A2021 – INFO

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

**Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).**

CONCOURS 2021

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée de l'épreuve : 2 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI.

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

INFORMATIQUE COMMUNE

L'énoncé de cette épreuve comporte 10 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.

Marchons, marchons, marchons...

Ce sujet propose d'appliquer des techniques informatiques à l'étude de trois marches de nature différente :

— une marche concrète (partie I - Randonnée)

— une marche stochastique (partie II - Mouvement brownien d'une petite particule)

— une marche auto-évitante (partie III - Marche auto-évitante)

Les trois parties sont indépendantes. L'annexe à la fin du sujet (pages 8 à 10) fournit les canevas de code en Python que vous devrez reprendre dans votre copie pour répondre aux questions de programmation.

Partie I. Randonnée

Lors de la préparation d'une randonnée, une accompagnatrice doit prendre en compte les exigences des participants. Elle dispose d'informations rassemblées dans deux tables d'une base de données :

— la table `Rando` décrit les randonnées possibles – la clef primaire entière `rid`, son nom, le niveau de difficulté du parcours (entier entre 1 et 5), le dénivelé (en mètres), la durée moyenne (en minutes) :

rid	**rnom**	**diff**	**deniv**	**duree**
1	La belle des champs	1	20	30
2	Lac de Castellane	4	650	150
3	Le tour du mont	2	200	120
4	Les crêtes de la mort	5	1200	360
5	Yukon Ho !	3	700	210
...	...	...	...	...

— la table `Participant` décrit les randonneurs – la clef primaire entière `pid`, le nom du randonneur, son année de naissance, le niveau de difficulté maximum de ses randonnées :

pid	**pnom**	**ne**	**diff_max**
1	Calvin	2014	2
2	Hobbes	2015	2
3	Susie	2014	2
4	Rosalyn	2001	4
...	...	...	...

Donner une requête SQL sur cette base pour :

❏ **Q1** – Compter le nombre de participants nés entre 1999 et 2003 inclus.

❏ **Q2** – Calculer la durée moyenne des randonnées pour chaque niveau de difficulté.

❏ **Q3** – Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile.

❏ **Q4** – Extraire les clés primaires des randonnées qui ont un ou des homonymes (nom identique et clé primaire distincte), sans redondance.

L'accompagnatrice a activé le suivi d'une randonnée par géolocalisation satellitaire et souhaite obtenir quelques propriétés de cette randonnée une fois celle-ci effectuée. Elle a exporté les données au format texte CSV (*comma-separated values* – valeurs séparées par des virgules) dans un fichier nommé `suivi_rando.csv` : la première ligne annonce le format, les suivantes donnent les positions dans l'ordre chronologique.

Voici le début de ce fichier pour une randonnée partant de Valmorel, en Savoie, un bel après-midi d'été :

```
lat(°),long(°),height(m),time(s)
45.461516,6.44461,1315.221,1597496965
45.461448,6.444426,1315.702,1597496980
45.461383,6.444239,1316.182,1597496995
45.461641,6.444035,1316.663,1597496710
45.461534,6.443879,1317.144,1597496725
45.461595,6.4437,1317.634,1597496740
45.461562,6.443521,1318.105,1597496755
...
```

Le module `math` de Python fournit les fonctions `asin`, `sin`, `cos`, `sqrt` et `radians`. Cette dernière convertit des degrés en radians. La documentation donne aussi des éléments de manipulation de fichiers textuels :

`fichier = open(nom_fichier, mode)` ouvre le fichier, en lecture si mode est `"r"`.

`ligne = fichier.readline()` récupère la ligne suivante de `fichier` ouvert en lecture avec `open`.

`lignes = fichier.readlines()` donne la liste des lignes suivantes.

`fichier.close()` ferme `fichier`, ouvert avec `open`, après son utilisation.

`ligne.split(sep)` découpe la chaîne de caractères `ligne` selon le séparateur `sep` : si `ligne` vaut `"42,43,44"`, alors `ligne.split(",")` renvoie la liste `["42", "43", "44"]`.

On souhaite exploiter le fichier de suivi d'une randonnée – supposé préalablement placé dans le répertoire de travail – pour obtenir une liste `coords` des listes de 4 flottants (latitude, longitude, altitude, temps) représentant les points de passage collectés lors de la randonnée.

À partir du canevas fourni en annexe et en ajoutant les `import` nécessaires :

❏ **Q5** – Implémenter la fonction `importe_rando(nom_fichier)` qui réalise cette importation en retournant la liste souhaitée, par exemple en utilisant certaines des fonctions ci-dessus.

On suppose maintenant l'importation effectuée dans `coords`, avec au moins deux points d'instants distincts.

❏ **Q6** – Implémenter la fonction `plus_haut(coords)` qui renvoie la liste (latitude, longitude) du point le plus haut de la randonnée.

❏ **Q7** – Implémenter la fonction `deniveles(coords)` qui calcule les dénivelés cumulés positif et négatif (en mètres) de la randonnée, sous forme d'une liste de deux flottants. Le dénivelé positif est la somme des variations d'altitude positives sur le chemin, et inversement pour le dénivelé négatif.

On souhaite évaluer de manière approchée la distance parcourue lors d'une randonnée. On suppose la Terre parfaitement sphérique de rayon $R_T = 6371$ km au niveau de la mer. On utilise la formule de haversine pour calculer la distance d du grand cercle sur une sphère de rayon r entre deux points de coordonnées (latitude, longitude) (φ_1, λ_1) et (φ_2, λ_2) :

$$d = 2\,r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

On prendra en compte l'altitude moyenne de l'arc, que l'on complètera pour la variation d'altitude par la formule de Pythagore, en assimilant la portion de cercle à un segment droit perpendiculaire à la verticale.

❏ **Q8** – Implémenter la fonction `distance(c1, c2)` qui calcule avec cette approche la distance en mètres entre deux points de passage. On décomposera obligatoirement les formules pour en améliorer la lisibilité.

❏ **Q9** – Implémenter la fonction `distance_totale(coords)` qui calcule la distance en mètres parcourue au cours d'une randonnée.

Partie II. Mouvement brownien d'une petite particule

De petites particules en suspension dans un liquide se déplacent spontanément sous l'effet de l'agitation thermique du milieu environnant, donnant ainsi naissance à des trajectoires apparemment chaotiques et peu régulières. Ce phénomène est à la base de la vie : l'agitation incessante des protéines permet aux réactions biochimiques de se produire à l'intérieur de nos cellules. Il a été observé expérimentalement en 1827 sur des grains de pollen en suspension dans l'eau par le botaniste Robert Brown, d'où le nom de mouvement brownien. Son étude théorique par Albert Einstein en 1905 a permis au physicien Jean Perrin d'estimer la valeur du nombre d'Avogadro dans une série d'expériences menées entre 1907 et 1909.

Dans cette partie, on s'intéresse à un modèle du mouvement brownien proposé par Paul Langevin en 1908. Dans ce modèle, la particule étudiée est supposée soumise à deux actions de la part du fluide :

— une force de frottement fluide $\overrightarrow{f_F} = -\alpha\,\overrightarrow{v}$;
— une force $\overrightarrow{f_B}$ aléatoire simulant l'action désordonnée des molécules d'eau sur la particule.

Le mouvement de cette particule est donc régi par l'équation différentielle :

$$\frac{d\overrightarrow{v}}{dt} = -\frac{\alpha\,\overrightarrow{v}}{m} + \frac{\overrightarrow{f_B}}{m}$$

Pour faire une simulation en deux dimensions, on prend une particule de masse $m = 10^{-6}$ kg, on attribue à α la valeur 10^{-5} kg/s, on suppose enfin que $\overrightarrow{f_B}$ change à chaque pas d'intégration, avec une direction isotrope aléatoire (l'angle suit une loi de probabilité uniforme) et une norme qui est la valeur absolue d'une variable aléatoire suivant une loi de probabilité gaussienne (loi normale) d'espérance μ nulle et d'écart-type $\sigma = 10^{-8}$ N.

On simule le vecteur d'état $E = (x, y, \dot{x}, \dot{y})$ de la particule en intégrant $\dot{E} = (\dot{x}, \dot{y}, \ddot{x}, \ddot{y})$ selon la méthode d'Euler. E et $\dot{E}$ seront chacun représentés par une liste de 4 flottants.

L'instruction `assert` *`expression`* de Python vérifie la véracité d'une expression booléenne et interrompt brutalement l'exécution du programme si ce n'est pas le cas. Elle permet de vérifier très simplement une précondition ou un invariant, comme illustré à la ligne 10 du canevas.

Le module `random` fournit la fonction `uniform(bi, bs)` qui renvoie un flottant aléatoire entre les valeurs `bi` et `bs` incluses en utilisant une densité de probabilité uniforme ainsi que la fonction `gauss(mu, sigma)` qui renvoie un flottant aléatoire en utilisant une densité de probabilité gaussienne d'espérance `mu` et d'écart-type `sigma`.

Le module `math` fournit enfin les fonctions `cos` et `sin` (en radians), la fonction `sqrt` et la constante `pi`. La fonction `abs` est directement disponible.

En utilisant le canevas fourni en annexe et en ajoutant les `import` nécessaires :

❏ **Q10** – Implémenter la fonction `vma(v1, a, v2)` (multiplication-addition sur des vecteurs) avec `v1` et `v2` des listes de flottants et `a` un scalaire flottant, qui renvoie une nouvelle liste de flottants correspondant à :

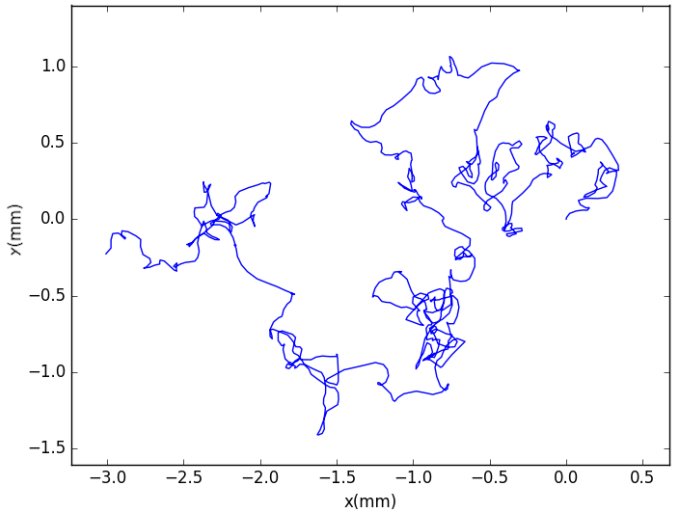
$$\overrightarrow{v} = \overrightarrow{v_1} + a\,\overrightarrow{v_2}$$

La fonction vérifiera que les deux listes en entrée sont de même longueur avec `assert`.

❏ **Q11** – Implémenter la fonction `derive(E)` qui renvoie la dérivée du vecteur d'état passé en paramètre d'après l'équation différentielle décrite en introduction de la partie.

❏ **Q12** – Implémenter la fonction `euler(E0, dt, n)` pour résoudre numériquement l'équation différentielle par la méthode d'Euler avec `E0` le vecteur d'état initial, `dt` le pas d'intégration et `n` le nombre de pas de la simulation. La fonction renvoie la liste des $n + 1$ vecteurs d'état.

À titre d'exemple, voici le tracé d'une trajectoire obtenue par cette méthode :



Partie III. Marche auto-évitante

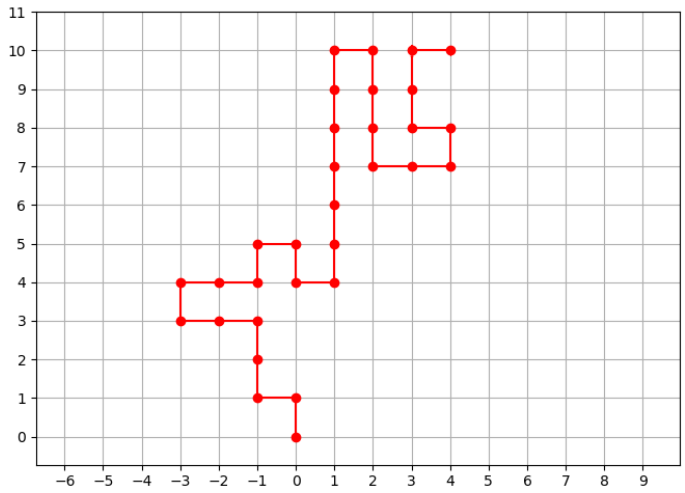
Une marche auto-évitante est un processus au cours duquel un point décrit un chemin auto-évitant, c'est-à-dire qui ne se recoupe jamais lui-même. Ce modèle peut servir lors de la modélisation d'une chaîne polymère. En effet, deux monomères distincts de la chaîne ne peuvent pas se trouver à deux positions identiques pour des raisons d'encombrement stérique.

Dans ce sujet, on appellera chemin auto-évitant (CAE) de longueur n tout ensemble de $n+1$ points $P_i \in \mathbb{Z}^2$ pour $0 \leq i \leq n$ tels que :

- $\forall i \qquad \|P_{i+1} - P_i\| = 1$
- $\forall (i,j) \quad i \neq j \Rightarrow P_i \neq P_j$

Chaque point du chemin sera représenté par une liste à deux éléments entiers précisant les deux coordonnées (par exemple $P_i = (5,-4)$ est représenté par la liste `[5,-4]`). Le chemin lui-même est constitué de la liste des points, dans l'ordre dans lequel ils sont atteints. Les codes Python produits dans cette partie devront manipuler exclusivement des coordonnées entières.

