A 2016 - INFO.

École des PONTS ParisTech,
ISAE-SUPAERO, ENSTA ParisTech,
TÉLÉCOM ParisTech, MINES ParisTech,
MINES Saint-Étienne, MINES Nancy,
TÉLÉCOM Bretagne, ENSAE ParisTech (Filière MP).

CONCOURS 2016

ÉPREUVE D'INFORMATIQUE
TOUTES LES FILIÈRES

(Durée de l'épreuve : 1 h 30)
L'usage de l'ordinateur ou de la calculatrice est interdit.

Sujet mis à la disposition des concours :
Concours Commun TPE/EIVP, Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle international).

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE

L'énoncé de cette épreuve comporte 9 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

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 indépendantes.*

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
```

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
```

❑ **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`. Citer un algorithme de tri plus efficace dans le pire des cas. Quelle en est la complexité dans le meilleur et dans le pire des cas ?

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** – 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 plus grand nombre de nouveaux cas de paludisme en 2010 (on pourra supposer qu'il n'y a pas de pays *ex æquo* pour les nombres de cas).

On considère la requête R qui s'écrit dans le langage de l'algèbre relationnelle :

$$R = \pi_{\texttt{nom},\texttt{deces}}\left(\sigma_{\texttt{annee}=2010}(\texttt{palu})\right)$$

On suppose que le résultat de cette requête a été converti en une liste `Python` stockée dans la variable `deces2010` et constituée de couples (chaîne, entier).

❑ **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 ?

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, "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 :

$$\left.\begin{aligned} \frac{d}{dt}S(t) &= -r\,S(t)I(t) \\ \frac{d}{dt}I(t) &= r\,S(t)I(t) - (a+b)\,I(t) \\ \frac{d}{dt}R(t) &= a\,I(t) \\ \frac{d}{dt}D(t) &= b\,I(t) \end{aligned}\right\} \tag{1}$$

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 $R(0) = D(0) = 0$.

❑ **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

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

❑ **Q11** – Compléter 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).

```
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)
```

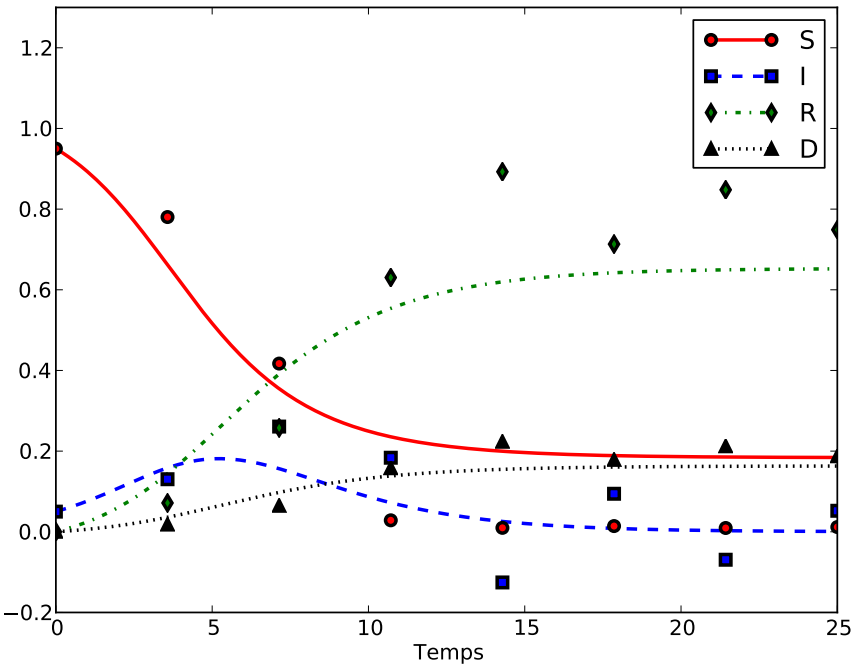

FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

❑ **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 (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 :

$$\begin{cases} \dfrac{d}{dt}S(t) = -r\,S(t)I(t-\tau) \\ \dfrac{d}{dt}I(t) = r\,S(t)I(t-\tau) - (a+b)\,I(t) \\ \dfrac{d}{dt}R(t) = a\,I(t) \\ \dfrac{d}{dt}D(t) = b\,I(t) \end{cases}$$

