**DS Informatique (concours blanc) durée 3h** **PC/PC***

**Merci de bien préciser sur la copie le numéro (1, 2, 3, etc.) de chaque question traitée.**
*Ce sujet est composé de quatre parties complètement indépendantes.*

## Partie 1 Questions de cours

**1.** On considère la suite $(u_n)_{n\geqslant 0}$ définie par :

$$u_0 = 2$$
$$\forall n \in \mathbb{N},\ u_{n+1} = 2u_n - n$$

Écrire une fonction *récursive* `u(n)` qui calcule et renvoie la valeur de $u_n$.

**2.** Donner la complexité temporelle dans le pire cas des trois algorithmes de tri au programme.

**3.** *On suppose que l'on dispose d'un type pile ainsi que des fonctions primitives suivantes :*

- `pile_vide()` *qui construit et renvoie une nouvelle pile, ne contenant aucun élément ;*
- `empiler(P,x)` *qui ajoute l'élément* `x` *au sommet de la pile* `P` *(cette fonction ne renvoie aucun résultat mais modifie directement la pile* `P` *passée en paramètre) ;*
- `depiler(P)` *qui retire l'élément qui est au sommet de la pile et renvoie sa valeur (cette fonction modifie la pile* `P` *passée en paramètre, de plus elle produit une erreur si la pile ne contient aucun élément) ;*
- `est_vide(P)` *qui renvoie* `True` *si la pile* `P` *est vide et* `False` *sinon.*

*Les piles ne pourront être manipulées qu'à travers ces fonctions exclusivement.*
Écrire une fonction `somme(P)` qui calcule et renvoie la somme des éléments contenus dans une pile de nombres $P$. On veut également qu'à la fin de cette fonction le contenu de la pile $P$ soit identique à ce qu'il était au début.

## Partie 2 Bases de données

*On considère une base de données enregistrant des informations sur des films. La première table répertorie des réalisateurs en enregistrant pour chacun d'eux un identifiant et son nom :*

**table realisateurs (extrait)**

| IdRealisateur | NomRealisateur |
|---|---|
| 20 | The Wachowski Brothers |
| 25 | John Cornell |

*La seconde table répertorie des films en enregistrant pour chacun d'eux un identifiant, le nom du film, son réalisateur (donné par son identifiant), son année de sortie, son budget et les recettes réalisées :*

**table films (extrait)**

| IdFilm | NomFilm | IdRealisateur | AnneeSortie | Budget | Recettes |
|---|---|---|---|---|---|
| 10003 | Crocodile Dundee II | 25 | 1988 | 15800000 | 239606210 |
| 10004 | The Matrix | 20 | 1999 | 63000000 | 463500000 |

**4.** Écrire une commande SQL permettant d'obtenir les noms des films dont le budget est supérieur (ou égal) à 10 000 000.

**5.** Écrire une commande SQL permettant d'obtenir les noms des réalisateurs ayant réalisé un film dont le budget est supérieur ou égal à 10 000 000.

**6.** Écrire une commande SQL permettant d'obtenir le total des recettes de chaque année.

**7.** Écrire une commande SQL permettant d'obtenir le nom (uniquement) du (ou des) film(s) ayant réalisé la recette la plus importante.

## Partie 3 Modélisation

*On considère dans cette partie une solution $x(t)$ de l'équation différentielle :*

$$(E) \qquad x''(t) + \omega^2 x(t) = 0$$

*avec les conditions initiales $x(0) = x_0$ et $x'(0) = v_0$ où $x_0$, $v_0$ et $\omega$ sont des constantes et $\omega > 0$. On souhaite approcher numériquement cette solution $x(t)$ et on considère pour cela un réel $\Delta_t > 0$ fixé ainsi qu'un entier $N > 0$.*
*On pourra supposer, sans avoir à l'écrire sur la copie, que le programme* PYTHON *commence par la déclaration :*

```
import numpy as np
```

*et se poursuit avec la définition des constantes* **`x0`**, **`v0`**, **`omega`**, **`Deltat`** *et* **`N`** *qui correspondent respectivement à $x_0$, $v_0$, $\omega$, $\Delta_t$ et $N$.*
*On notera $v$ la fonction définie par $v(t) = x'(t)$.*

**8.** Justifier que la quantité $v(t)^2 + \omega^2 x(t)^2$ est indépendante du temps.

**9.** Écrire le système d'équations différentielles d'ordre 1 vérifié par les fonctions $x$ et $v$.

**10.** Justifier, si $\Delta_t$ est assez petit, l'approximation : $x(t+\Delta t) \simeq x(t) + \Delta_t v(t) - \dfrac{\Delta_t^2}{2}\omega^2 x(t)$.

