Devoir d'informatique 4

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

19.04.2019

Durée de l'épreuve : 2h00.

L'usage de la calculatrice n'est pas autorisé.

L'énoncé de ce devoir comporte 6 pages.

- *Si, au cours de l'épreuve, vous repérez ce qui vous semble être une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition en expliquant les raisons des initiatives que vous êtes amené à prendre.*
- *Il ne faudra pas hésiter à formuler des commentaires. Le barème tiendra compte de ces initiatives ainsi que des qualités de rédaction de la copie.*
- *La numérotation des exercices doit être respectée. Les résultats doivent être systématiquement encadrés.*
- *Les pages doivent être numérotées de la façon suivante : n°page courante/nombre total de pages.*

Données pour tous les exercices.

On rappelle que :
- La méthode `append()` ajoute l'argument en queue de liste. Appel : `l.append(elt)`.
- La fonction `len()` renvoie un entier égal à la longueur de la liste passée en argument. Appel : `len(L)`.
- La fonction `float()` convertit l'argument en flottant. Appel `float(n)`.
- La fonction `int()` convertit l'argument en un entier. Si l'argument est un flottant, la fonction `int()` renvoie la partie entière de ce nombre. Appel `int(x)`.
- La fonction `range(a,b,p)` génère la liste suivante : $[a, a+p, a+2p, \dots, a+jp, \dots, b-p]$. L'appel `range(a,b)` est équivalent à `range(a,b,1)` et l'appel `range(b)` est équivalent à `range(0,b,1)`.
- La fonction `linspace(a,b,N)` génère le tableau de flottants (type array) suivant : $[a, a+p, a+2p, \dots, a+jp, \dots, a+(N-1)p]$ avec $p = \frac{b-a}{N}$.
- Slicing : l'expression `l[i:j]` renvoie la sous-liste ou le sous-tableau comprenant les éléments d'indice i inclus à j exclu de la liste `l`.

1 Modèle à compartiments pour la propagation d'une épidémie

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 d'autres maladies comme la poliomyélite ou la rougeole. Ils ont également permis d'expliquer l'apparition d'épidémies de grippe tous les hivers. Aujourd'hui, des modèles de plus en plus complexes et puissants sont éveloppés 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.

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
```

On s'intéresse ici 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, « susceptible »), 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 :

$$\left.\begin{array}{l} \frac{d}{dt}S(t) = -rS(t)I(t) \\ \frac{d}{dt}I(t) = rS(t)I(t) - (a+b)I(t) \\ \frac{d}{dt}R(t) = aI(t) \\ \frac{d}{dt}D(t) = bI(t) \end{array}\right\} \quad (1)$$

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 (1) s'écrive sous la forme

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

2. Compléter la ligne 4 (utiliser autant de lignes que nécessaire) 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 $N = 7$ correspond 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 pas de symptômes et n'est pas contagieux. On peut prendre en compte cette phase d'incubation à l'aide du système à retard suivant :

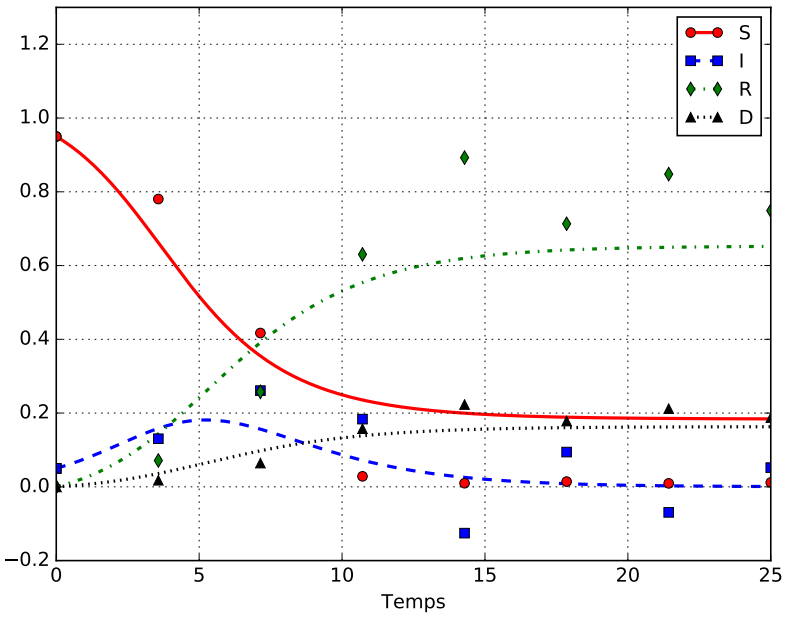


FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

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

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

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 écrit le code suivant :

```
1  def f(X,Itau):
2      """
3      Fonction definissant l'equation differentielle
4      Itau est la 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).

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.

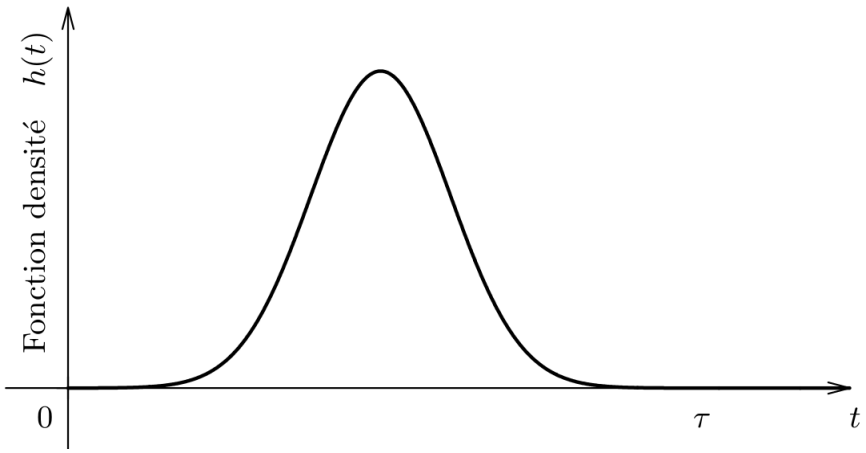


FIGURE 2 – Exemple d'une fonction de densité.

