Concours Blanc

Durée: 2h

Informatique

Modélisation de 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 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 bases de 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 totalement indépendantes. **Les candidats sont invités à traiter chaque partie sur une copie double différente** pour ne pas être obligés de les traiter dans l'ordre. À l'inverse, à l'intérieur d'une même partie, il est conseillé de répondre aux questions dans l'ordre dans lequel elles sont posées.

Dans tout le problème, on peut utiliser une fonction traitée précédemment. On suppose que les bibliothèques `math`, `numpy` et `random` ont été importées via les commandes

```
1 from math import *
2 import numpy as np
3 import random as rd
```

L'usage de l'ordinateur ou de la calculatrice est interdit.

Partie I

Tri et bases de données

Dans le but ultérieur de réaliser des études statistiques, on souhaite se doter d'une fonction tri. On se donne la fonction tri suivante, écrite en Python :

```
1  def tri(L):
2      n = len(L)
3      for i in range(1,n):
4          j = i
5          x = L[i]
6          while 0 < j and x < L[j-1]:
7              L[j] = L[j-1]
8              j = j-1
9          L[j] = x
10     return L
```

Q1. Lors de l'appel `tri(L)` lorsque `L` est la liste `[5, 2, 3, 1, 4]`, donner le contenu de la liste `L` à la fin de chaque itération de la boucle `for`.

Q2. Soit `L` une liste non vide d'entiers ou de flottants. Montrer que « la liste `L[0:i+1]` (avec la convention Python) est triée par ordre croissant à l'issue de l'itération `i` » est un invariant de boucle. En déduire que `tri(L)` trie la liste `L`.

Q3. Évaluer la complexité dans le meilleur et dans le pire des cas de l'appel `tri(L)` en fonction du nombre n d'éléments de `L`.

On souhaite, partant d'une liste constituée de couples (chaîne, entier), trier la liste par ordre croissant de l'entier associé suivant le fonctionnement suivant :

```
>>> L = [['Bresil', 76], ['Kenya', 26017], ['Ouganda', 8431]]
>>> tri_chaine(L)
>>> L
[['Bresil', 76], ['Ouganda', 8431], ['Kenya', 26017]]
```

Q4. Adaptez le code fourni pour la fonction `tri` afin d'écrire en Python une fonction `tri_chaine` réalisant cette opération.

Pour suivre la propagation des épidémies, de nombreuses données sont recueillies par les institutions internationales comme l'O.M.S. Par exemple, pour le paludisme, on dispose de deux tables :

– la table `palu` recense le nombre de nouveaux cas confirmés et le nombre de décès liés au paludisme ; certaines lignes de cette table sont données en exemple (on précise que iso est un identifiant unique pour chaque pays) :

nom	iso	annee	cas	deces
Bresil	BR	2009	309 316	85
Bresil	BR	2010	334 667	76
Kenya	KE	2010	898 531	26 017
Mali	ML	2011	307 035	2 128
Ouganda	UG	2010	1 581 160	8 431
...				

– la table `demographie` recense la population totale de chaque pays ; certaines lignes de cette table sont données en exemple :

pays	periode	pop
BR	2009	193 020 000
BR	2010	194 946 000
KE	2010	40 909 000
ML	2011	14 417 000
UG	2010	33 987 000
...		

Q5. Au vu des données présentées dans la table `palu`, parmi les attributs `nom`, `iso` et `annee`, quels attributs peuvent servir de clé primaire ? Un couple d'attributs pourrait-il servir de clé primaire ? (on considère qu'une clé primaire peut posséder plusieurs attributs). Si oui, en préciser un.

Q6. Écrire une requête en langage SQL qui récupère depuis la table `palu` toutes les données de l'année 2010 qui correspondent à des pays où le nombre de décès dus au paludisme est supérieur ou égal à 1 000.

On appelle taux d'incidence d'une épidémie le rapport du nombre de nouveaux cas pendant une période donnée sur la taille de la population-cible pendant la même période. Il s'exprime généralement en « nombre de nouveaux cas pour 100 000 personnes par année ». Il s'agit d'un des critères les plus importants pour évaluer la fréquence et la vitesse d'apparition d'une épidémie.

