Informatique – D.S. 2 *(3 heures)*

Le sujet se compose d'un problème (à traiter en premier) et d'un exercice « bonus »

Problème – *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 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 de 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
```

1. 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.

2. 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`.

3. Évaluer la complexité dans le meilleur des cas 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]]
```

4. Écrire en Python une fonction `tri_chaine` réalisant cette opération *(on pourra se contenter d'une complexité quadratique).*

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
...		

5. 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.

6. 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.

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

8. 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 :

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

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).

9. 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 :

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

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

10. 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

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

11. 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 définissant l'équation différentielle"""
3      global r, a, b
4      # à compléter
5
6  # Paramètres
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 # Méthode 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)
```

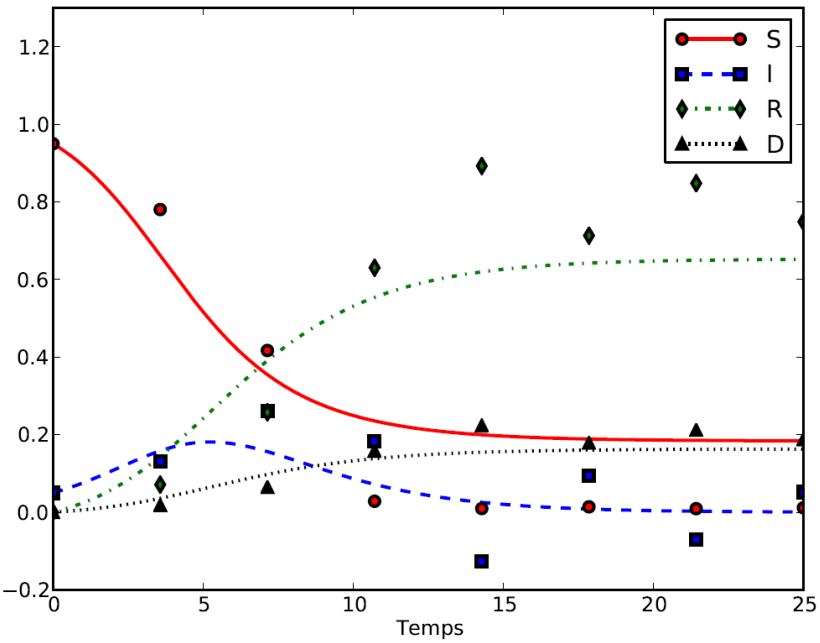

FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

12. 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 :

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

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      """Fonction définissant l'équation differentielle
3         Itau est la valeur de I(t-p*dt)"""
4      global r, a, b
5      # à compléter
6
7  # Paramètres
8  r = 1.
9  a = 0.4
10 b = 0.1
11 X0 = np.array([0.95, 0.05, 0., 0.])
12
```

```
13 tmax = 25.
14 N = 250
15 dt = tmax/N
16 p = 50
17
18 t = 0.
19 X = X0
20 tt = [t]
21 XX = [X]
22
23 # Méthode d'Euler
24 for i in range(N):
25     t = t+dt
26     # à compléter
27     tt.append(t)
28     XX.append(X)
```

13. Compléter les lignes 5 et 26 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.

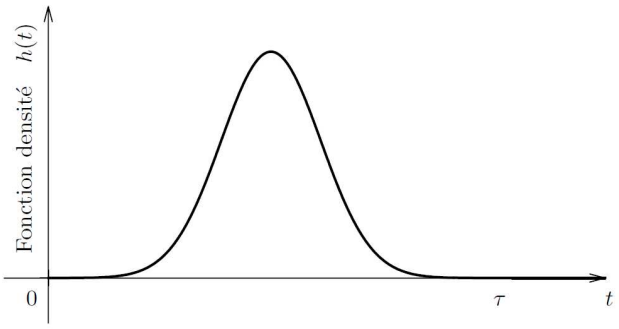

FIGURE 2 – Exemple d'une fonction de densité

On obtient alors le système intégro-différentiel :

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

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 :

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

14. 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 5 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.