Voici un exemple de représentation graphique de CAE de longueur 30 à partir de $(0,0)$:



On s'intéresse dans un premier temps à une méthode naïve pour générer un chemin auto-évitant de longueur n sur une grille carrée. La méthode adoptée est une approche gloutonne :

1. Le premier point est choisi à l'origine : $P_0 = (0, 0)$.
2. En chaque position atteinte par le chemin, on recense les positions voisines accessibles pour le pas suivant et on en sélectionne une au hasard. En l'absence de positions accessibles l'algorithme échoue.
3. On itère l'étape 2 jusqu'à ce que le chemin possède la longueur désirée ou échoue.

Le module `random` fournit la fonction `randrange(a, b)` qui renvoie un entier compris entre a et $b - 1$ inclus, ainsi que la fonction `choice(L)` qui renvoie l'un des éléments de la liste `L`, dans les deux cas choisis aléatoirement avec une probabilité uniforme.

L'expression `x in L` est une expression booléenne qui vaut `True` si `x` est l'un des éléments de `L` et `False` dans le cas contraire. On supposera que la méthode employée pour évaluer cette expression sur une liste est une recherche séquentielle.

La valeur spéciale `None` est utilisée en Python pour représenter une valeur invalide, inconnue ou indéfinie. L'expression booléenne `v is None` indique si la valeur de `v` est cette valeur spéciale.

En utilisant le canevas fourni en annexe et en ajoutant les `import` nécessaires :

❏ **Q13** – Implémenter la fonction `positions_possibles(p, atteints)` qui construit la liste des positions suivantes possibles à partir du point `p`. La liste `atteints` contient les points déjà atteints par le chemin.

❏ **Q14** – Mettre en évidence graphiquement un exemple de CAE le plus court possible pour lequel, à une étape donnée, la fonction `positions_possibles` va renvoyer une liste vide. En prenant en compte les symétries, combien de tels chemins distincts existent pour cette longueur minimale ?

❏ **Q15** – Implémenter la fonction `genere_chemin_naif(n)` qui construit la liste des points représentant le chemin auto-évitant de longueur `n` et renvoie le résultat, ou bien renvoie la valeur spéciale `None` si à une étape `positions_possibles` renvoie une liste vide.

❏ **Q16** – Évaluer avec soin la complexité temporelle asymptotique dans le pire des cas de la fonction `genere_chemin_naif(n)` en fonction de `n`, en supposant que la fonction ne renvoie pas `None`.

Une personne curieuse et patiente a écrit le code suivant :

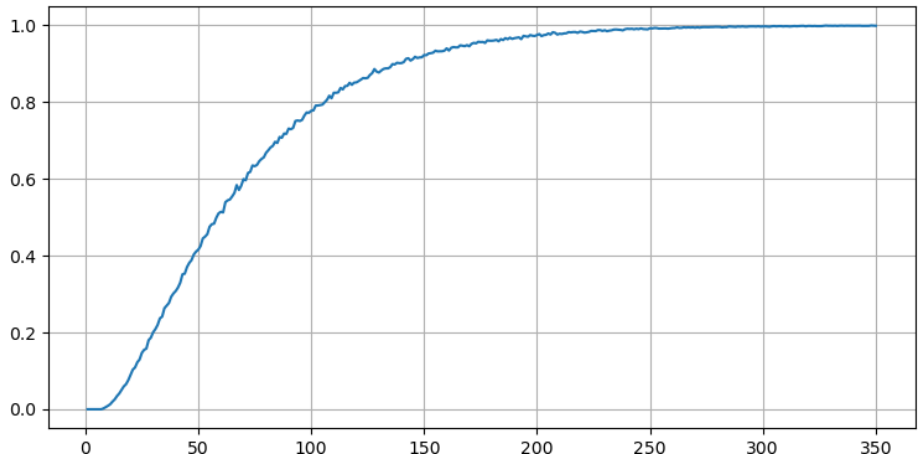
```
from chemin import genere_chemin_naif

N, M, L, P = 10000, 351, [], []

for n in range(1, M):
    nb = 0
    for i in range(N):
        chemin = genere_chemin_naif(n)
        if chemin is None:
            nb += 1
    L.append(n)
    P.append(nb / N)

import matplotlib.pyplot as plt
plt.plot(L, P)
plt.grid()
plt.show()
```

Après un long moment, elle a obtenu le graphique suivant, volontairement laissé sans étiquettes d'axes :

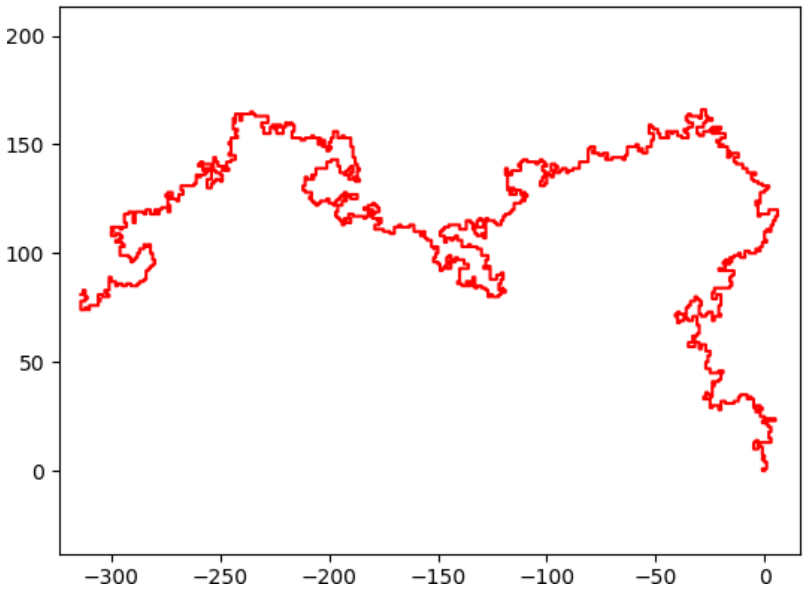


❏ **Q17** – Décrire ce que représente ce graphique et interpréter sa signification pour la méthode naïve.

Afin d'éviter les inconvénients de la méthode précédente, on s'intéresse à une solution différente nommée *méthode du pivot*, proposée par Moti Lal en 1969. Son principe est le suivant :

1. On part d'un chemin auto-évitant arbitraire de longueur n. Ici, on choisira une initialisation très simple, le chemin droit `[ [0, 0], [1, 0], [2, 0], ..., [n, 0] ]`.
2. On sélectionne au hasard un point, nommé pivot, entre le second et l'avant-dernier point du chemin, et un angle aléatoire de rotation parmi π, $\frac{\pi}{2}$ et $-\frac{\pi}{2}$.
3. On laisse les points avant le pivot inchangés et on fait subir à l'ensemble des points situés strictement après le pivot une rotation ayant pour centre le pivot et pour angle, l'angle choisi à l'étape 2 ci-dessus.
4. Si le chemin ainsi obtenu est auto-évitant, on le garde. Sinon, on reprend à l'étape 2 la sélection d'un pivot et d'un angle, jusqu'à en trouver une paire qui conviennent.
5. On répète les étapes 2 à 4 un certain nombre de fois. Le choix du nombre minimal de rotations à effectuer pour obtenir un chemin non corrélé au chemin initial est laissé de côté dans ce sujet.

Cette méthode permet de générer de manière efficace des marches auto-évitantes de plusieurs milliers de points, comme la marche ci-dessous de longueur 2000 :



Dans cet algorithme, une étape importante est la vérification qu'un chemin donné est auto-évitant. Pour vérifier cela on pourrait bien sûr s'y prendre comme dans la fonction `positions_possibles`, mais on adopte une méthode différente avec une fonction `est_CAE(chemin)` qui trie les points du chemin puis met à profit ce tri pour vérifier efficacement que le chemin est auto-évitant.

On utilise pour cela la fonction `sorted` qui renvoie une nouvelle liste triée par ordre croissant. Elle fonctionne sur tout type d'éléments disposant d'une relation d'ordre, y compris des listes pour lesquelles l'ordre lexicographique (ordre du premier élément, en cas d'égalité du second, etc.) est appliqué. On suppose de plus que la complexité temporelle asymptotique dans le pire des cas de cette fonction est la meilleure possible.

❏ **Q18** – Rappeler, sans la justifier, la complexité temporelle asymptotique dans le pire des cas attendue de `sorted` en fonction de la longueur de la liste à trier. Donner le nom d'un algorithme possible pour son implémentation.

En utilisant le canevas fourni en annexe et en ajoutant les `import` nécessaires :

❏ **Q19** – Implémenter la fonction `est_CAE(chemin)` qui vérifie si un chemin est auto-évitant en se basant sur `sorted` et renvoie un résultat booléen. Elle devra ne pas être de complexité temporelle asymptotique dans le pire des cas supérieure à la fonction `sorted` : vous prouverez ce dernier point.

❏ **Q20** – Implémenter la fonction `rot(p, q, a)` qui renvoie le point image du point `q` par la rotation de centre `p` et d'angle défini par la valeur de `a` : `0` pour π, `1` pour $\frac{\pi}{2}$, `2` pour $-\frac{\pi}{2}$.

❏ **Q21** – Implémenter la fonction `rotation(chemin, i_piv, a)` qui renvoie un nouveau chemin identique à `chemin` jusqu'au pivot d'indice `i_piv`, et ayant subi une rotation de `a` autour du pivot (même codage que la fonction précédente) sur la partie strictement après le pivot.

❏ **Q22** – Implémenter la fonction `genere_chemin_pivot(n, n_rot)` permettant de générer un chemin auto-évitant de longueur `n` en appliquant `n_rot` rotations. L'application d'une rotation peut nécessiter plusieurs tentatives.

❏ **Q23** – On considère un pivot, son point précédent et son point suivant. Quel est l'impact prévisible sur les rotations admissibles ? Suggérer un moyen de prendre en compte cette propriété pour améliorer l'algorithme.

Fin de l'énoncé

Annexe canevas de codes Python sur les pages suivantes

Annexe : canevas de codes Python

— Partie I : Randonnée

```python
# import Python à compléter...

# importation du fichier d'une randonnée
def importe_rando(nom_fichier):
    # À compléter...

coords = importe_rando("suivi_rando.csv")

# donne le point (latitude, longitude) le plus haut de la randonnée
def plus_haut(coords):
    # À compléter...

print("point le plus haut", plus_haut(coords))
# exemple : point le plus haut [45.461451, 6.443064]

# calcul des dénivelés positif et négatif de la randonnée
def deniveles(coords):
    # À compléter...

print("dénivelés", deniveles(coords))
# exemple : denivelés [4.059999999999945, -1.1759999999999309]

RT = 6371 # rayon moyen volumétrique de la Terre en km

# distance entre deux points
def distance(c1, c2):
    # À compléter...

print("premier intervalle", distance(coords[0], coords[1]), "m")
# exemple : premier intervalle 16.230964254992816 m

# distance totale de la randonnée
def distance_totale(coords):
    # À compléter...

print("distance parcourue", distance_totale(coords), "m")
# exemple : distance parcourue 187.9700904658368 m
```

— Partie II : Mouvement brownien

```python
# import Python à compléter...

# paramètres physiques
MU = 0.0       # N
SIGMA = 1E-8   # N
M = 1E-6       # kg
ALPHA = 1E-5   # kg/s

# vérification des hypothèses sur les paramètres
assert MU >= 0 and SIGMA > 0 and M > 0 and ALPHA > 0

# multiplication-addition vectorielle
def vma(v1, a, v2):
    # À compléter...

# dérivée du vecteur d'état
def derive(E):
    # À compléter...

# intégration par la méthode d'Euler
def euler(E0, dt, n):
    Es = [ E0 ]
    # À compléter...
    return Es

# simulation
DT = 0.002     # durée du pas en secondes
N = 5000       # nombre de pas

Es = euler([ 0.0, 0.0, 0.0, 0.0 ], DT, N)

print("trajectoire", Es)
```

— Partie III : Chemin auto-évitant - méthode naïve

```python
# import Python à compléter...

# positions auto-évitantes suivantes possibles
def positions_possibles(p, atteints):
    possibles = []
    # À compléter...
    return possibles

# génération gloutonne d'un chemin de longueur n
# renvoie None en cas d'échec
def genere_chemin_naif(n):
    chemin = [ [ 0, 0 ] ] # on part de l'origine
    # À compléter...
    return chemin

N = 10
print("chemin", genere_chemin_naif(N))
```

— Partie III : Chemin auto-évitant - méthode du pivot

```python
# import Python à compléter...

# vérifie si un chemin est CAE
def est_CAE(chemin):
    # À compléter...

# calcule la rotation de q autour de p selon a :
# Pi si a vaut 0, Pi/2 si a vaut 1, -Pi/2 si a vaut 2
def rot(p, q, a):
    # À compléter...

# renvoie le chemin dont les points après i_pivot
# ont subi une rotation a codé comme précédemment
def rotation(chemin, i_pivot, a):
    # À compléter...

# génère un chemin de longueur n (donc n+1 points)
def genere_chemin_pivot(n, n_rot):
    # À compléter...

N, A = 1000, 2.3
print("chemin", genere_chemin_pivot(N, int( A * N )))
```

4 Informatique

4.1 Informatique pour tous

4.1.1 Généralités et présentation du sujet

Le sujet d'informatique commune comportait cette année trois parties indépendantes, chacune étant axée sur un type de marche différent. Dans la première partie, on menait l'étude d'une randonnée concrète via des questions sur les bases de données et des questions plus algorithmiques visant à déterminer les caractéristiques usuelles (distance parcourue, dénivelé) d'une randonnée. Dans la seconde partie, on étudiait une marche aléatoire sur un réseau, modélisant le mouvement brownien d'une petite particule en suspension dans un fluide. La dernière partie était consacrée à la génération de chemins auto-évitants dans $\mathbb{Z}^2$.
L'épreuve abordait ainsi un large éventail de notions étudiées durant les deux années de préparation des candidats.