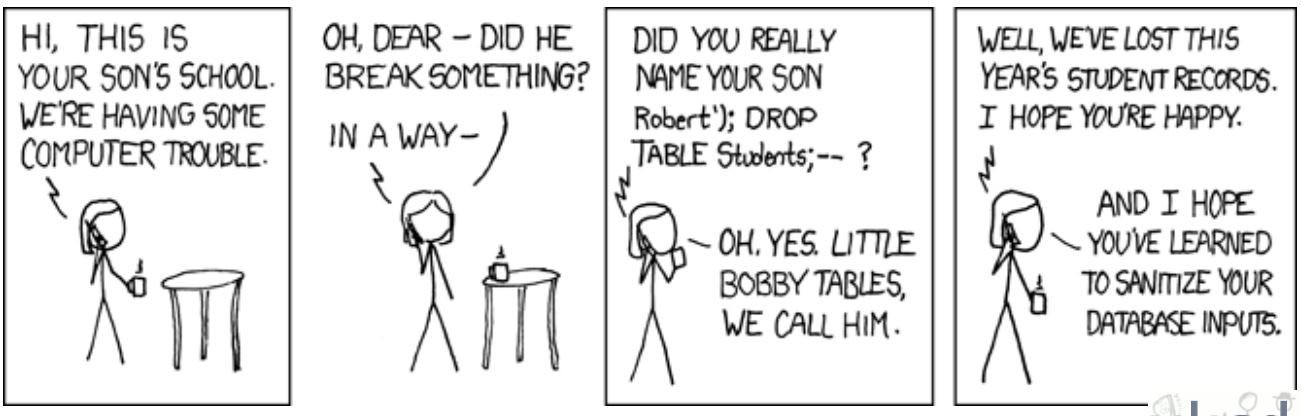
Q7. Écrire une requête en langage SQL qui détermine le taux d'incidence du paludisme en 2011 pour les différents pays de la table `palu`.

Q8. Écrire une requête en langage SQL permettant de déterminer le nom du pays ayant eu le deuxième[1] plus grand nombre de nouveaux cas de paludisme en 2010. Afin d'en simplifier l'écriture et si jamais vous avez besoin d'utiliser une sous-requête, on supposera que le langage SQL permet des construction du type

```
R1 = (SELECT nom,annee,cas FROM palu)
SELECT nom,annee FROM R1
```

On suppose qu'on a récupéré via une requête SQL la liste des couples (`nom`, `deces`) pour l'année 2010 et que cette liste a été stockée dans la variable Python `deces2010`.

Q9. Quelle instruction peut-on écrire en Python pour trier la liste `deces2010` par ordre croissant du nombre de décès dus au paludisme en 2010 ?



`Her daughter is named Help I'm trapped in a driver's license factory.`

[1] On pourra supposer qu'il n'y a pas de pays ex æquo pour les nombres de cas.

Partie II

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), infectés (I), rétablis (R, ils sont immunisés) et décédés (D). 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 $\mathrm{S}(t)$, $\mathrm{I}(t)$, $\mathrm{R}(t)$ et $\mathrm{D}(t)$ la fraction de la population appartenant à chacune des quatre catégories à l'instant t, on obtient le système :

$$\begin{cases} \dfrac{\mathrm{dS}}{\mathrm{d}t} = -r\,\mathrm{S}(t)\,\mathrm{I}(t) \\ \dfrac{\mathrm{dI}}{\mathrm{d}t} = r\,\mathrm{S}(t)\,\mathrm{I}(t) - (a+b)\,\mathrm{I}(t) \\ \dfrac{\mathrm{dR}}{\mathrm{d}t} = a\,\mathrm{I}(t) \\ \dfrac{\mathrm{dD}}{\mathrm{d}t} = b\,\mathrm{I}(t) \end{cases} \quad (\mathbf{1})$$

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 $\mathrm{S}(0) = 0{,}95$, $\mathrm{I}(0) = 0{,}05$ et $\mathrm{R}(0) = \mathrm{D}(0) = 0$.