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

```
1 def grille(n):
2     M = []
3     for i in range(n):
4         L = []
5         for j in range(n): L.append(0)
6         M.append(L)
7     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.

16. 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, la transforme en case infectée et enfin renvoie `G`.

17. 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.

18. Écrire en Python une fonction `zone_infectee(G, L)` qui a pour arguments une grille `G` et une liste `L` (les éléments de `L` étant des listes de la forme `[i,j]`, où `i` et `j` sont dans $[\![0, n-1]\!]$ et `G` est de taille $n \times n$) et renvoie `True` si et seulement si l'une au moins des listes `[i,j]` présentes dans `L` correspond aux coordonnées d'une case infectée (valeur 1) dans `G`.

On écrit alors en Python la fonction `est_exposee(G, i, j)` de la page suivante.

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

19. Compléter les lignes 12 et 20 de la fonction `est_exposee`.

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

20. 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.

21. 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).

Justifier la terminaison de la simulation et donner un majorant du nombre d'étapes.

22. 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 3.

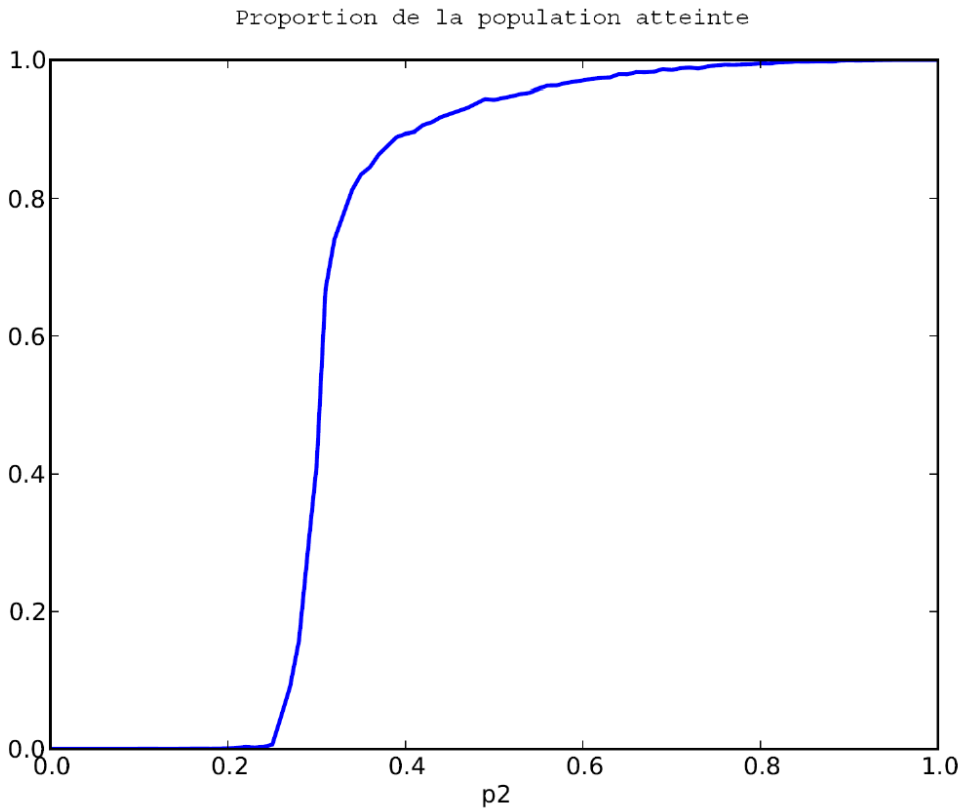

FIGURE 3 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2.

23. 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 `Lp2` et `Lxa` de même longueur. 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
```

24. Peut-on supprimer le test en ligne 8 ?

25. Que renvoie l'appel `init_vac(5, 0.2)` ?

Exercice bonus – *Le compte est bon*

La règle du jeu de ce fameux jeu télévisé est simple : on donne une liste L_0 de nombres de $\mathbb{N}^*$ ainsi qu'un entier naturel non nul N ; le but du jeu est de trouver une succession d'opérations arithmétiques (additions, soustractions, multiplications, divisions) permettant d'obtenir N en utilisant uniquement les nombres de L_0 et les résultats intermédiaires, qui doivent tous être dans $\mathbb{N}^*$; chaque élément de L_0 et chaque résultat intermédiaire peut être utilisé au plus une fois.