4.1.2 Commentaires généraux

- Si certaines copies sont très faibles (voire presque vides), certaines sont excellentes et frisent parfois la perfection. La longueur et la difficulté du sujet étaient ainsi tout à fait adaptées à ce type d'épreuve, ce qui a permis de classer les candidats.
- Cette année encore, le jury souhaite souligner l'importance de la présentation des copies. Certaines sont très brouillonnes, sales, voire parfois illisibles. Un nombre trop important de ratures nuit forcément à la lecture des codes Python produits, et peut même provoquer des erreurs de syntaxe. On peut certes tolérer quelques ratures propres (correction d'un oubli, d'une erreur de syntaxe), qui ne nuisent pas à la poursuite de la lecture ni à la structure des codes proposés, mais un code trop difficile à déchiffrer (excès de ratures ou de rajouts par le biais de flèches ou d'astérisques) est forcément sanctionné.
- De la même façon, une erreur ponctuelle de syntaxe (oubli d'une parenthèse fermante) peut être tolérée. En revanche, l'absence récurrente des parenthèses (en écrivant par exemple systématiquement `for i in range n` ou `len L`) a été sanctionnée.
- L'importation des bibliothèques en Python doit être maîtrisée. Il existe plusieurs façons de procéder, qui ne doivent pas être confondues. On pourra par exemple écrire `import math`, `import math as m`, `from math import *` ou encore `from math import cos, sin`. Il est cependant demandé que la syntaxe de l'appel aux fonctions Python de ces bibliothèques soit en adéquation avec la syntaxe d'importation choisie.
- Le sujet demandait explicitement de manipuler des listes. À ce titre, il n'était pas opportun d'utiliser le module `numpy`. Les candidats doivent maîtriser la manipulation des listes, notamment :
  - la construction d'une liste élément par élément.Par exemple, l'initialisation d'une liste `L = [ ]` suivie, dans une boucle `for`, d'une affectation `L[i] = elt` provoque une erreur ;
  - l'ajout d'un élément à la fin d'une liste. Comme indiqué dans les rapports des années précédentes, la syntaxe `L.append(elt)` est à privilégier. D'une part, elle est plus efficace, mais elle est également moins source d'erreurs. L'emploi de la syntaxe `L = L + [elt]`(ou `L+ = [elt]`) a par exemple provoqué beaucoup d'oubli de crochets, quand `elt` était lui-même une liste ;

— les erreurs de syntaxe “à la `numpy`” : l'addition terme à terme de deux listes de même taille (respectivement la multiplication terme à terme d'une liste par un flottant ou un entier) ne peut pas se faire à l'aide d'une syntaxe du type `L1 + L2` (respectivement `a * L`), réservée aux tableaux `numpy`. L'accès à un élément d'une liste de listes se fait quant à lui *via* une syntaxe du type L[i][j], et pas `L[i, j]` ;

— le caractère modifiable des listes ; il s'agit d'un aspect subtil de Python, dont les candidats doivent avoir pris conscience au cours de leur formation. Par exemple, quand l'énoncé demande de créer une nouvelle liste, on ne peut pas juste modifier la liste entrée en paramètre d'une fonction ; de même, l'affectation `L1 = L2` ne permet pas de créer une copie indépendante de la liste `L2` ;

— la modification des éléments d'une liste ; la commande `for elt in L : elt = float(elt)` n'apporte par exemple aucun changement à la liste `L`.

- Le jury a remarqué cette année un nombre croissant de copies dans lesquelles les affectations des variables sont faites à l'envers ; par exemple, si `L` était une liste placée en paramètre d'une fonction, beaucoup de candidats ont écrit `L = a, b, c, d` au lieu de `a, b, c, d = L` pour affecter les valeurs des éléments de `L` dans les variables `a, b, c, d`. En outre, rappelons que le choix du nom d'une variable doit être valide et pertinent. Un nom de variable ne peut être composé que de caractères alphanumériques et du caractère _ (pas de ', -, ., qui provoquent une erreur de syntaxe, ni de lettres grecques, comme φ). Enfin, rappelons que l'affectation d'une variable se fait en Python à l'aide d'un =, que beaucoup de candidats ont utilisé à tort à la place d'un == pour tester l'égalité des valeurs de deux objets.

4.1.3 Commentaires spécifiques à chaque question

Q1 - Trop de candidats oublient le `SELECT` quand ils utilisent une fonction d'agrégation. De plus, la syntaxe `1998 < ne < 2004` ne permet pas de répondre à la question ; on préférera écrire par exemple `ne > 1998 AND ne < 2004`, ce qui n'est pas équivalent en `SQL`.

Q2 - La fonction d'agrégation AVG n'est pas toujours maîtrisée, ainsi que l'usage de `GROUP BY`.

Q3 - Beaucoup de candidats ont voulu utiliser une jointure entre les tables `Rando` et `Participant` en utilisant une condition du type `ON rid = pid`, montrant ainsi un manque de compréhension de cette notion. Un produit cartésien (suivi d'une clause `WHERE`) ou l'utilisation d'une sous-requête permettaient de répondre à la question.

Q4 - Cette question était plus délicate. Une auto-jointure permettait par exemple de répondre à la question. Certains candidats ont également cherché à utiliser une sous-requête ; attention dans ce cas à la confusion entre `HAVING` et `WHERE`. De manière générale, beaucoup de candidats utilisent une syntaxe erronée de la forme `attribut.table` en lieu et place de `table.attribut`.

Q5 - Les fonctionnements de `readline`(), `readlines`() et `split`() n'ont pas été assez compris. La conversion d'une chaîne de caractères en flottant a également posé de nombreux problèmes, voire a été souvent complètement occultée. Si des progrès ont été notés concernant la gestion des fichiers texte par les candidats, des efforts sont néanmoins à poursuivre sur ce point.

Q6 - Lors d'une recherche de maximum dans une liste, l'initialisation du maximum courant à 0 est erronée.

Q7 - Le principe est souvent maîtrisé, mais la syntaxe a trop souvent été entachée d'une erreur de gestion des indices dans la boucle `for`. Les candidats sont invités à vérifier systématiquement les éventuels dépassements d'indices lors d'un parcours d'une liste. Beaucoup de candidats ont également commis une erreur de signe concernant le dénivelé négatif ; le canevas fourni en fin d'énoncé permettait pourtant de lever un quelconque doute à ce sujet.

Q8 - Cette question était assez difficile. Il fallait dans un premier temps bien comprendre le type d'un point de passage (beaucoup de candidats l'ont interprété à tort comme une liste de deux flottants, contredisant l'énoncé). La conversion (en radians pour les angles, en mètres pour les longueurs) était ensuite attendue. La prise en compte de l'altitude moyenne de l'arc a porté à confusion, ainsi que l'utilisation du théorème de Pythagore. Certains candidats ont illustré leur code d'un dessin, les aidant vraisemblablement à mieux comprendre la question.

Q9 - Question plutôt réussie, excepté l'erreur fréquente sur le type d'un point de passage.

Q10 - Beaucoup trop de candidats ont cherché à utiliser une syntaxe "à la `numpy`" dans cette question, ce qui n'est pas possible avec des listes.

Q11 - Outre la difficulté de projeter l'équation différentielle sur les deux axes, de nombreux candidats ont eu des difficultés à coder correctement la force $\vec{f_B}$. Les fonctions `uniform` ou `gauss` génèrent des valeurs différentes à chaque utilisation, ce qui a rendu faux le code de beaucoup de candidats.

Q12 - Le principe de la méthode d'Euler n'est pas toujours maîtrisé. L'enchaînement des questions incitait également à utiliser les fonctions précédentes (`vma` et `derive`).

Q13 - Le principe a globalement été compris. Les erreurs ont souvent porté sur les voisins possibles de `p` (les voisins en diagonale n'étaient pas à considérer) ainsi que sur le test de leur appartenance à atteints. Les conditionnelles utilisant `elif` ne permettaient pas de répondre correctement. La construction de la liste voulue à partir d'une liste vide [ ] et de `append` successifs est un incontournable à maîtriser.

Q14 - De nombreux candidats ont utilisé un point en trop dans leur exemple de CAE, ou oublié la moitié des cas dans le dénombrement des CAE les plus courts.

Q15 - Beaucoup de candidats ont confondu la liste vide [ ] avec la valeur spéciale `None`. Trop de copies ont également recalculé deux ou trois fois la même valeur de $\texttt{positions}_\texttt{p}\texttt{ossibles}(\texttt{p}, \texttt{atteints})$ au lieu de la stocker.

Q16 - Comme souvent, la détermination rigoureuse de la complexité (dans le cas le pire) d'une fonction a beaucoup posé problème. Trop de candidats pensent qu'une telle complexité est toujours en $O(n^k)$, où k est le nombre de boucles for présentes dans la fonction. On attendait ici que l'on détermine et somme les complexités de chaque fonction appelée dans la boucle.

Q17 - L'interprétation rigoureuse de la valeur de l'ordonnée représentée dans ce graphique a souvent posé problème. Il ne s'agissait par exemple pas d'un pourcentage.

Q18 - Les tris, ainsi que leur complexité respective, ne sont pas toujours maîtrisés. Rappelons que le tri rapide a une complexité dans le cas le pire en $O(n^2)$, alors que celle du tri-fusion est en $O(n \log n)$.

Q19 - Le principe de trier la liste `chemin` a été plutôt bien compris, et la question assez réussie.

Q20 - L'obtention des formules pour les coordonnées de l'image d'un point par une rotation a souvent été entachée d'erreur.

Q21 à 23 - Figurant en fin d'énoncé, ces questions ont permis de valoriser la prise de recul et la maîtrise des listes des meilleurs candidats.

4.2 Informatique option MP

4.2.1 Généralités

Le sujet s'intéresse à l'analyse et à la programmation du jeu du solitaire. Il est composé de deux parties : une première partie avec un jeu sur le tablier européen en un minimum de coups et une autre partie où le jeu se déroule sur un tablier unidimensionnel.
Le sujet est composé de 30 questions. Elles permettent de reconnaitre des motifs, de définir des coups simples puis composés, d'identifier des parties minimales, de reconnaître des motifs sur un tablier

A2020 – INFO

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARISTECH,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS, CHIMIE PARISTECH.**

**Concours Centrale-Supélec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.**

CONCOURS 2020

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée de l'épreuve : 1 heure 30 minutes

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI

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

INFORMATIQUE COMMUNE

L'énoncé de cette épreuve comporte 12 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.

Images de vagues et de structures

Préambule

Dans une production cinématographique en images de synthèse, les images sont crées une à une pour donner l'illusion du mouvement (sur le principe du dessin animé). Pour satisfaire les spectateurs, il est efficace de réaliser des images conformes aux équations de la physique.

Le sujet aborde la réalisation d'une scène montrant un bateau à moteur traversant un canal, créant un sillage à la surface de l'eau, et faisant osciller une gondole amarrée à proximité (figure 1).

Les programmes demandés sont à rédiger en langage `Python` 3. Si toutefois le candidat utilise une version antérieure de `Python`, il doit le préciser. Il n'est pas nécessaire d'avoir réussi à écrire le code d'une fonction pour pouvoir s'en servir dans une autre question. Les questions portant sur les bases de données sont à traiter en langage `SQL`.

Notations mathématiques et physiques

On note $\overrightarrow{V}$ un vecteur de $\mathbb{R}^3$, et `V` la variable qui lui est associée en `Python`. Dans l'ensemble du sujet, les vecteurs sont représentés par une liste de trois flottants. Par exemple, un vecteur dont les coordonnées sont $x = 2$, $y = 3$ et $z = 1{,}5$ sera exprimé par :

code Python

```
V = [2., 3., 1.5]
```

Partout où cela est nécessaire, les variables sont considérées être exprimées dans les unités SI. Le repère $(O,\overrightarrow{e_x},\overrightarrow{e_y},\overrightarrow{e_z})$ servant de référentiel est fixe par rapport au décor.

Modèle de facettes

La scène contient plusieurs objets géométriques tri-dimensionnels (3D). Chaque objet géométrique est représenté de manière numérique par un maillage. On définit les termes suivants :

- **Maillage :** ensemble des facettes qui constituent la géométrie d'un objet. Un maillage sera représenté par une liste de facettes.
- **Facette :** polygone élémentaire qui constitue une partie de la surface d'un objet. Ici, toutes les facettes seront des triangles. Une facette sera représentée par une liste ordonnée de 3 sommets.
- **Sommet** : point délimitant une facette. Il peut être commun à une ou plusieurs facettes. Tout point sera représenté par son vecteur position de coordonnées (x,y,z).

La figure 2(a) représente un exemple de maillage simple (tétraèdre), composé de 4 facettes (notées S_1 à S_4) et de 4 sommets (notés A à D) avec $AB = AD = AC = 1$. Sa représentation en `Python` est donnée dans le paragraphe I.b.

Organisation du sujet Ce sujet se compose de trois parties indépendantes.