On obtient alors le système intégro-différentiel :

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

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 :

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

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).

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 Écrire en langage Python la fonction `bisect` avec les arguments adéquats déterminant à e près le zéro d'une fonction f dont on sait qu'il appartient à l'intervalle $[a, b]$.

   1.2 Montrer la terminaison de la fonction `bisect`

   1.3 Montrer la correction de la fonction `bisect`.

2. **Minimum d'une fonction convexe**

   On dit qu'une fonction f est convexe sur un intervalle I, lorsque tout segment reliant deux points de sa courbe représentative se trouve au dessus de la courbe. On admettra que si une fonction est convexe sur $I = [a, b]$, elle y admet un minimum atteint en un seul point que l'on notera μ. Notre objectif est de décrire et d'implémenter un algorithme de calcul approché de μ.

   Notations : pour x_1, x_2 tels que $a < x_1 < x_2 < b$, on note p_0, p_1, p_2 les pentes des sécantes à la courbe représentative de f aux points d'abscisses (a, x_1), (x_1, x_2) et (x_2, b).

   2.1 A l'aide d'une illustration graphique dire si chacune des conjectures suivantes vous paraît raisonnable :
   - $p_0 \leq p_1 \leq p_2 \leq 0 \Rightarrow \mu \in [x_2, b]$;
   - $p_0 \leq p_1 \leq 0 \leq p_2 \Rightarrow \mu \in [x_1, b]$;
   - $p_0 \leq 0 \leq p_1 \leq p_2 \Rightarrow \mu \in [a, x_2]$;
   - $0 \leq p_0 \leq p_1 \leq p_2 \Rightarrow \mu \in [a, x_1]$.

   Y-a-t-il a priori d'autres cas possibles ? Justifier.

   2.2 En vous inspirant de la méthode de dichotomie, écrire en langage Python la fonction `min_function` calculant de façon approchée (à ε près) le minimum de f dont on sait qu'il existe sur un intervalle $[a, b]$. Pour cela on posera :

$$\begin{cases} x_1 = m - \lambda(b-a) \\ x_2 = m + \lambda(b-a) \end{cases} \text{ avec } m = \frac{a+b}{2} \quad \text{et} \quad 0 < \lambda < 1/2$$

   2.3 Exprimer la complexité, dans le pire des cas, de la fonction `min_function` en fonction de ε, b, a et λ. Application numérique pour $\lambda = 1/3$, $\varepsilon = 1 \times 10^{-3}$ et $b - a = 1$.

3 Simulation d'une file de voitures

On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir fig.3). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

FIGURE 3 – Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10

On considère le cas d'une seule file, illustré par la figure 3. Une file de longueur n est représentée par n cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la figure 3) et sont indifférenciées.

1. **Préliminaires**

   1.1 Proposer une implémentation de la file de voitures à l'aide d'une liste de booléens. Ecrire alors l'instruction `Python` permettant de définir une liste `A` représentant la file de voiture illustrée par la figure 3.

   1.2 Soit `L` une liste représentant une file de longueur n et i un entier tel que $0 \leq i < n$. Définir en Python la fonction `occupe(L, i)` qui renvoie `True` lorsque la case d'indice i de la file est occupée par une voiture et `False` sinon.

   1.3 Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse

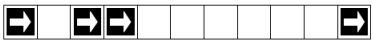
2. **Déplacement des voitures dans une file**

On identifie désormais une file de voitures à une liste. On considère les schémas de la figure 6 représentant des exemples de files. Une étape de simulation pour une file consiste à déplacer les voitures de la file, à tour de rôle, en commençant par la voiture la plus à droite, d'après les règles suivantes :

— une voiture se trouvant sur la case la plus à droite de la file sort de la file ;
— une voiture peut avancer d'une case vers la droite si elle arrive sur une case inoccupée ;
— une case libérée par une voiture devient inoccupée ;
— la case la plus à gauche peut devenir occupée ou non, selon le cas considéré.

On suppose avoir écrit en `Python` la fonction `avancer` prenant en paramètres une liste de départ, un booléen indiquant si la case la plus à gauche doit devenir occupée lors de l'étape de simulation, et renvoyant la liste obtenue par une étape de simulation.

Par exemple, l'application de cette fonction à la liste illustrée par la figure 6(a) permet d'obtenir soit la liste illustrée par la figure 6(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la figure 6(c) lorsque l'on considère qu'une voiture nouvelle est introduite.



(a) Liste initiale A

(b) $B =$ `avancer`$(A,$ `False`$)$

(c) $C =$ `avancer`$(A,$ `True`$)$

FIGURE 4 – Etape de simulation

2.1 Étant donnée `A` la liste définie à la question 1.1, que renvoie l'instruction `avancer(avancer(A,False),True)` ?

2.2 Implémenter en `Python` la fonction `avancer`.

Par la suite, on peut simuler un bouchon en bloquant une voiture, simuler un carrefour...