Q10. 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

$$\frac{\mathrm{dX}}{\mathrm{d}t} = f(\mathrm{X})$$

Q11. Modifier 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) afin de définir correctement le système différentiel que l'on compte résoudre.

```
1  def f(X):
2      """ Fonction définissant l'équation différentielle. """
3      global r,a,b      # Variables définies hors de la fonction
4      return [1,2,3,4]  # Ligne à modifier...
5
6  # Paramètres
7  tmax = 25.0
8  r = 1.0
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
```

```
tt= [t]
XX= [X]

# Méthode d'Euler
for i in range(N):
    t = t + dt
    X = X + np.array(f(X)) * dt
    tt.append(t)
    XX.append(X)
```

Q12. La figure 1 représente les quatre catégories en fonction du temps obtenues en effectuant deux simulations : la première avec N = 7 correspond aux points (cercles, carrés, losanges, triangles) 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 ? Les deux simulations sont-elles cohérentes avec la réalité ?

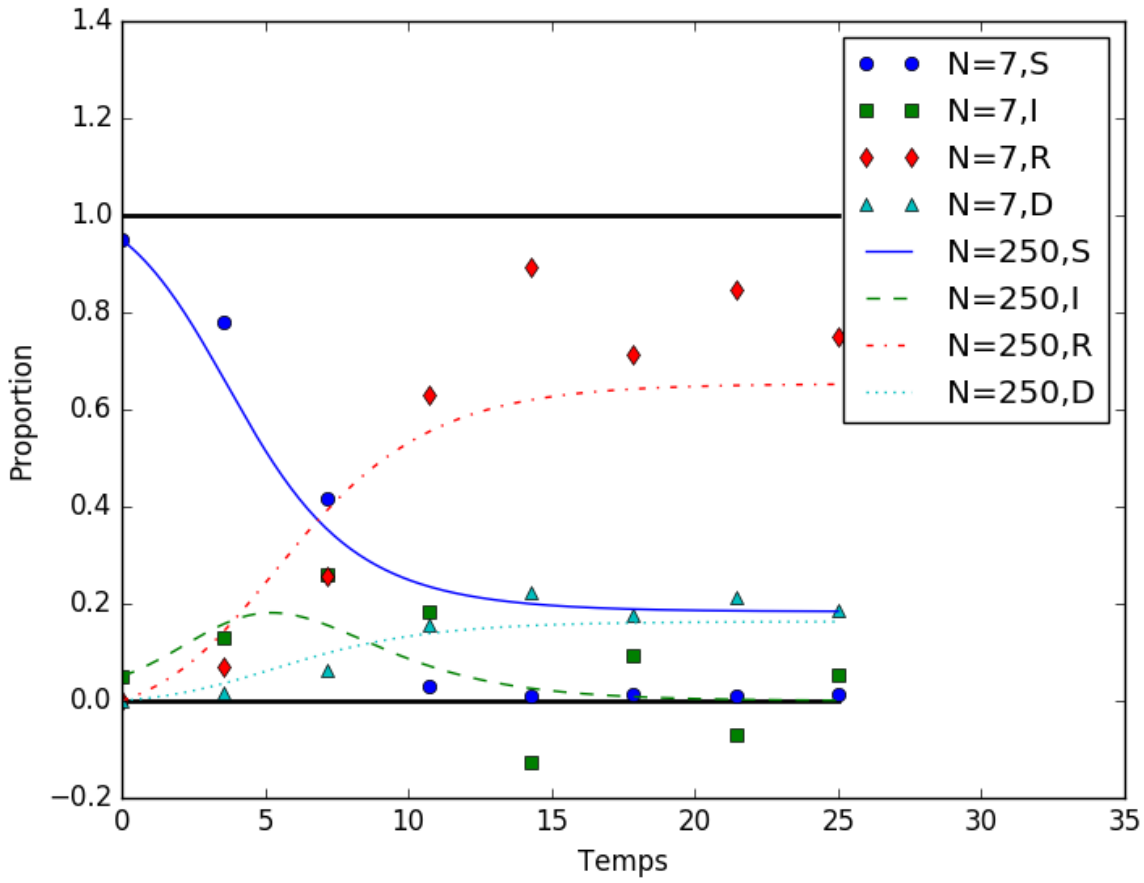


Fig. 1: Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour N = 7 (points individuels) et N = 250 (courbes) avec les mêmes conditions initiales. On a tracé en traits gras horizontaux les lignes délimitant les proportions de 0% et de 100%.