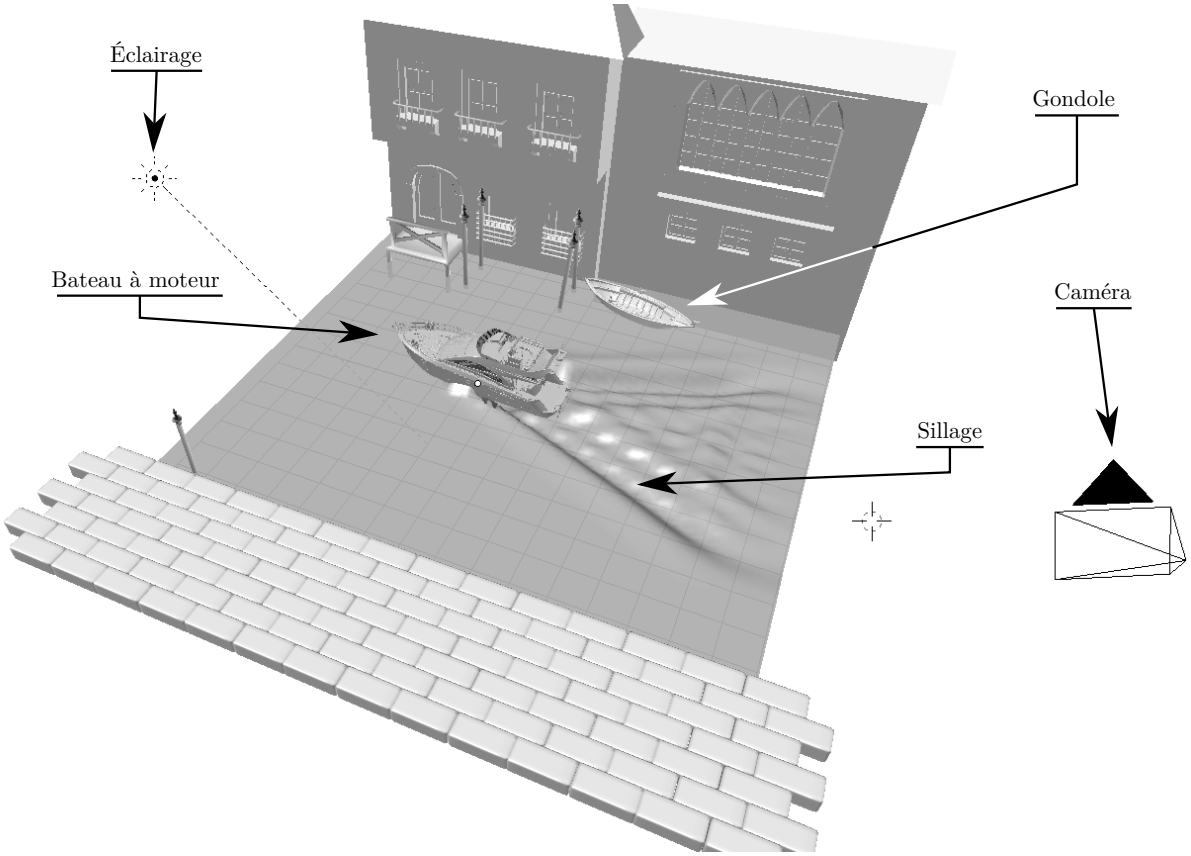
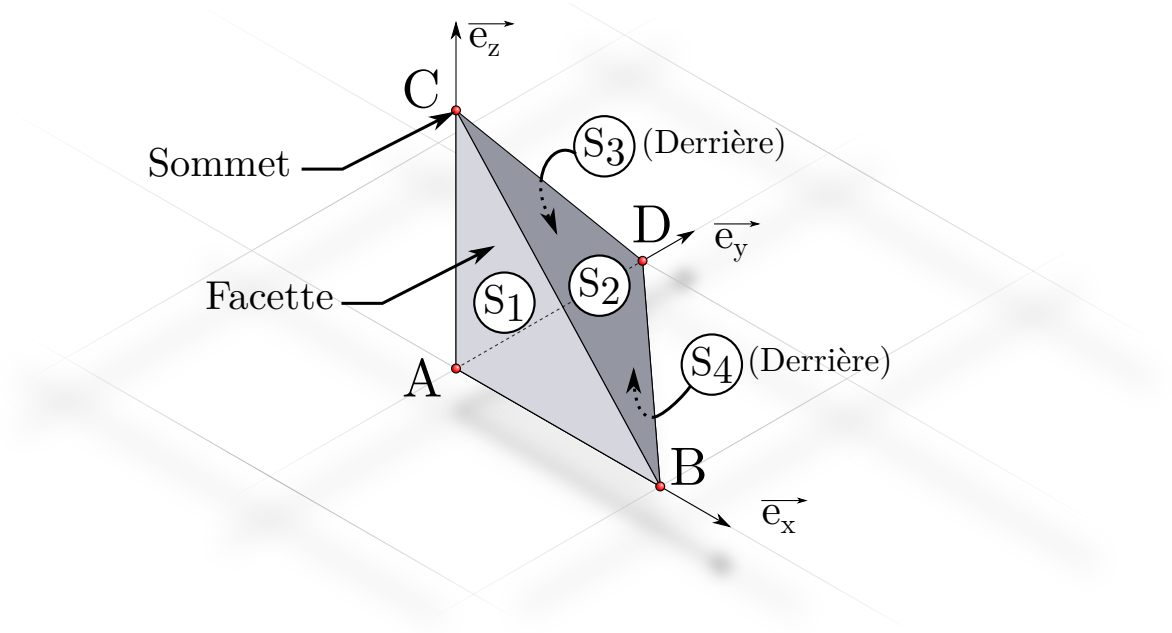
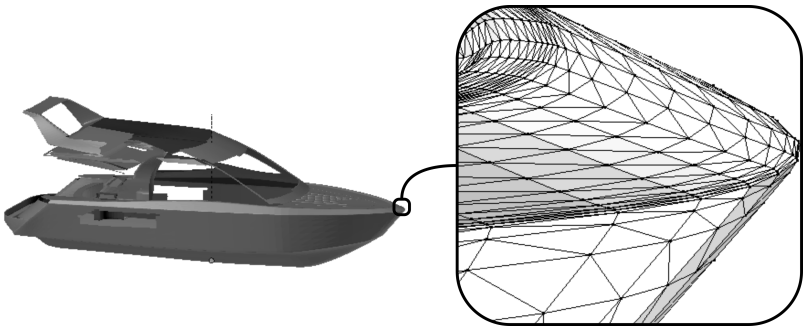


FIGURE 1 – Présentation de la scène étudiée.



(a) Un maillage simple : le tétraèdre



(b) Un maillage plus complexe : la coque d'un bateau.

FIGURE 2

Partie I. Création d'un objet dans la scène

I.a Chargement d'un modèle 3D à partir d'une base de données

On souhaite importer, dans `Python`, le modèle d'un bateau à moteur, élaboré préalablement. Ce modèle définit plusieurs maillages élémentaires (coque, bouée, échelle, moteur, etc.). Le fichier contenant ce modèle est une base de données relationnelle. Son schéma détaillé ci-dessous est illustré en figure 3 :

- relation `maillages_bateau` : ensemble des maillages. Un maillage possède un identifiant (entier) et un nom (chaîne de caractères) :

`maillages_bateau` (<u>`id`</u>,`nom`)

- relation `faces` : ensemble des facettes du modèle. Chaque facette est définie par un numéro unique, l'identifiant du maillage auquel elle appartient ainsi que les identifiants des sommets qui la composent. Tous sont des entiers.

`faces` (<u>`numero`</u>,`maillage`,`s1`,`s2`,`s3`)

- relation `sommets` : liste des sommets du modèle. Chaque sommet est défini par un identifiant (entier) et ses coordonnées dans l'espace par rapport au repère principal de la scène (flottant) :

`sommets` (<u>`id`</u>,`x`,`y`,`z`)

id	**nom**
1	coque
2	bouée
3	échelle
4	moteur
...	...

(a) Relation `maillages_bateau`

numero	**maillage**	**s1**	**s2**	**s3**
1	3	1	2	3
2	3	2	4	3
3	2	3	12	5
...	...	...	...	...

(b) Relation `faces`

id	**x**	**y**	**z**
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	1.0	0.0
...	...	...	...

(c) Relation `sommets`

FIGURE 3 – Exemples des relations de la base de données.

❑ **Q1** – ***Proposez une requête SQL permettant de compter le nombre de maillages que contient le modèle du bateau.***

❑ **Q2** – ***Proposez une requête SQL permettant de récupérer la liste des numéros des facettes (`numero`) du maillage nommé « gouvernail ».***

❑ **Q3** – ***Expliquez ce que renvoie la requête SQL suivante :***

Requête SQL

```
1 SELECT (MAX(x)-MIN(x))
2 FROM sommets AS s JOIN faces AS f JOIN  maillages_bateau AS m
3 ON (s.id=f.s1 OR s.id=f.s2 OR s.id=f.s3) AND f.maillage=m.id
4 WHERE m.nom="coque"
```

I.b Travail sur les facettes

À partir de requêtes sur la base de données précédente, on suppose avoir construit la variable `Python` suivante :

```
maillage_tetra = [ [[0.,0.,0.], [0.,0.,1.], [0.,1.,0.]],
                   [[0.,0.,0.], [0.,1.,0.], [1.,0.,0.]],
                   [[0.,0.,0.], [1.,0.,0.], [0.,0.,1.]],
                   [[1.,0.,0.], [0.,1.,0.], [0.,0.,1.]] ]
```

Cette variable représente un maillage. L'exemple illustré en figure 2(a) représente le tétraèdre (le point A a pour coordonnées (0,0,0)).

❑ **Q4 –** ***À partir de la variable*** `maillage_tetra`***, écrire une expression*** `Python` ***permettant de récupérer la coordonnée*** y ***du premier sommet de la première facette.***

❑ **Q5 –** ***À quel élément, sur la figure 2(a), correspond*** `maillage_tetra[1]` ***?***

Dans le reste du sujet, on suppose qu'on dispose d'un module `Python` dont la documentation suit.

Help on module operations_vectorielles :

NAME

operations_vectorielles

DESCRIPTION

Ce module contient des fonctions qui implémentent des opérations usuelles sur les vecteurs. Sauf indication contraire explicite les arguments sont des vecteurs passés sous la forme de liste de trois réels.

FUNCTIONS

addition(V1, V2)

Renvoie le vecteur correspondant à l'opération vectorielle V1+V2.

aire(F)

Renvoie l'aire d'une facette.
Argument :
F – une facette (liste de trois vecteurs)

prod_scalaire(V1, V2)

Renvoie le produit scalaire de V1 avec V2.

prod_vectoriel(V1, V2)

Renvoie le vecteur correspondant au produit vectoriel de V1 avec V2.

soustraction(V1, V2)

Renvoie le vecteur correspondant à l'opération vectorielle V1-V2.

barycentre(F)

Renvoie le vecteur position du barycentre d'une facette.
Argument :
F – une facette (liste de trois vecteurs)

FILE

operations_vectorielles.py

❑ **Q6** – ***On souhaite utiliser les fonctions de ce module depuis un autre fichier*** `Python`***. Complétez le code ci-dessous afin qu'il fournisse le résultat attendu :***

code Python

```
1 from ??? import ?? as ?
2 vect_1 = [1., 2., 3.]
3 vect_2 = [2., 3., 4.]
4 scal12 = ps(vect_1, vect_2) #Calcul du produit scalaire de vect_1 avec vect_2
```

Dans ce module, une fonction n'a pas été documentée :

code Python

```
1 def mystere1(V):
2     return (V[0]**2 + V[1]**2 + V[2]**2)**0.5
```

❑ **Q7** – ***Que fait la fonction*** `mystere1` ***?***

❑ **Q8** – ***Créer la fonction*** `multiplie_scalaire`***, prenant comme argument un flottant*** `a` ***et un vecteur*** `V` ***et renvoyant un nouveau vecteur correspondant à*** $a\,\overrightarrow{V}$ ***.***

La fonction `barycentre` (incomplète) est définie ci-dessous.

code Python

```
1 def barycentre(F):
2     G = [0,0,0]
3     for i in range(3):  #Pour chaque point de F
4         .....    # Ligne à compléter
5         .....    # Ligne à compléter
6     return G
```

❑ **Q9** – ***Compléter les lignes 4 et 5 permettant de calculer le barycentre.***

❑ **Q10** – ***Pour une facette*** `F`***=(A,B,C) d'aire non-nulle, proposer une fonction*** `normale`***, prenant comme argument une facette*** `F` ***et renvoyant le vecteur unitaire normal***

$$\overrightarrow{n} = \frac{\overrightarrow{AB} \wedge \overrightarrow{AC}}{\|\overrightarrow{AB} \wedge \overrightarrow{AC}\|}$$

I.c Liste des sommets

❑ **Q11** – ***Compte tenu de la représentation limitée des nombres réels en machine, deux sommets*** S_1 ***et*** S_2 ***supposés être au même endroit peuvent avoir des coordonnées légèrement différentes. Proposer une fonction*** `sont_proches`***, prenant comme arguments deux sommets*** `S1` ***et*** `S2` ***(représentés par leur vecteur position) et un flottant positif*** `eps`***, et qui renvoie*** `True` ***si*** `S1` ***et*** `S2` ***sont proches (i.e. si leur distance au sens de la norme Euclidienne est inférieure à*** `eps`***) et*** `False` ***sinon.***

Soient les fonctions suivantes :

code Python

```
1  def mystere2(S1, L):
2      for S2 in L:
3          if sont_proches(S1, S2, 1e-7):
4              return True
5      return False
6
7
8  def mystere3(maillage):
9      res = []
10     for facette in maillage:
11         for sommet in facette :
12             if not mystere2(sommet, res):
13                 res.append(sommet)
14         return res
```

❑ **Q12** – ***Sous quelle condition la fonction*** `mystere2` ***renvoie-t-elle*** `True` ***?***

❑ **Q13** – ***Donner (sans justification) ce que renvoie*** `mystere3(maillage_tetra)`***, dans le cas où*** `maillage_tetra` ***est la variable définie précédemment.***

❑ **Q14** – ***Pour une liste*** `L` ***de longueur*** n***, discuter la complexité de la fonction*** `mystere2`***. En déduire la complexité de*** `mystere3`***, pour un maillage contenant*** m ***facettes triangulaires. On distinguera le meilleur et le pire des cas.***

Partie II. Génération de vagues

Le bateau à moteur qui traverse le canal génère des vagues (appelées *sillage de Kelvin*). On ne dispose de formules analytiques décrivant ces ondes que dans des cas très simples, comme celui d'un bateau en translation rectiligne uniforme.
Le calcul numérique permettant d'évaluer la forme de ces vagues étant coûteux, il est réalisé par un programme extérieur. Ce dernier génère l'état du plan d'eau à chaque image de la scène (25 par seconde).
Pour chacune de ces images, on représente le plan d'eau par une grille régulière carrée, de taille $(m+1) \times (m+1)$ (figure 4). Chaque case (i,j) de cette grille correspond à un sommet du maillage de la surface de l'eau, noté $n_{i,j}$ de coordonnées (x_i,y_j) (avec $(i,j) \in [\![0,m]\!]^2$).
La hauteur (sur $\overrightarrow{e_z}$) de $n_{i,j}$ est notée $h_{i,j}$. L'ensemble de tous les $h_{i,j}$ est stocké dans une liste de listes nommée `mat_h`, tel que `mat_h[i][j]`$= h_{i,j}$.
Le programme extérieur calcule ainsi `mat_h` pour chaque nouvelle image. L'ensemble de toutes les valeurs de `mat_h` est stocké dans une liste nommée `liste_vagues`.