*On présente ici une méthode de résolution numérique de l'équation différentielle $(E)$, appelée méthode Leapfrog. Cette méthode consiste à construire deux suites $(x_i)_{i\in\mathbb{N}}$ et $(v_i)_{i\in\mathbb{N}}$ définies à partir des valeurs initiales $x_0$ et $v_0$ à l'aide des relations de récurrence :*

$$x_{i+1} = x_i + \Delta_t v_i - \frac{\Delta_t^2}{2}\omega^2 x_i$$

$$v_{i+1} = v_i - \frac{\Delta_t}{2}\omega^2 (x_i + x_{i+1})$$

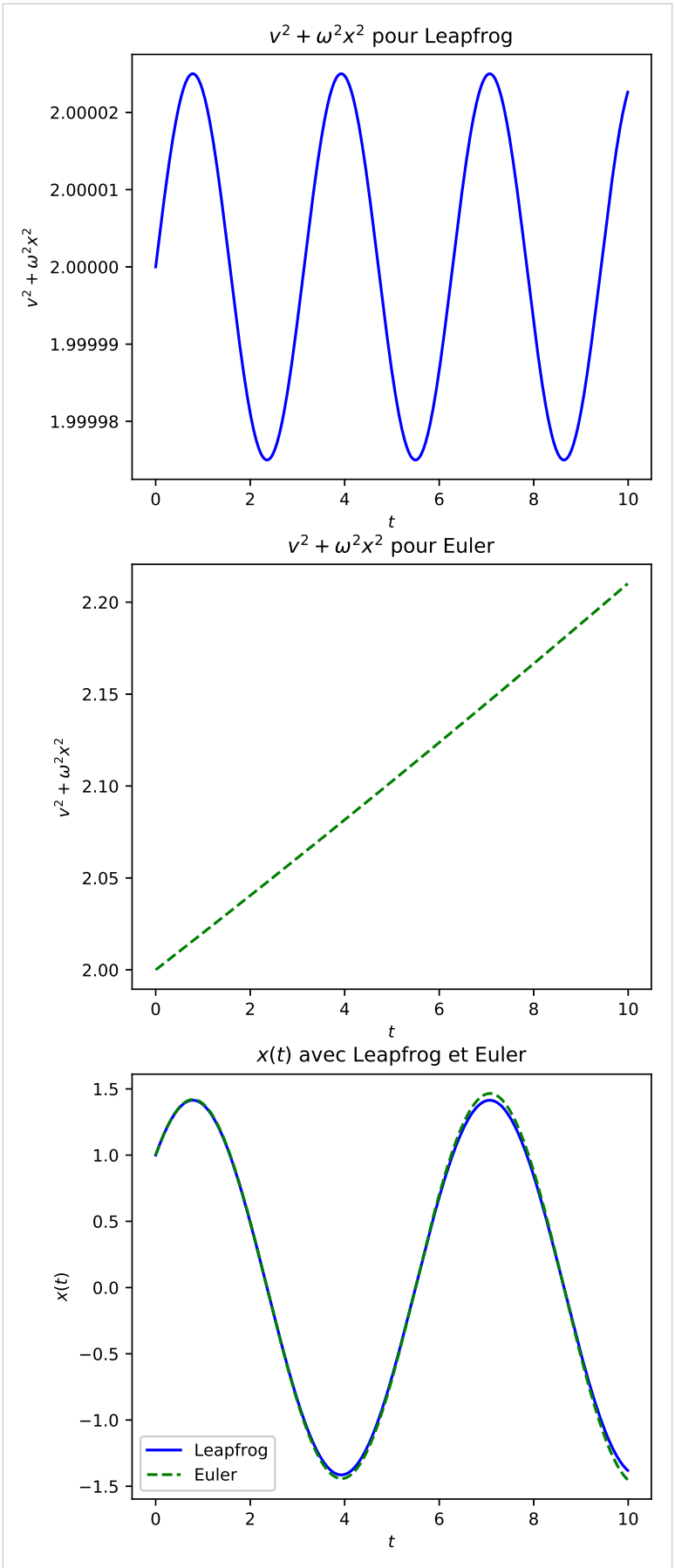
*Les valeurs $x_i$ et $v_i$ sont les valeurs approchées de $x(i\Delta_t)$ et $v(i\Delta_t)$.*

**11.** Écrire le code permettant de construire la liste $\mathbf{t} = [t_0, \ldots, t_{N-1}]$ avec $t_i = i\Delta_t$. On rappelle que $N$ et $\Delta_t$ sont fixés et définis par ailleurs dans le programme PYTHON.

**12.** Écrire le code permettant de construire les listes $\mathbf{x} = [x_0, \ldots, x_{N-1}]$ et $\mathbf{v} = [v_0, \ldots, v_{N-1}]$.

**13.** Si on avait choisi d'utiliser la méthode d'Euler, comment aurait on défini $x_{i+1}$ et $v_{i+1}$ en fonction de $x_i$ et $v_i$ ?

