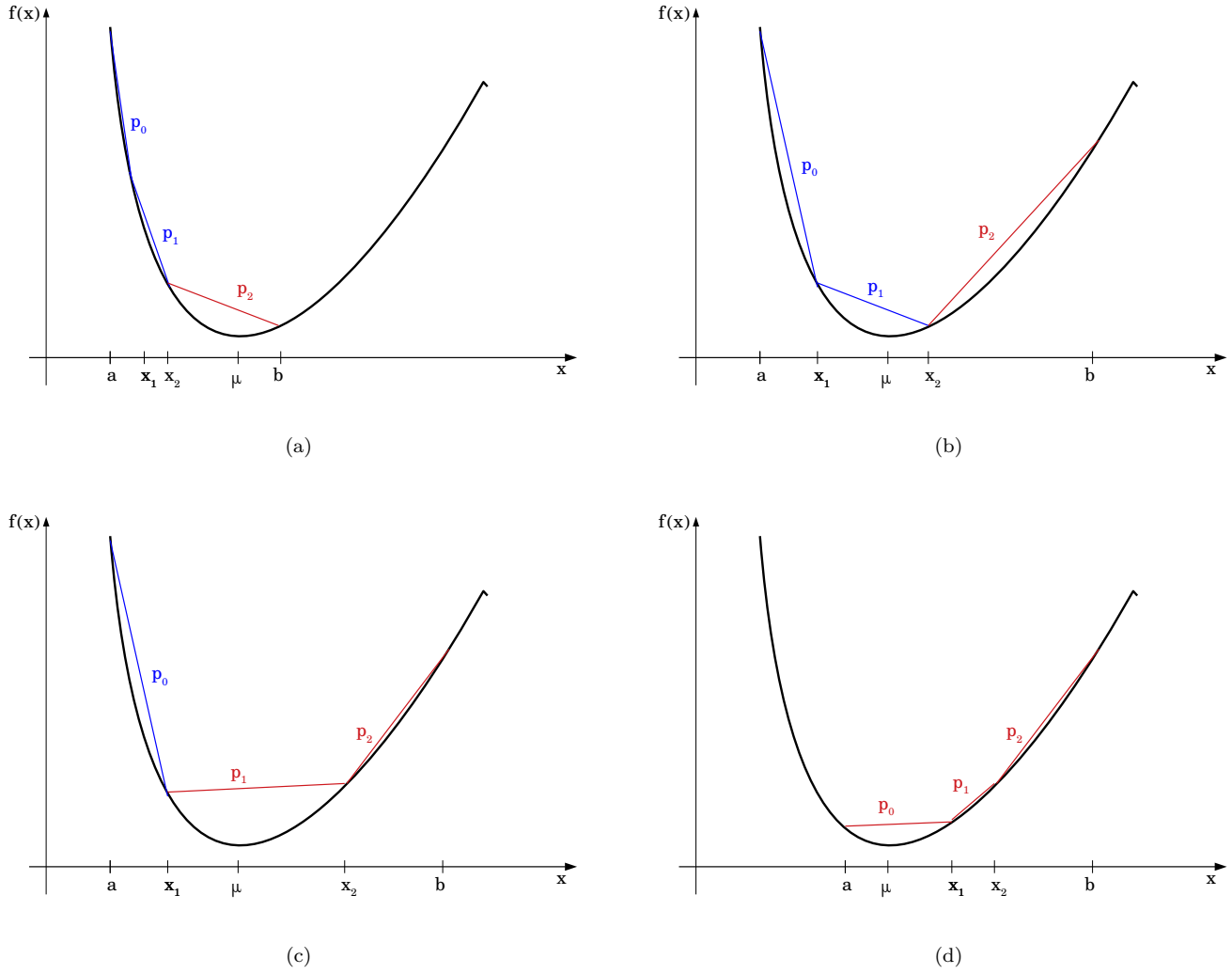

FIGURE 2 – Les différentes signes possibles des pentes des sécantes aux points d'abscisses a, x_1, x_2 et b.

— dans le cas 3 : $d_{i+1} = x_2 = \dfrac{d_i + g_i}{2} - \lambda(d_i - g_i)$ et $g_{i+1} = g_i$ d'où :

$$l_{i+1} = \frac{d_i + g_i}{2} + \lambda(d_i - g_i) - g_i = \frac{d_i - g_i}{2} + \lambda(d_i - g_i) = \left(\lambda + \frac{1}{2}\right) l_i$$

— dans le cas 4 : $d_{i+1} = x_1 = \dfrac{d_i + g_i}{2} - \lambda(d_i - g_i)$ et $g_{i+1} = g_i$ d'où :

$$l_{i+1} = \frac{d_i + g_i}{2} - \lambda(d_i - g_i) - d_i = \frac{d_i - g_i}{2} - \lambda(d_i - g_i) = \left(\lambda - \frac{1}{2}\right) l_i$$

Comme $0 < \lambda < 1/2$:

— les relations de récurrence $l_{i+1} = \left(\lambda + \dfrac{1}{2}\right) l_i$ et $l_{i+1} = \left(\lambda - \dfrac{1}{2}\right) l_i$ assure la convergence de la suite vers 0 et donc la terminaison de l'algorithme,

— la relation $l_{i+1} = \left(\lambda + \dfrac{1}{2}\right) l_i$ assure la convergence la plus lente.