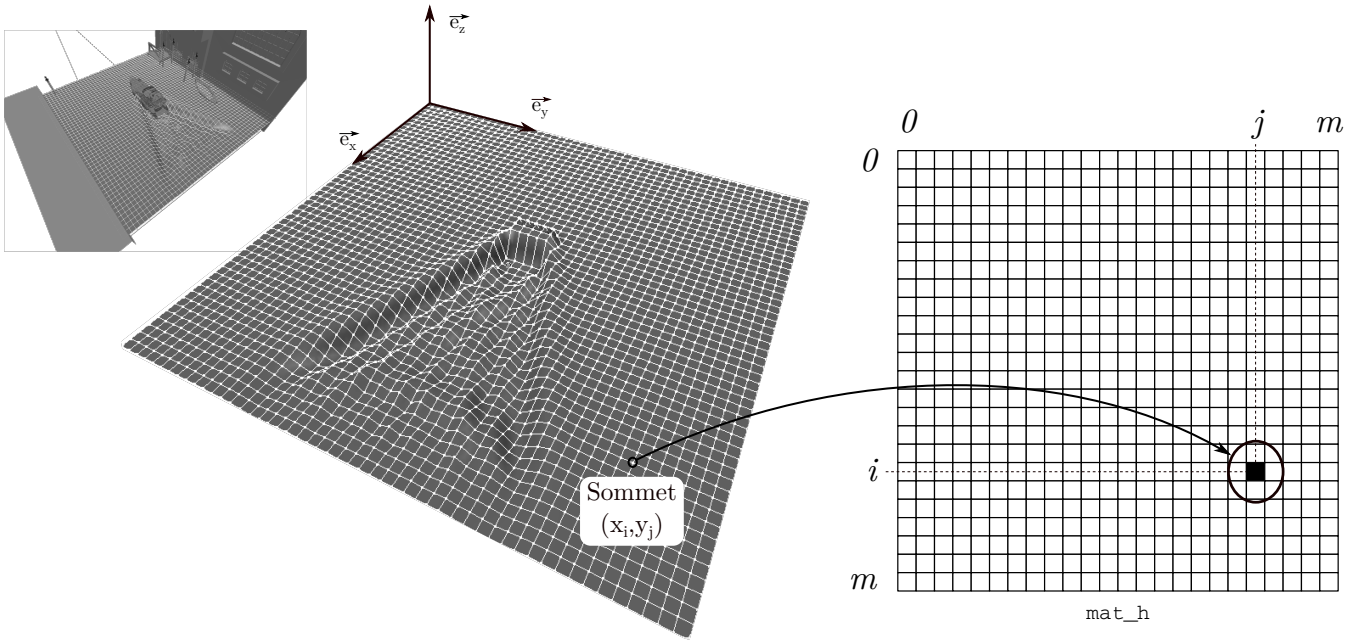


FIGURE 4 – Illustration du stockage de la hauteur d'eau dans un tableau.

La scène est composée de 350 images. Le plan d'eau est composé de 200 × 200 sommets. Chaque hauteur $h_{i,j}$ est un flottant codé sur 64 bits.

❑ **Q15** – ***Quel est l'espace occupé en mémoire vive par l'ensemble des données (en*** *Mo****).***

On souhaite enregistrer le contenu de `liste_vagues` dans un fichier afin de le transmettre au logiciel d'animation.

❑ **Q16** – ***Écrire une fonction*** `mat2str` ***qui prend en argument une liste de listes (représentant un*** `mat_h`***) et renvoie les données qu'elle contient sous forme d'une chaîne de caractères qui respecte le format suivant :***

$$\left|\begin{array}{l} h_{01};h_{02};\ \ldots\ ;h_{0m} \\ h_{11};\ \ldots\ \ \ ;h_{1m} \\ \ldots \\ h_{m0};\ \ldots\ \ \ ;h_{mm} \end{array}\right.$$

On rappelle que le retour à la ligne est codé par le caractère "\n".

❑ **Q17** – ***En s'appuyant sur*** `mat2str`***, proposer un code*** `Python` ***qui permet de sauvegarder le contenu de*** `liste_vagues` ***dans un fichier nommé*** ■ `fichier_vagues.txt` ■ ***(dans le répertoire courant), en séparant la représentation de chaque*** `mat_h` ***par deux sauts de lignes consécutifs.***

Le fichier texte obtenu est jugé trop volumineux. On décide de recourir à des matrices creuses. Dans ce qui suit on désigne par matrice creuse une matrice dont seuls la valeur et l'emplacement des éléments non-nuls (c'est-à-dire significativement éloignés de zéro) sont enregistrés.
On propose d'utiliser le format de fichier nommé « Coordinate Format » qui consiste à stocker :

- une liste `I`, comportant les numéros de ligne de chaque élément non-nul,
- une liste `J`, comportant les numéros de colonne de chaque élément non-nul,
- une liste `N`, comportant la valeur de chaque élément non-nul.

Les valeurs d'une liste sont enregistrées sur une même ligne et séparées par des points-virgules. Chaque liste est séparée des autres par un retour à la ligne. On admet que les flottants sont écrits dans le fichier avec 15 caractères (tout compris).

❑ **Q18** – ***Après avoir défini judicieusement les types des éléments contenus dans*** `I`***,*** `J` ***puis*** `N`***, estimer la taille (en octets) que prendra une matrice ayant*** p ***éléments non-nuls, au format « Coordinate Format », dans le fichier.***

❑ **Q19** – ***En déduire à partir de combien d'éléments non-nuls il devient moins avantageux d'enregistrer une matrice creuse qu'une matrice complète classique.***

❑ **Q20** – ***Proposer un code permettant de construire, pour un tableau*** `mat_h` ***donné, les listes*** `Python` `I,J` ***et*** `N`***. On considérera nulles les hauteurs inférieures à*** 10^{-3} ***(en valeur absolue).***

Partie III. Mouvement de flottaison

La gondole amarrée sur le bord du canal perçoit les vagues générées par le bateau à moteur et effectue un mouvement pseudo-oscillant conséquence de sa flottaison. On étudie ici le mouvement de translation verticale de la gondole (on ne prendra en compte ni le tangage ni le roulis). On cherche à modéliser ce mouvement en calculant à un instant donné la force exercée par le fluide sur la gondole.

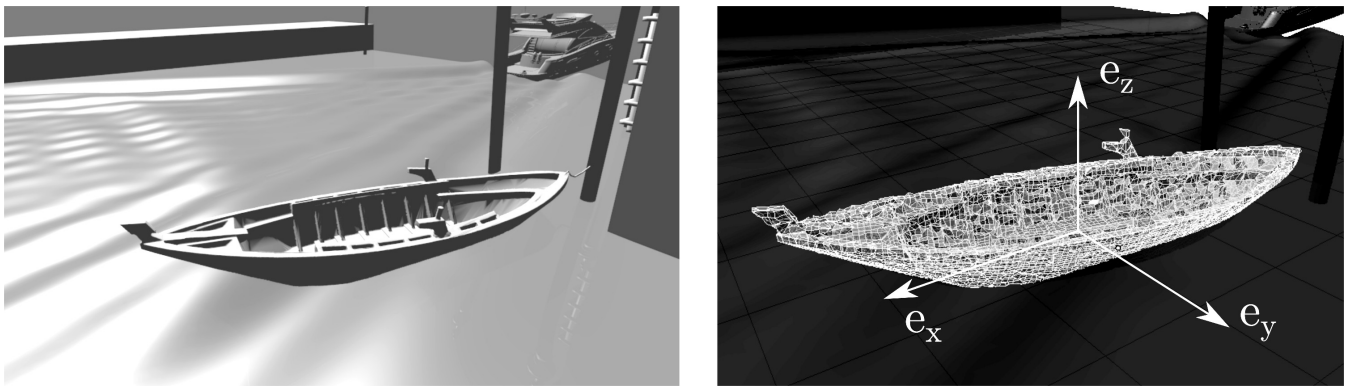


FIGURE 5 – Maillage de la gondole.

III.a Estimation de la poussée d'Archimède

Seul le mouvement de translation verticale (selon la direction verticale $\overrightarrow{e_z}$) est étudié.
On s'intéresse ici au maillage qui constitue la coque extérieure de la gondole. Certaines facettes sont émergées (*i.e.* leur barycentre est en dehors de l'eau), d'autres sont immergées (*i.e* leur barycentre est sous l'eau).

À chaque pas de temps, on suppose connue la fonction `hauteur(x,y)` qui, à tout point (x,y) du plan d'eau, associe la hauteur des vagues par rapport au repère de la scène. Le maillage composant la coque de la gondole est défini dans une liste de facette nommée `maillageG`, similaire à celle présentée dans la partie précédente.

❑ **Q21** – ***Proposer une fonction*** `lister_FI` ***prenant comme argument un maillage*** `M` ***et renvoyant la liste des facettes immergées (i.e dont le centre de gravité est sous la surface définie par*** `hauteur`***). On pourra utiliser les fonctions de la partie I.***

Pour calculer la poussée d'Archimède s'exerçant sur la coque du bateau, on doit calculer la résultante des forces dues à la pression, appliquées par l'eau sur chacune des facettes immergées.
On modélise la force appliquée par l'eau sur une facette i par :

$$\overrightarrow{F_i} = -S_i \times p(G_i)\,\overrightarrow{n_i}$$

avec :
- S_i : l'aire de la facette,
- $\overrightarrow{n_i}$: le vecteur normal sortant de la coque,
- $p(G_i)$: la pression hydrostatique de l'eau sur la facette en son barycentre G_i.

Le théorème de Pascal détermine la pression de l'eau d'un point G en fonction de sa profondeur par rapport à la surface :

$$p(x_G,y_G,z_G) = \rho \cdot g \cdot (\texttt{hauteur}(x_G,y_G) - z_G)$$

avec :
- ρ : masse volumique de l'eau ($\rho \approx 1000$)
- g : accélération de la pesanteur (ici : $g \approx 9{,}81$)

❑ **Q22** – ***Proposer une fonction*** `force_facette` ***prenant en argument une facette*** `F`***, et renvoyant le vecteur force appliqué par l'eau sur cette facette. On pourra utiliser les fonctions définies précédemment.***

La force résultante sur toute la coque s'exprime par la somme de toutes les forces appliquées sur chaque facette immergée.

❏ **Q23** – ***Définir la fonction*** `resultante` ***prenant comme argument une liste*** `L` ***de facettes (supposées immergées), renvoyant la somme des forces sur l'axe*** $\overrightarrow{z}$ ***de l'eau, appliquée sur l'ensemble des surfaces.***

III.b Tri des facettes

On cherche à optimiser l'efficacité de la fonction `resultante` qui devra être utilisée intensément pour réaliser un grand nombre d'images. On remarque que la taille des facettes n'est pas homogène : la coque est composée de grandes facettes et de petites facettes. Les petites facettes représentent souvent des détails d'intérêt graphique n'apportant qu'une très faible contribution à la résultante des forces hydrostatiques.
Ainsi, une étude montre que la moitié des facettes représente à elle seule 99% de la surface totale de la coque. Pour alléger le processus, on souhaite donc trier les facettes par aire décroissante, afin de n'appliquer les calculs de la poussée d'Archimède qu'à la moitié d'entre elles (les plus grandes). On propose le code (incomplet) du tri-fusion ci-dessous :

code Python

```
1  def fusion(L1, L2):
2      # À compléter (sur une ou plusieurs lignes)
3
4  def trier_facettes(L):
5      # À compléter (sur une ou plusieurs lignes)
6
7  grandesFacettes = # À compléter
```

❏ **Q24** – ***Compléter la fonction*** `fusion`***, prenant comme argument deux listes de facettes*** `L1` ***et*** `L2` ***(supposée chacune triée par aire décroissante) et renvoyant une nouvelle liste composée des facettes de*** `L1` ***et*** `L2` ***triées par aire décroissante.***

❏ **Q25** – ***Compléter la fonction récursive*** `trier_facettes`***, prenant comme argument une liste de facettes*** `L`***, et renvoyant une nouvelle liste de facettes triées dans l'ordre des aires décroissantes, par la méthode du tri-fusion.***

❏ **Q26** – ***Affecter à une nouvelle variable*** `grandesFacettes` ***la liste des facettes de*** `maillageG`***, privée de la moitié des facettes les plus petites (en cas de nombre impair d'éléments, on inclura la facette médiane).***

III.c Mouvement vertical de la gondole

La gondole est attachée à un repère local dont l'origine est son centre de gravité (de coordonnées (x_G,y_G,z_G) et de vitesse verticale notée v) par rapport au décor. Le principe fondamental de la dynamique en projection sur l'axe vertical $\overrightarrow{e_z}$, appliqué à la gondole énonce que :

$$\frac{dv}{dt} = \frac{1}{m} \times F_{eau \mapsto gondole} - g$$
$$\frac{dz_G}{dt} = v$$

avec

- m : masse de la gondole
- $F_{eau \mapsto gondole}$ est la résultante des forces appliquées par l'eau sur la gondole (renvoyée par la fonction `resultante`, vue précédemment).