Dans la version officielle du jeu, la liste L_0 initiale comporte six valeurs de l'ensemble

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\},$$

chaque valeur pouvant apparaître plusieurs fois dans L_0, et $N \in [\![100, 999]\!]$.

Par exemple, si $L_0 = [10, 2, 2, 100, 4, 6]$ et $N = 667$, une solution possible est :

$$\boxed{4} * \boxed{10} = 40, \quad 40 * \boxed{100} = 4\,000, \quad 4\,000 + \boxed{2} = 4\,002, \quad 4\,002 \;/\; \boxed{6} = 667.$$

Les éléments de L_0 ayant été utilisés sont encadrés, les autres valeurs sont les résultats intermédiaires. On aurait pu utiliser 2 une seconde fois puisqu'il était en deux exemplaires dans L_0.

À propos de la division (diabolique) qui a servi dans l'exemple ci-dessus, on rappelle que, dans Python, si `a` est un entier et `b` un entier non nul, `a//b` et `a%b` donnent respectivement le quotient et le reste de la division euclidienne de `a` par `b`.

Le but de l'exercice est de programmer une recherche récursive exhaustive de toutes les solutions (et puisque la recherche est exhaustive, le programme détectera les cas où il n'y a aucune solution !).

On propose d'écrire un programme de la forme suivante :

```
1 def compte(L0, N):
2     def aux(L, calc):
3         # à compléter
4     solutions=[]
5     aux(L0, '')
6     return solutions
```

Le programme principal (trivial) initialise la liste `solutions` (destinée à contenir les solutions mémorisées durant l'exécution de la fonction `aux`), lance la fonction auxiliaire récursive `aux`, puis renvoie les solutions. Noter que l'entier `N` et la liste `solutions` sont accessibles à chaque instant de l'exécution de la fonction `aux`. Ainsi ladite fonction `aux` prend seulement deux paramètres :

- `L`, liste des entiers disponibles pour les calculs ;
- `calc`, chaîne de caractères contenant les calculs intermédiaires ayant permis de remplacer la liste initiale par `L`.

Le cahier des charges de la fonction `aux` est le suivant :

- si `N` est dans la liste `L`, on ajoute la chaîne `calc` à la liste `solutions`
- sinon, on effectue une double boucle pour parcourir tous les couples (i, j) tels que $0 \leq i < j < len(\texttt{L})$ et, pour chacun de ces couples :
  - on crée une nouvelle liste `L2`, contenant les éléments de `L` d'indices distincts de i et de j. Ainsi `L2` a deux éléments de moins que `L` ;
  - puis on lance (au plus) quatre appels récursifs de la fonction `aux`, pour les quatres opérations arithmétiques autorisées. Par exemple, **pour l'addition**, on appelle

    `aux(L2+[a+b], calc+' '+str(a)+'+'+str(b)+'='+str(a+b))`

    où l'on a ajouté à la liste `L2` la valeur $a + b$ (par concaténation) et où l'on a ajouté aux calculs stockés dans la chaîne `calc` le nouveau calcul intermédiaire qui a permis d'ajouter $a + b$ à la liste des entiers disponibles pour les calculs ultérieurs (les calculs successifs sont séparés par des espaces). On procède enfin de même avec les trois autres opérations, en évitant les calculs inutiles ou interdits...

Noter que `aux` "fonctionne" en fait comme une procédure (sans `return`), son action se traduisant *in fine* par la modification de la liste `solutions`.

Question 1 : programmer la fonction `aux`, justifier sa terminaison et sa correction.

Question 2 : proposer un programme qui donne les solutions permettant d'obtenir un entier $\widetilde{N}$ le plus proche possible de N, cela lorsque N n'est pas accessible depuis les valeurs de `L0`.