Ainsi le pire des cas correspond à une exécution de l'algorithme conduisant à un test satisfaisant la relation $l_{i+1} = \left(\lambda + \dfrac{1}{2}\right) l_i$. Comme $l_0 = (b - a)$, il vient :

$$l_i = \left(\lambda + \frac{1}{2}\right)^i (b - a)$$

Soit n le nombre d'itération avant terminaison de l'algorithme alors, au rang $n - 1$:

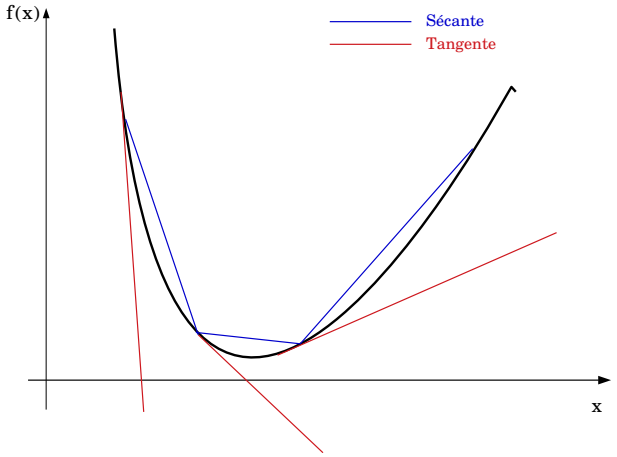

FIGURE 3 – Deux conséquences de la convexité de la fonction f. Premièrement, par définition, la dérivée donc la pente des tangentes, croît. Deuxième, la pente d'une sécante est toujours plus élevée que la pente de la tangente au point gauche de la sécante.

$$l_{n-1} > e \Leftrightarrow \left(\lambda + \frac{1}{2}\right)^n (b-a) < e$$

Soit :

$$\boxed{n < \frac{\ln\left(\frac{e}{b-a}\right)}{\ln\left(\lambda + \frac{1}{2}\right)} - 1}$$

Donc la complexité de la fonction dans le pire des cas $O\left(\frac{\ln\left(\frac{e}{b-a}\right)}{\ln\left(\lambda + \frac{1}{2}\right)}\right)$.

3 Simulation d'une file de voitures

1. **Préliminaires**

1.1 On implémente la file de voitures par une liste `A` de booléens, l'élément i de la liste représentant la case i de la file. Si la case i est occupée par une voiture alors `L[i]` vaut `True`, si la case i est libre alors `L[i]` vaut `False`. Exemple fig.4.

```
A=[True, False, True, True, False, False, False, False, False, False, True]
```

FIGURE 4 – Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10 ; et son implémentation par une liste de booléens.

1.2 La fonction `occupe(L,i)` n'a qu'à renvoyer la valeur `L[i]` ! L'intérêt d'une telle fonction réside seulement dans la lisibilité du code à suivre. Voir fig.5.

```
1 def occupe(L,i):
2     return L[i]
```

FIGURE 5 – Fonction `occupe`

1.3 La file comprenant n cases et chaque case ne pouvant prendre que deux états possibles (`True` ou `False`), il existe 2^n files de longueur n.

2. **Déplacement des voitures dans une file**

2.1 L'instruction `avancer(avancer(A,False),True)` appelle deux fois la fonction `avancer` : une première fois sans introduire de nouvelle voiture à gauche (`avancer(A,False)`), puis une deuxième fois sur la liste renvoyée par `avancer(A,False)` en introduisant une nouvelle voiture à gauche (`avancer(avancer(A,False),True)`). Résultat de l'appel dans le tableau fig.**??**.

Instruction	Liste
`A=[True, False.....True]`	`[True, False, True, True, False, False, False, False, False, False, True]`
`avancer(A,False)`	`[False, True, False, True, True, False, False, False, False, False, False]`
`avancer(avancer(A,False),True)`	`[True, False, True, False, True, True, False, False, False, False, False]`

FIGURE 6 – Liste renvoyée par l'instruction `avancer(avancer(A,False),True)`

2.2 Implémentation en `Python` la fonction `avancer` : fig.7.

```
1  def avancer(L,voit_depart):
2      n=len(L)
3      Q=L[:]#copie locale pour ne pas modifier la liste pointee par L
4      #Dans tous les cas, la case la plus a droite prend la valeur False initialement
5      #puisqu'elle est soit vide, soit la voiture qui l'occupe sort de la file
6      Q[n-1]=False
7      #On deplace les voitures de la droite vers la gauche donc il faut parcourir
8      #la liste a l'envers
9      i=n-2
10     while i>=0:
11         if occupe(Q,i) and not occupe(Q,i+1):
12             Q[i+1]=True
13             Q[i]=False
14         i=i-1
15     #Voiture la plus a gauche
16     Q[0]=voit_depart
17     return Q
```

FIGURE 7 – Fonction `avancer`