Partie III

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 cinq états suivants : bord, saine, infectée, rétablie, décédée. On choisit de représenter ces cinq états par les entiers :

-1 (bord), 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 cinq cases voisines et trois pour une case d'un coin). Les règles de transition sont les suivantes :

- une case du bord reste une case du bord (elle ne fait pas à proprement parler partie de la population);
- 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.

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

Q13. On a écrit en Python la fonction `grille(n)` suivante

```
1  def grille(n):
2      M = []
3      for i in range(n+2):
4          L = []
5          for j in range(n+2):
6              if i == 0 or i == n+1 or j == 0 or j == n+1:
7                  L.append(-1)
8              else:
9                  L.append(0)
10         M.append(L)
11     return M
```

Décrire ce que retourne cette fonction. Faire un dessin pour la cas $n = 3$. Combien de cases (en fonction de n) correspondent à la population lors de l'appel à `grille(n)` ?

On pourra dans la question suivante utiliser la fonction `rd.randrange(n)` de la bibliothèque `random`[2] qui, pour un entier positif n, renvoie un entier choisi aléatoirement entre 0 et $n-1$ inclus.

Q14. Écrire en Python une fonction `init(n)` qui construit une grille `G` de taille $(n+2) \times (n+2)$ vérifiant les spécifications demandées. En particulier, les bords sont des cases « bord », l'intérieur est constitué uniquement de cases saines hormis une, choisie aléatoirement, qui est infectée. La fonction doit renvoyer la grille `G`.

Q15. Écrire en Python une fonction `compte(G)` qui prend en argument une grille `G` et renvoie la liste `[n0,n1,n2,n3]` formée des nombres de cases dans chacun des quatre états intéressants[3] (S, I, R et D).

[2] Qui a été aliasée à `rd` dans le préambule.

[3] Attention à ne pas compter les bords.

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      if G[i][j] == -1: return False
3      prod = 1
4      for k in range(i-1,i+2):
5          for l in range(j-1,j+2):
6              if k!=i or l!=j:
7                  prod = prod * (G[k][l] - 1)
8      return (prod == 0)
```

Q16. Quel est le type du résultat renvoyé par la fonction `est_exposee` ?

Q17. Expliquer son mode de fonctionnement en quelques mots, notamment la notion de « voisinage » utilisée (à l'aide d'un petit dessin par exemple).

Q18. Le cas particulier traité à la ligne 2 est-il nécessaire si le programmeur utilise cette fonction correctement ? (Justifier...)

On définit 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: return 1
4      else:     return 0
```

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

```
random.random()
      Return the next random floating point number in the range [0.0, 1.0).
```

On rappelle (à toutes fins utiles) que pour correctement copier une liste simple, on peut utiliser la syntaxe `new_L=old_L[:]`. Pour une liste de liste, il faut utiliser `new_L=copy.deepcopy(old_L)` après import du module `copy` (fonctionne aussi pour une liste simple).

Q19. 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)`.

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.

Q20. Écrire en Python une fonction `simulation(n,p1,p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ (en population) pour les probabilités p_1 et p_2, et renvoie la liste `[x0,x1,x2,x3]` formée des proportions de cases dans chacun des quatre états intéressants à la fin de la simulation (une simulation s'arrête lorsque la grille n'évolue plus).