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

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 a 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)
```

❑ **Q13** – 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. On obtient alors le système intégro-différentiel :

$$\begin{cases} \dfrac{d}{dt}S(t) = -r\,S(t)\displaystyle\int_0^\tau I(t-s)h(s)\,ds \\ \dfrac{d}{dt}I(t) = r\,S(t)\displaystyle\int_0^\tau I(t-s)h(s)\,ds - (a+b)\,I(t) \\ \dfrac{d}{dt}R(t) = a\,I(t) \\ \dfrac{d}{dt}D(t) = b\,I(t) \end{cases}$$

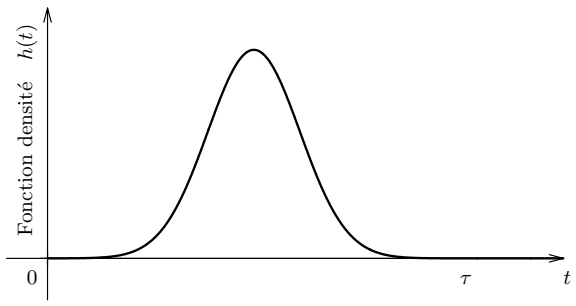

FIGURE 2 – Exemple d'une fonction de densité.

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 :

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

❑ **Q14** – 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).

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

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 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.

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

❑ **Q15** – On a écrit en `Python` la fonction `grille(n)` suivante

```
def grille(n) :
    M=[ ]
    for i in range(n) :
        L=[ ]
        for j in range(n): L.append(0)
        M.append(L)
    return M
```

Décrire ce que retourne cette fonction.

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

❑ **Q16** – Écrire en `Python` une fonction `init(n)` qui construit une grille G de taille $n \times n$ ne contenant que des cases saines, choisit aléatoirement une des cases et la transforme en case infectée, et enfin renvoie G.

❑ **Q17** – Écrire en `Python` une fonction `compte(G)` qui a pour argument une grille `G` et renvoie la liste `[n0, n1, n2, n3]` formée des nombres de cases dans chacun des quatre états.

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      n = len(G)
3      if i == 0 and j == 0:
4          return (G[0][1]-1)*(G[1][1]-1)*(G[1][0]-1) == 0
5      elif i == 0 and j == n-1:
6          return (G[0][n-2]-1)*(G[1][n-2]-1)*(G[1][n-1]-1) == 0
7      elif i == n-1 and j == 0:
8          return (G[n-1][1]-1)*(G[n-2][1]-1)*(G[n-2][0]-1) == 0
9      elif i == n-1 and j == n-1:
10         return (G[n-1][n-2]-1)*(G[n-2][n-2]-1)*(G[n-2][n-1]-1) == 0
11     elif i == 0:
12         # a completer
13     elif i == n-1:
14         return (G[n-1][j-1]-1)*(G[n-2][j-1]-1)*(G[n-2][j]-1)*(G[n-2][j+1]-1)*(G[n-1][j+1]-1) == 0
15     elif j == 0:
16         return (G[i-1][0]-1)*(G[i-1][1]-1)*(G[i][1]-1)*(G[i+1][1]-1)*(G[i+1][0]-1) == 0
17     elif j == n-1:
18         return (G[i-1][n-1]-1)*(G[i-1][n-2]-1)*(G[i][n-2]-1)*(G[i+1][n-2]-1)*(G[i+1][n-1]-1) == 0
19     else:
20         # a completer
```

❑ **Q18** – Quel est le type du résultat renvoyé par la fonction `est_exposee` ?

❑ **Q19** – Compléter les lignes 12 et 20 de la fonction `est_exposee`.