La position initiale de la gondole est $z_{G0} = 0$. Sa vitesse verticale initiale est $v_0 = 0$.

On souhaite estimer le mouvement par la méthode d'Euler. Pour ce faire, on utilise la fonction `nouvelle_hauteur` avant d'afficher chaque nouvelle image. Cette fonction a pour but de recalculer la hauteur (et la vitesse) de la gondole pour un nouveau pas de temps. Elle prend trois arguments :

- `posG` contiendra le vecteur position actuel du centre de gravité de la gondole au moment de l'appel (liste de trois flottants) ;
- `vitG` contiendra le vecteur vitesse actuel de ce même point au moment de l'appel (liste de trois flottants) ;
- `mailG` contiendra la liste des grandes facettes de la gondole (privée des petites, au sens de la question précédente), au moment de l'appel.

code Python

```
1 def nouvelle_hauteur(posG, vitG, mailG):
2     dt=1.0/25.0 # Pas de temps correspondant à une image du film.
3     facettes_immergees = lister_FI(mailG)
4     posG = posG + ..........    # à compléter
5     vitG = vitG + ...........   # à compléter
6     return posG, vitG
```

❑ **Q27** – ***Compléter les lignes 4 et 5 du code précédent conformément à la méthode d'Euler.***

FIN DE L'ÉPREUVE

5 Informatique

5.1 Informatique pour tous

Le sujet d'informatique commune portait cette année sur des techniques algorithmiques autour du thème de la discrétisation spatiale en facettes d'une scène de cinéma.
L'épreuve abordait, comme toujours, un large spectre des notions vues durant les deux années de préparation des candidats.

5.1.1 Remarques générales

- Les copies sont très contrastées et l'épreuve a bien joué son rôle de classement, il y a en particulier un nombre non négligeable de copies qui montrent une bonne maîtrise du langage et une compréhension relativement bonne, et parfois excellente, des problématiques abordées.
- Certaines copies sont trop proches d'un brouillon, avec beaucoup de ratures réalisées sans soin. Si l'on est capable de comprendre qu'un candidat se reprenne sur une question particulière ou une ligne de code, il ne nous semble pas acceptable de croiser des copies contenant de nombreuses ratures sales sur toute la copie. Les candidats ont eu un temps de formation conséquent pour cadrer leurs productions écrites et doivent respecter le correcteur dans la mise en forme de leurs copies.
- Nous avons constaté un phénomène gênant qui prend de l'ampleur : certaines parenthèses, pourtant indispensables, tendent à disparaître. On a souvent lu
  `for i in range m :` ou encore `m = len L` par exemple. Si on peut le laisser passer une fois, nous l'avons sanctionné lorsque c'était trop redondant dans la copie.
- La première partie du sujet définissait des opérations algébriques simples sur les vecteurs. Il pouvait être tentant de dévier vers des réflexes propres à l'utilisation du module `numpy` (par exemple la multiplication d'un scalaire par un vecteur ?), mais il fallait résister à cette tentation : on rappelle en particulier qu'une opération de type `a*L` avec a un flottant et `L` une liste n'a pas de sens en python. Les candidats sont donc invités à réfléchir attentivement aux nombreuses différences entre les listes et les tableaux numériques (`ndarrays`).
- Une remarque déjà faite les années précédentes, mais que nous pensons nécessaire de renouveler : la syntaxe `L=L+[a]` est beaucoup moins efficace pour ajouter un élément à la fin d'une liste que `L.append(a)`. Il serait souhaitable que les étudiants privilégient systématiquement la deuxième solution au cours de leur formation.

5.1.2 Commentaires spécifiques à chaque question

Q1 - Trop de candidats ne maitrisent pas la syntaxe d'une fonction d'agrégation, confondent `COUNT` avec `SUM`, ou n'utilisent pas la bonne relation.

Q2 - Mieux réussie que **Q1**, sans doute parce que l'énoncé donnait une exemple de jointure à la question suivante.

Q3 - Assez bien réussie, mais il manque souvent une information : selon l'axe x, ou la mention d'une largeur/longueur/taille.

Q4 - Une simple expression était demandée, pas une fonction. On a pu constater quelques confusions avec la syntaxe numpy pour les indices (`L[0,0,1]` ne fonctionne pas pour une liste).

Q5 - L'indexage du maillage ne correspondait pas à la numérotation des facettes sur la figure, il fallait donc être attentif et ne pas répondre `S2` trop vite.

Q6 - L'import d'une fonction d'un module est une syntaxe importante et doit être maîtrisée.

Q7 - On peut au choix se féliciter que 95% des candidats sachent reconnaître le calcul de la norme d'un vecteur, ou s'inquiéter que 5% des candidats n'y parviennent pas ?

Q8 - Cette question d'apparence simple contenait pourtant un certain nombre de pièges. En particulier, on demandait explicitement une nouvelle liste : comme les listes sont mutables, on ne pouvait pas écrire `V[i]=a*V[i]` par exemple, qui modifiait la liste en argument. On ne peut pas non plus utiliser de syntaxe « à la numpy » comme `return a*V`. Enfin, un certain nombre de candidats tombent dans le piège classique d'initialiser une nouvelle liste vide (`W=[]`) puis tentent de la remplir avec `W[i]=` ?

Q9 - La définition du barycentre, explicitement au programme de physique de toutes les filières, n'est pas toujours maitrisée.

Q10 - Il y avait deux pièges sur cette question : il fallait éviter de recoder à la main une ou plusieurs fonctions données dans le module `operations_vectorielles` (surtout la soustraction), et on ne peut pas non plus utiliser de syntaxe « à la numpy » comme `return n/norme`.

Q11 - Trop de candidats recodent à la main la fonction soustraction. On peut également souhaiter une syntaxe plus élégante ; on rappelle que la syntaxe suivante :

```
if bool :
return True
else :
return False
```

peut s'écrire plus simplement : `return bool`
Enfin, même s'il ne s'agit pas d'une épreuve de mathématiques, nous avons été surpris par le nombre de candidats confondant la norme d'une différence avec la différence des normes.

Q12 - On a pu lire un nombre incalculable de fois « la fonction renvoie `True` si tous les sommets de `L` sont proches de `S1` », ce qui témoigne d'une mauvaise compréhension de l'algorithmique de base. De même, « la fonction renvoie `True` si `S1` et `S2` sont proches » n'était pas une réponse acceptable car le deuxième paramètre de la fonction était une liste `L` qu'il fallait impérativement évoquer.

Q13 - Il fallait donner explicitement ce qui était renvoyé ; attention au type (beaucoup d'oublis du point pour les flottants). Nous avons bien entendu fait preuve de tolérance compte tenu de l'erreur d'indentation de l'énoncé, et accepté les réponses tenant compte ou pas de cette erreur. Nous demandions seulement la cohérence avec l'analyse de complexité à la question suivante.

Q14 - Que de réponses affirmées sans aucune justification ! Il est capital de justifier proprement une analyse de complexité. Ici, il est attendu une description explicite du meilleur et du pire des cas avant la justification. Cette distinction a souvent été oubliée pour `mystere2`. De plus, trop de candidats justifient leurs complexités par le simpliste (et faux) « il y a une boucle `for` donc c'est de complexité linéaire » ? Il fallait remarquer que la complexité de `mystere3` ne pouvait dépendre que de `m` (taille de l'entrée) et pas de `n`, caractéristique d'une variable intermédiaire. Cette étude a très rarement été bien réalisée, erreur d'indentation ou pas. Enfin, on a encore retrouvé dans les copies quelques complexités en $O(n!)$, $O(3n)$ ou encore $O(1/n)$ sans que cela paraisse gêner les candidats qui les proposent.

Q15 - L'application numérique est régulièrement fausse (même avec une certaine tolérance), alors que le calcul posé est correct.

Q16 - La gestion des chaînes de caractères n'est pas maitrisée : il s'agit d'un objet non mutable (on ne peut pas écrire `ch[-1]="\n"` par exemple), pas de append, conversion en chaîne de caractères à l'aide de str non connue (on a lu un nombre incalculable de fois `"mat_h[i][j]"` pour cette conversion). Les candidats sont encouragés à s'entraîner sur la maitrise de ce type de variable.

Q17 - L'ouverture et l'écriture dans un fichier n'est pas maîtrisée.

Q18 et 19 - Très peu de candidats ont compris la question (ou lu le paragraphe précédent).

Q20 - La valeur absolue a souvent été oubliée. On ne demandait pas une fonction, mais un script/code.

Q21 - La condition d'immersion n'a pas toujours été comprise (on attendait notamment le calcul de la hauteur des vagues au barycentre de la face). Beaucoup de candidats ont également calculé trois fois

le barycentre pour en extraire ses trois coordonnées, ce que nous avons sanctionné puisqu'une solution simple et plus économe existait.

Q22 - Assez bien réussie ; on attendait des candidats qu'ils réinvestissent les fonctions définies précédemment.

Q23 - L'opérateur + n'est pas adapté pour les listes (il les concatène). La projection sur l'axe vertical a parfois été oubliée.

Q24 - On a pu croiser certaines structures en boucles imbriquées :

```
for i in range(len(L1) :
for j in range(len(L2)) :
...
```

qui témoignent d'une mauvaise compréhension de l'algorithme de fusion de deux listes.
De manière générale, cet algorithme classique de cours devrait être mieux maitrisé par les candidats. Parmi ceux qui connaissaient l'algorithme, beaucoup ont oublié de comparer les aires des éléments (facettes) et pas simplement les éléments eux-mêmes. Enfin, la version récursive de la fusion de deux listes nous semble à éviter absolument, car très peu efficace : nous encourageons les candidats à bien maitriser (et présenter) la version itérative.

Q25 - Le cas de base a souvent été oublié ou mal traité.

Q26 - Le principe (tri puis extraction) a plutôt été maitrisé, mais la condition sur le nombre de facettes a posé de nombreux problèmes. On pouvait très facilement vérifier soi-même la cohérence de l'expression sur de petites valeurs (`len(maillageG)=2` et `len(maillageG)=3`) pour l'inclusion de la médiane.

Q27 - Question assez simple si on avait bien compris le sujet et le principe de la méthode d'Euler. Nous avons fait preuve de tolérance concernant le type de variable utilisé (scalaire ou vecteur) en tenant compte de l'ambiguïté de l'énoncé.

5.2 Informatique option MP

5.2.1 Généralités

Le sujet s'inscrit dans le domaine de la compression de données. Il traite particulièrement le cas du comptage efficace d'occurrences de symboles (codes Unicode) dans un texte, puis comment coder, à l'aide d'un encodeur efficace, un texte dans une suite de bits.
Le sujet comporte 33 questions. Même s'il traite d'un unique problème de bout en bout, le sujet est découpé en deux parties indépendantes. Les candidats ont bien pris en compte cette séparation et ont bien su gérer le passage de l'une à l'autre en vue de répondre au maximum de questions.
En général, les candidats ont bien compris le problème posé. Ils répondent aussi bien aux questions de programmation et de complexité qu'aux questions portant sur des démonstrations et/ou des justifications rigoureuses.
Enfin, les correcteurs ont noté, avec satisfaction, qu'il y avait une diminution importante de compositions traitant exclusivement les questions portant sur l'écriture de programmes ou les questions portant sur des démonstrations.

5.2.2 Analyse Globale

Les candidats ont une bonne compréhension des structures complexes mises en œuvre dans l'énoncé du problème. Ils font un usage de la récursivité souvent clair et efficace. Le niveau des réponses et des copies est satisfaisant voire bon. Pour les questions programmation, les correcteurs ont pris en

A2019 – INFO

ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT Atlantique, ENSAE PARISTECH,
CHIMIE PARISTECH.

Concours Centrale-Supélec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.

CONCOURS 2019

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée de l'épreuve : 1 heure 30 minutes

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI

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

INFORMATIQUE COMMUNE

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.

Autour des nombres premiers

Préambule

Chiffrer les données est nécessaire pour assurer la confidentialité lors d'échanges d'informations sensibles. Dans ce domaine, les nombres premiers servent de base au principe de clés publique et privée qui permettent, au travers d'algorithmes, d'échanger des messages chiffrés. La sécurité de cette méthode de chiffrement repose sur l'existence d'opérations mathématiques peu coûteuses en temps d'exécution mais dont l'inversion (c'est-à-dire la détermination des opérandes de départ à partir du résultat) prend un temps exorbitant. On appelle ces opérations « fonctions à sens unique ». Une telle opération est, par exemple, la multiplication de grands nombres premiers. Il est aisé de calculer leur produit. Par contre, connaissant uniquement ce produit, il est très difficile de déduire les deux facteurs premiers.

Le sujet étudie différentes questions sur les nombres premiers.

Les programmes demandés sont à rédiger en langage `Python` 3. Si toutefois le candidat utilise une version antérieure de Python, il doit le préciser. Il n'est pas nécessaire d'avoir réussi à écrire le code d'une fonction pour pouvoir s'en servir dans une autre question. Les questions portant sur les bases de données sont à traiter en langage `SQL`.