**14.** On donne ci-dessous la représentation graphique de $v(t)^2 + \omega^2 x(t)^2$ en fonction de $t$ en utilisant les valeurs numériques obtenues avec la méthode Leapfrog (trait plein) ainsi que celles obtenues avec la méthode d'Euler (pointillés). Sur la troisième représentation graphique, on a représenté conjointement l'approximation numérique de $x(t)$ obtenue par la méthode Leapfrog (traits pleins) et la méthode d'Euler (en pointillés). Commenter les résultats obtenus.


$v^2 + \omega^2 x^2$ pour Leapfrog
2.00002
2.00001
2.00000
1.99999
1.99998
$v^2 + \omega^2 x^2$
0
2
4
6
8
10
$t$
$v^2 + \omega^2 x^2$ pour Euler
2.20
2.15
2.10
2.05
2.00
$v^2 + \omega^2 x^2$
0
2
4
6
8
10
$t$
$x(t)$ avec Leapfrog et Euler
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
$x(t)$
Leapfrog
Euler
0
2
4
6
8
10
$t$

## Partie 4 Algorithmique

***Quelques rappels sur les chaines de caractères.*** *Une chaine de caractères (on dira plus simplement « chaine » dans la suite) est un élément du type* Str *en* PYTHON *qui contient une suite de caractères. On appelle longueur de la chaine le nombre de caractères qu'elle contient, on l'obtient avec la fonction* **`len`** *de* PYTHON. *Une chaine de longueur nulle est appelée* chaine vide. *Si $s$ est une chaine de longueur $n$ et si $i$ est un entier tel que $0 \leqslant i < n$, alors $s[i]$ représente le caractère d'indice $i$ de la chaine $s$, les indices étant numérotés à partir de 0. On utilisera aussi la notation suivante : si $s$ est une chaine de longueur $n$ et $i$ est un entier tel que $0 \leqslant i \leqslant n$, alors $s[:i]$ représente la chaine obtenue en prenant les $i$ premiers caractères de $s$ (c'est à dire ceux dont les indices vont de 0 à $i-1$.*

***Quelques exemples.*** *Si $t =$* **`''`** *(chaine vide), alors* **`len`**$(t)$ *renvoie 0. Si $s =$* **`'Bonjour'`**, *alors* **`len`**$(s)$ *renvoie 7, $s[0]$ correspond à* **`'B'`** *et $s[6]$ correspond à* **`'r'`**. *Toujours sur cet exemple, $s[:3]$ est la chaine* **`'Bon'`** *et on a les cas particuliers $s[:0]$ (chaine vide) et $s[:7]$ (chaine identique à $s$).*

***Remarque importante.*** *On déconseille vivement au candidat d'utiliser toute autre opération sur les chaines que celles présentées ici.*

***Contexte.*** *Lorsque l'on utilise un correcteur orthographique, les mots du texte fournis sont comparés à ceux présents dans un dictionnaire et, lorsque le mot n'est pas trouvé, des propositions de remplacement sont faites. Ces propositions de remplacement sont obtenues en recherchant dans le dictionnaire des mots « proches » du mot erroné. On étudie dans cette partie des méthodes permettant de déterminer si deux chaines sont proches l'une de l'autre. Ceci sera réalisé en définissant sur les chaines une notion de distance; deux chaines seront alors d'autant plus proches que leur distance est petite et elles devront être identiques lorsque leur distance est nulle.*

***Première notion de distance.*** *Si $s$ et $t$ sont deux chaines de longueurs respectives $n$ et $m$ (éventuellement différentes), on définit leur distance notée $\Delta(s,t)$ en posant :*

$$\Delta(s,t) = |n-m| + c$$

*où $c$ est le nombre d'entiers $i$ tels que $0 \leqslant i < \min(n,m)$ et $s[i] \neq t[i]$.*

**15.** Déterminer $\Delta(s,t)$ lorsque $s$ est la chaine **`'ABAB'`** et $t$ est la chaine **`'ABB'`**.

**16.** Déterminer $\Delta(s,t)$ lorsque l'une des chaines $s$ ou $t$ est vide.

**17.** Écrire une fonction **`delta(s,t)`** qui calcule et renvoie la valeur de la distance $\Delta(s,t)$.

***Deuxième distance,*** *la distance de* LEVENSHTEIN. *La distance $\Delta$ définie plus haut n'est pas forcément pertinente dans tous les cas. Par exemple si $s$ est la chaine* **`'ABAB'`** *et $t$ est la chaine* **`'BAB'`** *alors $\Delta(s,t) = 4$ bien que $s$ et $t$ soient assez proches puisqu'il suffit de retirer le premier caractère de $s$ pour obtenir $t$. Ceci conduit à considérer la distance de* LEVENSHTEIN *qui est définie en comptant combien d'opérations il faut réaliser pour passer d'une chaine à l'autre. Plus précisément, les opérations considérées sont les suivantes :*