❑ **Q20** – 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)` 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`$)$.

```
def bernoulli(p):
    x = rd.random()
    if x <= p:
        return 1
    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).
```

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.

❑ **Q21** – Écrire en `Python` une fonction `simulation(n, p1, p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ pour les probabilités `p1` et `p2`, et renvoie la liste `[x0, x1, x2, x3]` formée des proportions de cases dans chacun des quatre états à la fin de la simulation (une simulation s'arrête lorsque la grille n'évolue plus).

❑ **Q22** – 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 ?

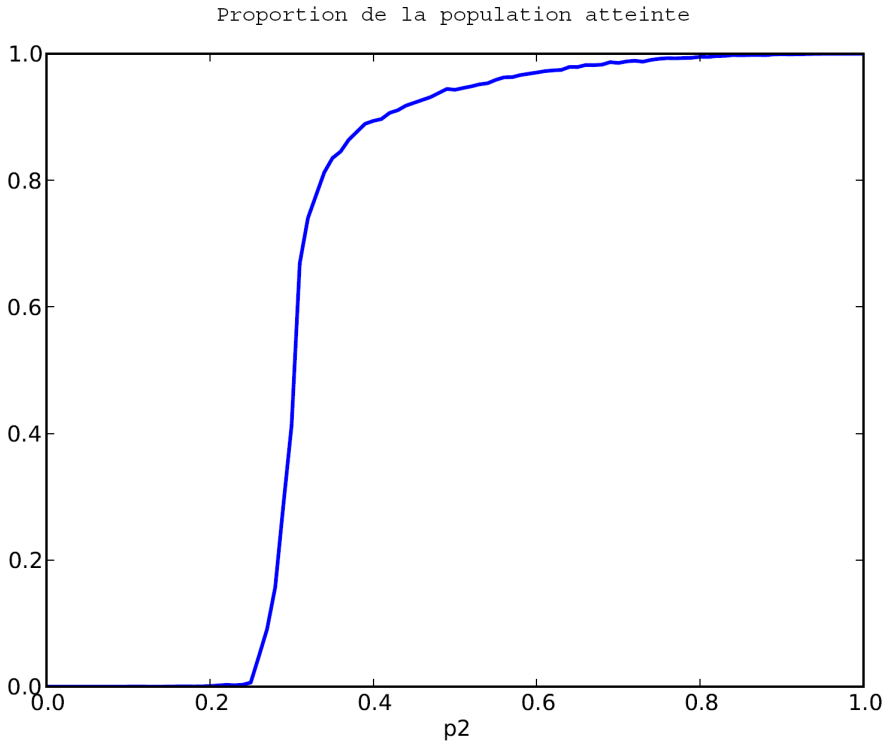

FIGURE 3 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2.

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 3.

❑ **Q23** – 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 3 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` croît de 0 à 1 et que la liste `Lxa` des valeurs correspondantes est croissante.

Pour étudier l'effet d'une campagne de vaccination, on immunise au hasard à l'instant initial une fraction `q` de la population. On a écrit la fonction `init_vac(n, q)`.

```
1  def init_vac(n, q):
2      G = init(n)
3      nvac = int(q * n**2)
4      k = 0
5      while k < nvac:
6          i = rd.randrange(n)
7          j = rd.randrange(n)
8          if G[i][j] == 0:
9              G[i][j] = 2
10             k += 1
11     return G
```

❑ **Q24** – Peut-on supprimer le test en ligne 8 ?

❑ **Q25** – Que renvoie l'appel `init_vac(5, 0.2)` ?

Fin de l'épreuve.

4.1-Informatique pour tous

PRÉSENTATION DU SUJET

L'épreuve d'informatique commune portait sur le traitement informatique de la modélisation d'une propagation d'épidémie. L'épreuve faisait appel à des notions variées du programme des deux années de classe préparatoires : tris, invariants de boucle, requêtes en langage SQL, algorithmique sur des tableaux, résolution numérique d'un système différentiel. Malgré un temps d'épreuve assez court, cette épreuve nous a paru de nature à garantir un classement efficace des candidats.