Définitions, rappels et notations

- Un nombre premier est un entier naturel qui admet exactement deux diviseurs : 1 et lui-même. Ainsi 1 n'est pas considéré comme premier.
- Un flottant est la représentation d'un nombre réel en mémoire.
- Quand une fonction `Python` est définie comme prenant un « nombre » en paramètre cela signifie que ce paramètre pourra être indifféremment un flottant ou un entier.
- On note $\lfloor x \rfloor$ la partie entière de x.
- `abs(x)` renvoie la valeur absolue de `x`. La valeur renvoyée est du même type de données que celle en argument.
- `int(x)` convertit vers un entier. Lorsque x est un flottant positif ou nul, elle renvoie la partie entière de x, c'est-à-dire l'entier n tel que $n \leqslant x < n + 1$.
- `round(x)` renvoie la valeur de l'entier le plus proche de `x`. Si deux entiers sont équidistants, l'arrondi se fait vers la valeur paire.
- `floor(x)` renvoie la valeur du plus grand entier inférieur ou égal à `x`.
- `ceil(x)` renvoie la valeur du plus petit entier supérieur ou égal à `x`.
- `log(x)` renvoie sous forme de flottant la valeur du logarithme népérien de `x` (supposé strictement positif).
- `log(x,n)` renvoie sous forme de flottant la valeur du logarithme de `x` en base `n`.
- La fonction `time()` du module `time` renvoie un flottant représentant le nombre de secondes depuis le 01/01/1970 avec une résolution de 10^{-7} seconde (horloge de l'ordinateur).
- L'opérateur usuel de division `/` renvoie toujours un flottant, même si les deux opérandes sont des multiples l'un de l'autre.
- L'infini $+\infty$ en `Python` s'écrit `float("inf")`.
- En `Python` 3, on peut utiliser des entiers illimités de plus de 32 bits avec le type `long`.

Partie I. Préliminaires

❑ **Q1** – Dans un programme `Python` on souhaite pouvoir faire appel aux fonctions `log`, `sqrt`, `floor` et `ceil` du module `math` (`round` est disponible par défaut). Écrire des instructions permettant d'avoir accès à ces fonctions et d'afficher le logarithme népérien de 0.5.

❑ **Q2** – Écrire une fonction `sont_proches(x, y)` qui renvoie `True` si la condition suivante est remplie et `False` sinon

$$|x - y| \leqslant atol + |y| \times rtol$$

où *atol* et *rtol* sont deux constantes, à définir dans le corps de la fonction, valant respectivement 10^{-5} et 10^{-8}. Les paramètres `x` et `y` sont des nombres quelconques.

❑ **Q3** – On donne la fonction `mystere` ci-dessous. Que renvoie `mystere(1001,10)` ? Le paramètre `x` est un nombre strictement positif et `b` un entier naturel non nul.

```
1 def mystere(x,b):
2     if x < b:
3         return 0
4     else:
5         return 1 + mystere(x / b, b)
```

❑ **Q4** – Exprimer ce que renvoie `mystere` en fonction de la partie entière d'une fonction usuelle.

❑ **Q5** – On donne le code suivant :

```
1 pas = 1e-5
2
3 x2 = 0
4 for i in range(100000):
5     x1 = (i + 1) * pas
6     x2 = x2 + pas
7
8 print("x1:", x1)
9 print("x2:", x2)
```

L'exécution de ce code produit le résultat :

```
x1: 1.0
x2: 0.9999999999980838
```

Commenter.

Partie II. Génération de nombres premiers

II.a Approche systématique

Le crible d'Ératosthène est un algorithme qui permet de déterminer la liste des nombres premiers appartenant à l'intervalle $[\![1,n]\!]$. Son pseudo-code s'écrit comme suit :

Données : N, entier supérieur ou égal à 1
Résultat : *liste_bool*, liste de booléens
début
  liste_bool ⟵ liste de N booléens initialisés à Vrai;
  Marquer comme Faux le premier élément de *liste_bool*;
  pour *entier* $i \leftarrow 2$ **à** $\lfloor\sqrt{N}\rfloor$ **faire**
    si *i n'est pas marqué comme Faux dans liste_bool* **alors**
      Marquer comme Faux tous les multiples de i différents de i dans *liste_bool* ;
    fin
  fin
  retourner *liste_bool*
fin

Algorithme 1 : Crible d'Ératosthène

À la fin de l'exécution, si un élément de `liste_bool` vaut Vrai alors le nombre codé par l'indice considéré est premier. Par exemple pour `N=4` une implémentation `Python` du crible renvoie `[False True True False]`.

❑ **Q6** – Sachant que le langage `Python` traite les listes de booléens comme une liste d'éléments de 32 bits, quel est (approximativement) la valeur maximale de `N` pour laquelle `liste_bool` est stockable dans une mémoire vive de 4 Go ?

❑ **Q7** – Quel facteur peut-on gagner sur la valeur maximale de `N` en utilisant une bibliothèque permettant de coder les booléens non pas sur 32 bits mais dans le plus petit espace mémoire possible pour ce type de données (on demande de le préciser) ?

❑ **Q8** – Écrire la fonction `erato_iter(N)` qui implémente l'algorithme 1 pour un paramètre `N` qui est un entier supérieur ou égal à 1.

❑ **Q9** – Quelle est la complexité algorithmique du crible d'Ératosthène en fonction de N ? On admettra que :

$$\sum_{p<N, p\ premier} \frac{1}{p} \simeq \ln(\ln(N)) \tag{1}$$

La réponse devra être justifiée.

❑ **Q10** – Quand on traite des nombres entiers il est intéressant d'exprimer la complexité d'un algorithme non pas en fonction de la valeur N du nombre traité mais de son nombre de chiffres n. Donner une approximation du résultat de la question précédente en fonction de n en précisant la base choisie.

II.b Génération rapide de nombres premiers

L'approche systématique qui précède est inefficace car elle revient à attendre d'avoir généré la liste de tous les nombres premiers inférieurs à une certaine valeur pour en choisir ensuite quelques uns au hasard. Une meilleure idée est d'utiliser des tests probabilistes de primalité. Ces tests ne garantissent pas vraiment qu'un nombre est premier. Cependant, au sens probabiliste, si un nombre

réussit un de ces tests alors la probabilité qu'il ne soit pas premier est prouvée être inférieure à un seuil calculable.
En suivant cette idée, une nouvelle approche est la suivante :

1. générer un entier pseudo-aléatoire (voir ci-dessous)
2. vérifier si cet entier a de fortes chances d'être premier
3. recommencer tant que le résultat n'est pas satisfaisant.

Pour générer un entier pseudo-aléatoire A on se base sur un certain nombre d'itérations de l'algorithme Blum Blum Shub, décrit comme suit. On initialise A à zéro au début de l'algorithme et pour chaque itération ($i \geq 1$) on calcule :

$$x_i = \text{reste de la division euclidienne de } x_{i-1}^2 \text{ par } M \tag{2}$$

où M est le produit de deux nombres premiers quelconques et x_0 une valeur initiale nommée « graine » choisie aléatoirement. On utilise ici l'horloge de l'ordinateur comme source pour x_0.
Puis, pour chaque x_i, s'il est impair, on additionne 2^i à A.

❑ **Q11** – On répète (2) pour i parcourant $[\![1,N-1]\!]$, quelle sera la valeur de A si x_i est impair à chaque itération ?

❑ **Q12** – Compléter (avec le nombre de lignes que vous jugerez nécessaire) la fonction `bbs(N)` donnée ci-dessous qui réalise ces itérations. La graine est un entier représentant la fraction de secondes du temps courant, par exemple 1528287738.7931523 donne la graine 7931523. Le paramètre `N` est un entier non nul.

```
1  ...
2  ...
3  def bbs(N):
4      p1 = 24375763
5      p2 = 28972763
6      M = p1 * p2
7      # calculer la graine
8      ... (à compléter)
9      A = 0
10     for i in range(N):
11         if ... (à compléter) # si xi est impair
12             A = A + 2**i
13         # calculer le nouvel xi
14         xi = ... (à compléter)
15     return(A)
```

Le test probabiliste de primalité le plus simple est le test de primalité de Fermat. Ce test utilise la contraposée du petit théorème de Fermat qu'on peut évoquer comme suit : si $a \in [\![2, p-1]\!]$ est premier et que le reste de la division euclidienne de a^{p-1} par p vaut 1, alors il y a de "fortes" chances pour que p soit premier.

❑ **Q13** – En combinant les résultats du test de primalité de Fermat pour $a = 2$, $a = 3$, $a = 5$ et $a = 7$, écrire une fonction `premier_rapide(n_max)` qui renvoie un nombre aléatoire inférieur strictement à *n_max* qui a de fortes chances d'être premier. Le paramètre `n_max` est un entier supérieur à 12.

❑ **Q14** – On souhaite caractériser le taux d'erreurs de `premier_rapide`.
Écrire une fonction `stats_bbs_fermat(N, nb)` qui contrôle pour `nb` nombres, inférieurs ou égaux à `N`, générés par `premier_rapide`, qu'ils sont réellement premiers. Cette fonction renvoie le taux relatif d'erreur ainsi que la liste des faux nombres premiers trouvés. Les paramètres `N` et `nb` sont des entiers strictement positifs.

Partie III. Compter les nombres premiers

La question de la répartition des nombres premiers a été étudiée par de nombreux mathématiciens, dont Euclide, Riemann, Gauss et Legendre. On étudie dans cette partie les propriétés de la fonction $\pi(n)$, qui renvoie le nombre de nombres premiers appartenant à $[\![1,n]\!]$.

III.a Calcul de $\pi(n)$ via un crible

❑ **Q15** – Écrire une fonction `Pi(N)` qui calcule la valeur exacte de $\pi(n)$ pour tout entier n de $[\![1,N]\!]$. Les nombres premiers sont déduits de la liste `liste_bool` renvoyée par la fonction `erato_iter` de la question 8. On demande que `Pi(N)` renvoie son résultat sous la forme d'une liste de `[n, `$\pi(n)$`]`. Par exemple `Pi(4)` renvoie la liste [[1, 0], [2, 1], [3, 2], [4, 2]].
Un seul appel à `erato_iter` est autorisé et on exige une fonction dont la complexité, en dehors de cet appel, est linéaire en fonction de `N`. Le paramètre `N` est un entier supérieur à 1.

Il a été prouvé que $\frac{n}{\ln(n)-1} < \pi(n)$ pour tout $n \geqslant 5393$. On souhaite vérifier cette inégalité en se basant sur la fonction `Pi(N)` écrite en Question 15.

❑ **Q16** – Écrire une fonction `verif_Pi(N)` qui renvoie `True` si l'inégalité est vérifiée jusqu'à `N` inclus, `False` sinon. Le paramètre `N` est un entier supposé supérieur ou égal à 5393.

III.b Calcul d'une valeur approchée de $\pi(n)$

Le calcul de $\pi(n)$ dépend de la capacité à calculer de manière exhaustive tous les nombres premiers de $[\![1,N]\!]$, or le temps nécessaire à ce calcul devient rapidement très grand lorsque N augmente. Il existe en revanche diverses méthodes pour calculer une valeur approchée de $\pi(n)$. Une méthode utilise la fonction logarithme intégral li, dont une représentation graphique est fournie en figure 1, et qui est définie comme :

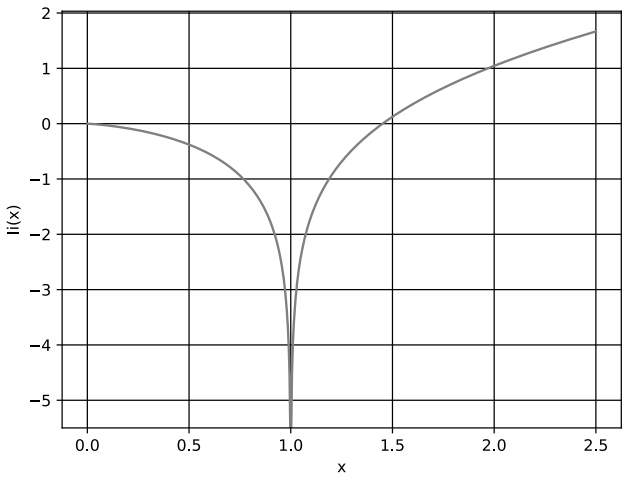


FIGURE 1 – Allure de li sur [0,2.5]

$$\begin{aligned} \mathrm{li} : \mathbb{R}^+ \setminus \{1\} &\to \mathbb{R} \\ x &\mapsto \int_0^x \frac{\mathrm{dt}}{\ln(t)} \end{aligned} \tag{3}$$

Si $x > 1$, l'intégrale impropre doit être interprétée comme sa valeur principale de Cauchy qui est définie comme :

$$\mathrm{li}(x) = \lim_{\varepsilon \to 0^+} \left(\int_0^{1-\varepsilon} \frac{\mathrm{dt}}{\ln(t)} + \int_{1+\varepsilon}^{x} \frac{\mathrm{dt}}{\ln(t)} \right) \tag{4}$$

L'intérêt de li pour compter les nombres premiers vient de la propriété suivante :

$$\lim_{x \to \infty} \frac{\pi(\lfloor x \rfloor)}{\mathrm{li}(x)} = 1 \tag{5}$$

On souhaite développer un programme permettant de calculer une valeur approchée de li. On compare ensuite les résultats obtenus à une implémentation de référence qui est nommée `ref_li`, réputée très précise.

Estimation de li par quadrature numérique

On choisit d'utiliser la méthode des rectangles à droite. On appelle *pas* la base des rectangles. La figure 4 illustre cette méthode appliquée au calcul de la valeur principale définie équation (4).
Par souci de simplification, on suppose que *pas* est choisi de manière à ce que 1 et x soient multiples de *pas* et on utilise $\varepsilon = pas$ dans l'équation (4).

❏ **Q17** – La fonction qui est évaluée sur l'intervalle d'intégration est supposée avoir une complexité constante quelles que soient ses valeurs d'entrée. Quelle est la complexité en temps de la méthode des rectangles à droite ? On prendra soin d'expliciter en fonction de quelle variable d'entrée cette complexité est exprimée.

❏ **Q18** – Dans les mêmes conditions d'évaluation, quelle est la complexité en temps de la méthode des rectangles centrés ? Donner aussi celle de la méthode des trapèzes.

❏ **Q19** – Écrire une fonction `inv_ln_rect_d(a, b, pas)` qui calcule par la méthode des rectangles à droite une valeur approchée de $\int_a^b \frac{\mathrm{dt}}{\ln(t)}$ avec un incrément valant `pas`. On suppose dans cette question que $a < b$ et que 1 n'appartient pas à l'intervalle $[a, b]$ de sorte que la fonction intégrée est définie et continue sur $[a, b]$.
On considère que le réel $b - a$ est un multiple du réel *pas*.
Les paramètres `a`, `b` et `pas` sont des flottants.