Q21. Quelle est la valeur de la proportion des cases infectées `x1` à la fin d'une simulation ? Quelle relation vérifient `x0`, `x1`, `x2` et `x3` ? Comment obtenir à l'aide des valeurs de `x0`, `x1`, `x2` et `x3` la valeur `x_atteinte` de la proportion des cases qui ont été atteintes par la maladie pendant une simulation ?

On fixe `p1` à 0,5 et on calcule la moyenne des résultats de plusieurs simulations pour différentes valeurs de `p2`. On obtient la courbe de la figure 2.

Q22. On appelle seuil critique de pandémie la valeur de `p2` à partir de laquelle plus de la moitié de la population a été atteinte par la maladie à la fin de la simulation. On suppose que les valeurs de `p2` et `x_atteinte` utilisées pour tracer la courbe de la figure 2 ont été stockées dans deux listes de même longueur `Lp2` et `Lxa`. Écrire en Python une fonction `seuil(Lp2,Lxa)` qui détermine par dichotomie un encadrement `[p2cmin, p2cmax]` du seuil critique de pandémie avec la plus grande précision possible. On supposera que la liste `Lp2` croit de 0 à 1 et que la liste `Lxa` des valeurs correspondantes est croissante.

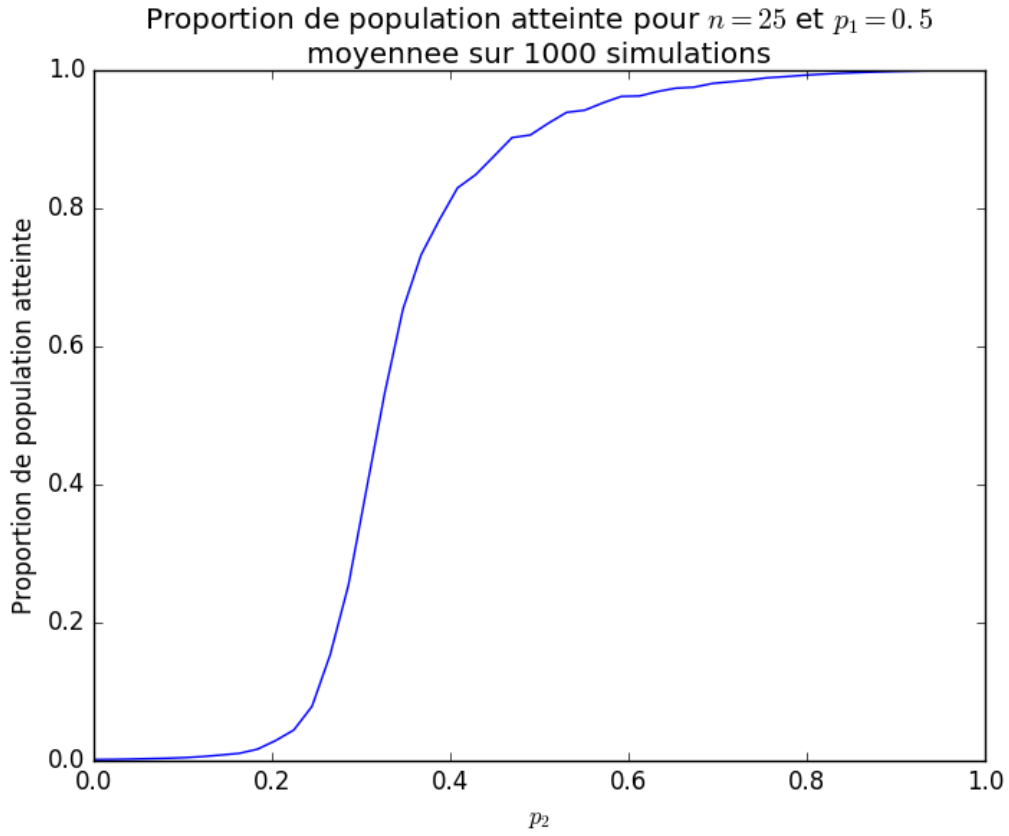


Fig. 2: Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2 pour une grille de taille 25×25 en population avec $p_1 = 0{,}5$ et des résultats moyennés sur 1000 simulations.

Fin de l'épreuve