Quelques excellentes copies témoignent d'un travail approfondi sur le programme, doublé d'une grande rapidité et d'une capacité à écrire des programmes clairs, concis, et syntaxiquement irréprochables. A l'inverse, quelques copies témoignent d'une méconnaissance flagrante des règles de bases de cette discipline, et parfois aussi d'un niveau de raisonnement scientifique remarquablement faible à ce niveau de formation.

Nous souhaitons rappeler aux candidats quelques conditions nécessaires pour espérer réussir cette épreuve :

- La maîtrise de la syntaxe de base des langages python et SQL est absolument indispensable. Le respect de l'indentation est capital, et la lisibilité de l'écriture est appréciée : il est rappelé que le doute ne bénéficie pas souvent au candidat.

- Les candidats sont invités à réfléchir sur les spécificités de chaque type informatique : une liste (ou une liste de listes) n'a pas les même propriétés qu'un tableau numpy, par exemple.

- Les questions qualitatives, où un court paragraphe est attendu (par exemple la question 12) doivent être traitées avec soin : la concision nécessaire n'excuse pas l'imprécision parfois très gênante des termes utilisés, ou du raisonnement.

Voici quelques commentaires question par question.

Q1 : Bien traitée par une grande majorité de candidats.

Q2 : La notion d'invariant de boucle a été souvent maltraitée. Un certain nombre de candidats montre son ignorance presque complète de cette notion par des raisonnements tels que «si la liste est triée au début, alors elle est triée à la fin et donc c'est un invariant». Parmi les candidats qui ont compris l'intérêt du raisonnement par récurrence, certain oublient l'importance de l'initialisation, ou bien d'utiliser l'invariant pour prouver que la liste est effectivement triée à la fin de l'exécution de la fonction.

Q3 : Cette question a donné lieu à un florilège de résultats particulièrement exotiques. Le cours sur les tri semble être passé de manière approximative chez beaucoup de candidats. Les complexités s'échelonnent de $O(1)$ à $O(n!n^n)$, avec beaucoup de complexité en $O(n!)$: il est dommage qu'un très grand nombre de candidats montrent par là qu'ils n'ont pas saisi la portée pratique de la notion de complexité pour l'exécution des programmes...

Les tris cités sont nombreux - le tri par insertion étant parfois proposé comme plus efficace que le tri de l'énoncé - et parmi les candidats qui citent à raison le tri fusion, beaucoup prétendent qu'ils possède une complexité linéaire dans le meilleur des cas et quasi-linéaire dans le pire des cas.

Q4 : Si de nombreux candidats ont bien pensé à adapter le programme fourni par l'énoncé, il est dommage que certains n'aient réalisé le tri que sur le second élément de la liste en oubliant le premier !

Q5 : La notion de clef primaire est très mal maîtrisée. On a ainsi pu lire des phrases très étonnantes, comme « l'attribut iso peut servir de clef primaire, et de même le couple iso/année peut servir de clef primaire car elle renvoie à une seule ligne du tableau ». Un travail supplémentaire sur cette notion importante semble nécessaire.

Q6 : Une requête commençant par **FROM palu IMPORT...** laisse dubitatif sur la qualité du travail de préparation des candidats sur le langage SQL. La majorité des candidats a cependant traité correctement cette question. Il est aussi rappelé à certains candidats créatifs que l'épreuve d'informatique n'est pas une épreuve d'anglais, et que bricoler une instruction vague avec des pseudo-mots d'anglais n'est pas suffisant pour écrire une requête en SQL.

Q7 : Cette requête comportait plusieurs difficultés. Tous les candidats n'ont pas perçu la nécessité d'une jointure symétrique, qui fut par ailleurs souvent mal écrite. Il fallait noter que la condition de jointure faisait intervenir deux attributs pour chaque table.