- *Suppression d'un caractère à une position de la chaine;*
- *Insertion d'un caractère à une position de la chaine (éventuellement au début ou à la fin);*
- *Remplacement d'un caractère à une position de la chaine par un autre.*

*La distance de* LEVENSHTEIN *entre deux chaines $s$ et $t$ est alors le nombre minimal de ces opérations qu'il faut réaliser en partant de $s$ pour obtenir $t$. On la notera $\Delta_L(s,t)$.*

***Par exemple,*** *si $s$ est la chaine* **`'ABC'`** *et $t$ est la chaine* **`'BA'`**, *alors on peut passer de $s$ à $t$ en réalisant deux opérations : tout d'abord remplacer le dernier caractère* **`'C'`** *par* **`'A'`**, *ce qui donne* **`'ABA'`** *puis ensuite supprimer le premier caractère ce qui donne* **`'BA'`**. *De plus, il n'est pas possible d'obtenir $t$ à partir de $s$ en utilisant une seule opération, donc $\Delta_L(s,t) = 2$.*

**18.** Déterminer $\Delta_L(s,t)$ lorsque $s$ est la chaine **`'ABC'`** et $t$ est la chaine **`'AA'`**.

**19.** Déterminer $\Delta_L(s,t)$ lorsque l'une des chaines $s$ ou $t$ est vide.

*On donne le code suivant.*

```
1  import numpy as np
2
3  def deltaL(s,t):
4      n = len(s)
5      m = len(t)
6      d = np.zeros((n+1,m+1))
7      for i in range(0,n+1):
8          d[i,0] = i
9      for j in range(0,m+1):
10         d[0,j] = j
11
12     for i in range(0,n):
13         for j in range(0,m):
14             if s[i]==t[j]:
15                 rempl = 0
16             else:
17                 rempl = 1
18             d[i+1,j+1] = min(d[i,j+1]+1,d[i+1,j]+1,d[i,j]+rempl)
19
20     return d[n,m]
```

*On rappelle que l'instruction* **`d=np.zeros((n+1,m+1))`** *construit une matrice nulle à* $n+1$ *lignes et* $m+1$ *colonne. Les indices de ligne* $i$ *vont alors de* 0 *à* $n$ *(inclus) et les indices de colonne* $j$ *vont de* 0 *à* $m$ *(inclus). On rappelle également que* **`min(a,b,c)`** *renvoie le minimum des nombres* $a$, $b$ *et* $c$.
*On* ***admettra*** *dans la suite que la matrice* **`d`** *obtenue une fois toutes les boucles effectuées, c'est à dire en ligne 19, vérifie la propriété suivante :*

$$\mathbf{d}[i,j] = \Delta_L(s[:i], t[:j])$$

*quels que soient les entiers* $i$ *et* $j$ *avec* $0 \leqslant i \leqslant n$ *et* $0 \leqslant j \leqslant m$.

**20.** À l'aide de cette propriété admise, justifier que la fonction **`deltaL`** renvoie bien la valeur $\Delta_L(s,t)$.

**21.** On considère, dans cette question, que l'on appelle la fonction **`deltaL`** avec la chaine $s$ égale à `'ABC'` et $t$ la chaine `'AA'`.

(a) Quelle est la matrice **`d`** une fois que les deux premières boucles ont été effectuées (en ligne 11) ?

(b) Quelle est la matrice **`d`** juste avant d'effectuer l'instruction **`return`** (en ligne 19) ?

**22.** Déterminer la complexité en temps de la fonction **`deltaL`** en fonction de $n$ et $m$, longueurs des chaines données en paramètres.

**23.** Déterminer la complexité en espace de la fonction **`deltaL`** en fonction de $n$ et $m$, longueurs des chaines données en paramètres.

**24.** Écrire une fonction **`deltaL2(s,t)`** qui calcule la distance de LEVENSHTEIN entre les chaines $s$ et $t$, dont la complexité en temps est $O(nm)$ et la complexité en espace est $O(m)$ avec $n$ et $m$ longueurs respectives des deux chaines données en paramètres.

***Remarque.*** *La distance de* DAMERAU–LEVENSHTEIN *généralise la distance de* LEVENSHTEIN *en ajoutant la possibilité d'une quatrième opération : l'échange de deux caractères consécutifs. Cette opération a été proposée par Frederick J.* DAMERAU *qui a signalé que les quatre opérations obtenues correspondent à plus de* 80% *des fautes d'orthographe réalisées. La distance de* DAMERAU-LEVENSHTEIN, *proposée au début pour des applications aux correcteurs d'orthographe, s'est avérée utile en biologie, dans l'étude des séquences d'ADN.*