Q8 : Question plus difficile, que des candidats malins ont traité astucieusement avec LIMIT et OFFSET, profitant d'une certaines ambiguïté du programme sur ce qui était exigible. On pouvait aussi s'en sortir avec des sous-requêtes.

Q9 : Question souvent bien traitée.

Q10 : Nous avons constaté sur cette question un grand nombre de réponses comme $f(X)=S(t) + I(t) + R(t) + D(t)$, qui témoignent d'une incompréhension du fonctionnement des systèmes différentiels. Certains candidats ont tenu à exprimer f sous forme d'une matrice carrée, ce qui était faux ici, le système différentiel n'étant pas linéaire.

Q11 : Question bien traitée par ceux qui ont réussi la question précédente, mais il ne fallait pas oublier de renvoyer un tableau numpy et pas une liste.

Q12 : Cette question qualitative pourtant simple a été assez mal traitée. On passe sur les candidats qui après un raisonnement parfois très torturé, arrivent à la conclusion que la simulation avec N=7 nécessite plus de temps... Le rôle du pas de discrétisation pour la précision de la méthode d'Euler était attendu, et n'est pas apparu clairement dans la majorité des copies. A l'inverse, on a pu lire des horreurs comme «la simulation est discrète pour N=7 alors que pour N=250 elle est continue».

Q13 et **Q14** : Questions plus difficiles qui nous ont souvent permis de repérer les meilleures copies.

Q15 : Question simple souvent bien traitée.

Q16 : Tous les candidats ne maîtrisent pas le double indice pour accéder à un élément d'un tableau (**G[i][j]**) ou le confondent avec la syntaxe adaptée aux tableau numpy (**G[i,j]**). Par ailleurs, quelques candidats ont modifié un élément situé obligatoirement sur la diagonale de la grille.

Q17 : Question assez simple, mais qui a souvent donné lieu à une structure lourde (**if... else... elif...**) alors qu'une possibilité beaucoup plus légère existait.

Q18 : Attention à ne pas confondre le type d'une variable (un booléen) avec les valeurs possibles de cette variable (**True** ou **False**).

Q19 : Question souvent bien traitée.

Q20 : Une erreur fréquente des candidats est de n'avoir pas perçu la nécessité de créer une nouvelle grille dès le début de la fonction, et de ne pas modifier **G** avant la fin de celle-ci: en effet, les valeurs de **G** sont nécessaires à la détermination de l'évolution, et on ne peut pas à la fois effectuer des tests utilisant les valeurs de **G** et modifier **G**.

Q21 : Le caractère aléatoire du résultat renvoyé par la fonction suivant rendait impossible la syntaxe :

```
while G!=suivant(G):
    G=suivant(G)
```

Pour le comptage, on a parfois assisté à des appels de fonction très maladroits, comme **n0=compte(G)[0]**, **n1=compte(G)[1]**, etc. Les candidats sont invités à réfléchir à l'efficacité algorithmique de leurs programmes pour éviter des appels multiples inutiles.

Q23 : Cette question nécessitait une maîtrise satisfaisante de l'algorithme de recherche par dichotomie. Elle a été peu traitée, par manque de temps.

Q24/Q25 : Certains candidats ayant perçu que ces questions étaient faciles mais manquant de temps ont proposé des réponses non justifiées (« Non c'est impossible » par exemple). Inutile de préciser que ces réponses n'ont valu aucun point à leurs auteurs.

Nous terminons ce rapport par quelques perles que nous avons choisi de distinguer particulièrement cette année:

- Le prix de la complexité la plus remarquable :

« Dans le pire des cas, la complexité est en $O(1/n^n)$ »

- Le prix de l'écologie :

« Un tri plus efficace dans le pire des cas est le tri sélectif »

- Le prix de la maîtrise de programme :

« pour trier une liste, il est plus efficace d'utiliser le pivot de Gauss »

- Le prix de la piété informatique :

« Cette fonction renvoie n^2 individus saints »

Nous souhaitons une bonne préparation et bon courage aux futurs candidats !