❏ **Q20** – Écrire une fonction `li_d(x, pas)` qui calcule une valeur approchée de $\mathrm{li}(x)$ avec la méthode des rectangles à droite en se basant sur `inv_ln_rect_d`. Si $x = 1$ la fonction renvoie $-\infty$.
On rappelle qu'on suppose que *pas* est choisi de manière à ce que 1 et x soient multiples de *pas* et qu'on utilise $\varepsilon = pas$ dans l'équation (4). Les paramètres `x` et `pas` sont des flottants.

Analyse des résultats de `li_d`

Après avoir testé `li_d` on obtient plusieurs résultats surprenants.

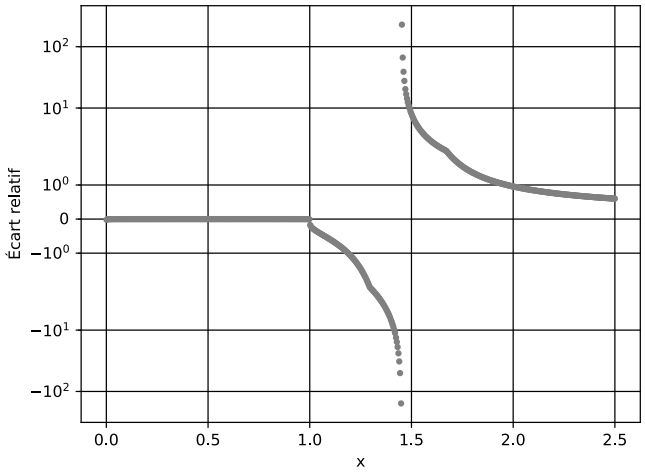


FIGURE 2 – Écart relatif entre `li_d` ($pas = 10^{-4}$) et l'implémentation de référence `ref_li`.

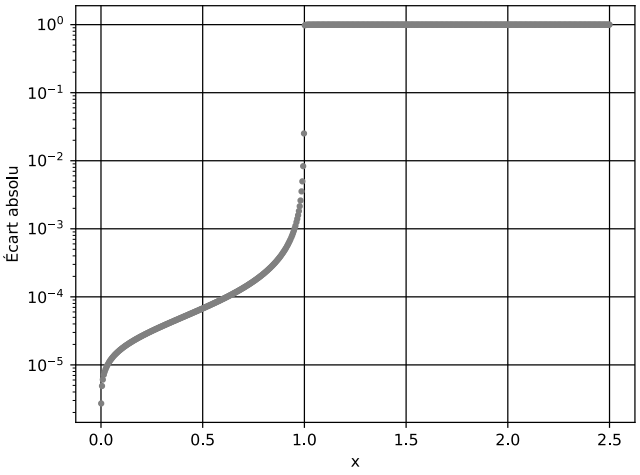


FIGURE 3 – Écart absolu entre `li_d` ($pas = 10^{-4}$) et l'implémentation de référence `ref_li`.

❏ **Q21** – Expliquer le comportement de l'écart relatif entre `li_d` et `ref_li`, illustré figure 2 au voisinage de $x \simeq 1.4$.

❑ **Q22** – On constate un écart absolu important entre `li_d` et `ref_li` au delà de $x = 1$, illustré figure 3. Expliquer succinctement d'où vient ce phénomène. On ne demande pas une démonstration mathématique rigoureuse.
Pour répondre à cette question on pourra remarquer qu'au premier ordre $\frac{1}{ln(1+\varepsilon)} \simeq -\frac{1}{ln(1-\varepsilon)}$ quand $\varepsilon \to 0$ et s'interroger sur la valeur que devrait avoir l'intégrale impropre de $\frac{1}{\ln(x)}$ sur un intervalle $[1-\varepsilon, 1+\varepsilon]$ avec $\varepsilon \ll 1$. Une analyse géométrique de la figure 4 peut aussi s'avérer utile.

❑ **Q23** – Proposer, en justifiant votre choix, une ou des modifications de l'algorithme utilisé afin d'éliminer le problème constaté sur l'écart absolu. Il n'est pas demandé d'écrire le code mettant en œuvre ces propositions.

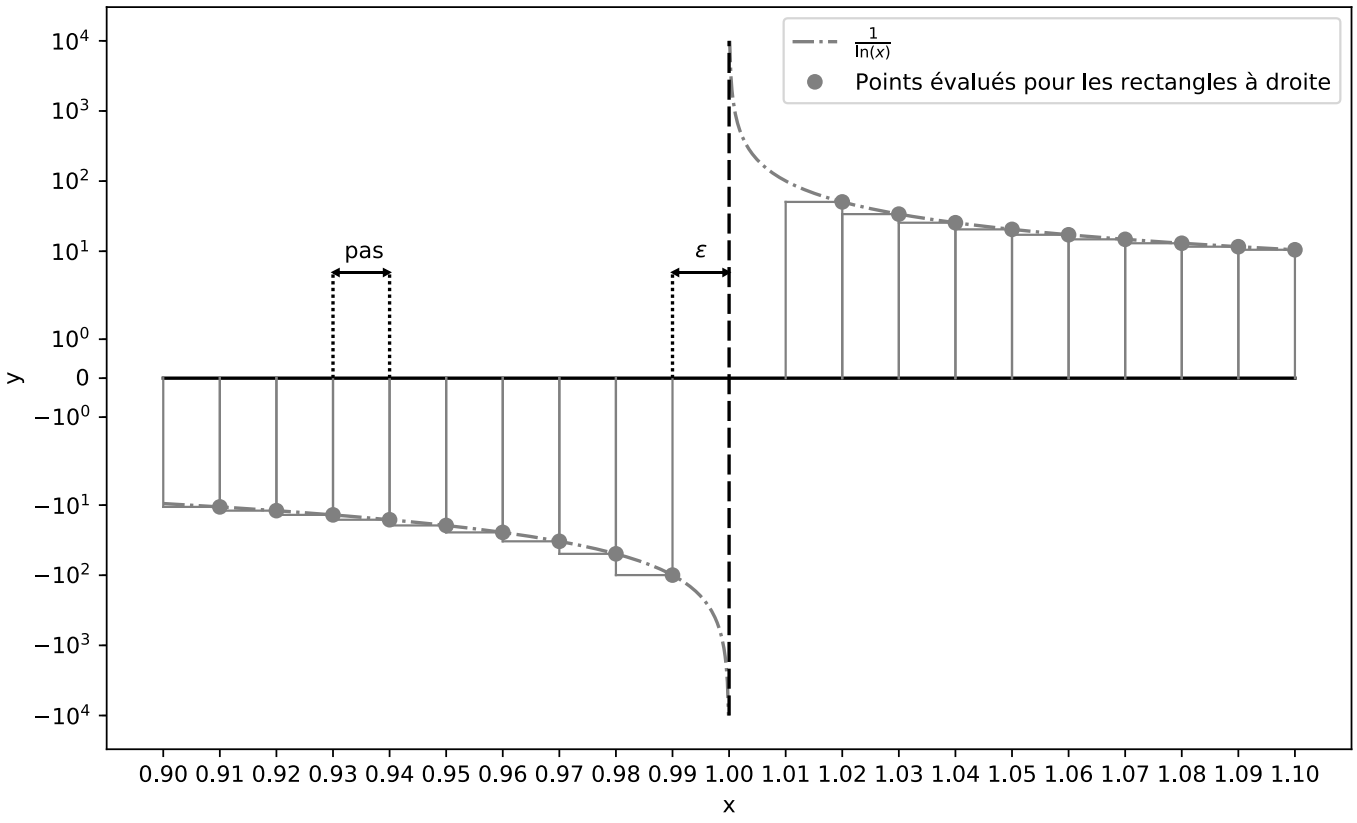


FIGURE 4 – Rectangles utilisés par la méthode des rectangles à droite au voisinage de 1 afin de calculer la valeur principale de Cauchy introduite dans l'équation (4). Les paramètres sont $pas = \varepsilon = 10^{-2}$

Estimation de li **via** Ei

L'approche par quadrature numérique n'est pas satisfaisante. Non seulement elle rend le temps d'exécution de `li_d` prohibitif quand x augmente mais de plus l'utilisateur doit choisir un pas sans règle claire à appliquer pour garantir une précision donnée. La fonction exponentielle intégrale Ei permet de pallier ce problème.

$$\begin{aligned} \mathrm{Ei} : \mathbb{R}^* &\to \mathbb{R} \\ x &\mapsto \int_{-\infty}^{x} \frac{e^t}{t} \mathrm{dt} \end{aligned} \tag{6}$$

Pour le cas $x > 0$ on utilise la valeur principale de Cauchy telle que vue pour li.
Le lien entre li et Ei est :

$$\mathrm{li}(x) = \mathrm{Ei}(\ln(x)) \tag{7}$$

Afin d'évaluer numériquement la valeur de Ei en un point on se base sur son développement (dit en série de Puiseux) sur $\mathbb{R}^{+*}$:

$$\mathrm{Ei}(x) = \gamma + \ln(x) + \sum_{k=1}^{\infty} \frac{x^k}{k \times k!} \tag{8}$$

Avec $\gamma \simeq 0.577215664901$ la constante d'Euler-Mascheroni.
Comme l'évaluation de la somme jusqu'à l'infini est impossible on utilise en pratique la somme suivante :

$$\mathrm{Ei}_n(x) = \gamma + \ln(x) + \sum_{k=1}^{n} \frac{x^k}{k \times k!} \tag{9}$$

Le choix de n se fait en comparant Ei_{n-1} à Ei_n jusqu'à ce qu'ils soient considérés comme suffisament proches.
L'évaluation via un ordinateur de ce développement est numériquement stable jusqu'à $x = 40$. Au delà les résultats sont entachés d'erreurs de calcul et d'autres méthodes doivent être utilisées.

❑ **Q24** – Écrire une fonction `li_dev(x)` qui calcule $\mathrm{li}(x)$ en se basant sur Ei_n et la fonction `sont_proches` de la question 2 (on pourra utiliser la fonction associée même si la question n'a pas été traitée). `li_dev` doit renvoyer `False` si :

- Ei_{n-1} et Ei_n ne peuvent pas être considérés comme proches au bout de `MAXIT` itérations.
- la valeur de `x` ne permet pas d'aboutir à un résultat.

Prendre `MAXIT = 100` se révèle largement suffisant à l'usage.
On demande à ce que la complexité dans le pire des cas soit $O($`MAXIT`$)$. Le paramètre `x` est un flottant quelconque.

Partie IV. Analyse de performance de code

Au cours du développement des fonctions nécessaires à la manipulation des nombres premiers on s'aperçoit que le choix des algorithmes pour évaluer chaque fonction est primordial pour garantir des performances acceptables. On souhaite donc mener des tests à grande échelle pour évaluer les performances réelles du code qui a été développé. Pour ce faire on effectue un grand nombre de tests sur une multitude d'ordinateurs. Les données sont ensuite centralisées dans une base de données composée de deux tables.
La première table est `ordinateurs` et permet de stocker des informations sur les ordinateurs utilisés pour les tests. Ses attributs sont :

- `nom TEXT`, clé primaire, le nom de l'ordinateur.
- `gflops INTEGER` la puissance de l'ordinateur en milliards d'opérations flottantes par seconde.
- `ram INTEGER` la quantité de mémoire vive de l'ordinateur en Go.

Exemple du contenu de cette table :

```
nom                 gflops      ram
----------          ----------  ----------
nyarlathotep114     69          32
nyarlathotep119     137         32
...
shubniggurath42     133         16
azathoth137         85          8
```

La seconde table est `fonctions` et stocke les informations sur les tests effectués pour différentes fonctions en cours de développement. Ses attributs sont :

- `id INTEGER` l'identifiant du test effectué.
- `nom TEXT` le nom de la fonction testée (par exemple li, Ei, etc).
- `algorithme TEXT` le nom de l'algorithme qui permet le calcul de la fonction testée (par exemple BBS si on teste une fonction de génération de nombres aléatoires).
- `teste_sur TEXT` le nom du PC sur lequel le test a été effectué.
- `temps_exec INTEGER` le temps d'exécution du test en millisecondes.

Exemple du contenu de cette table :

```
id          nom         algorithme  teste_sur        temps_exec
----------  ----------  ----------  ----------       ----------
1           li          rectangles  nyarlathotep165  2638
2           li          rectangles  shubniggurath28  736
3           li          trapezes    nyarlathotep165  4842
...
2154        Ei          puiseux     nyarlathotep145  2766
2155        aleatoire   BBS         azathoth145      524
```

❑ **Q25** – Expliquer pourquoi il n'est pas possible d'utiliser l'attribut `nom` comme clé primaire de la table `fonctions`.

❑ **Q26** – Écrire des requêtes `SQL` permettant de :

1. Connaître le nombre d'ordinateurs disponibles et leur quantité moyenne de mémoire vive.
2. Extraire les noms des PC sur lesquels l'algorithme `rectangles` n'a pas été testé pour la fonction nommée `li`.
3. Pour la fonction nommée `Ei`, trier les résultats des tests du plus lent au plus rapide. Pour chaque test retenir le nom de l'algorithme utilisé, le nom du pc sur lequel il a été effectué et la puissance du PC.

Fin de l'épreuve.

4. INFORMATIQUE

4.1. Informatique pour tous

Le sujet d'informatique commune portait cette année sur des techniques algorithmiques autour du thème des nombres premiers.

L'épreuve abordait un large spectre des notions vues durant les deux années de préparation des candidats.

Remarques générales

- La présentation générale de la très grande majorité des copies est satisfaisante.
- Le jury a été surpris de relever beaucoup d'erreurs de calcul parfaitement incongrues à ce niveau de formation. La simplification d'une somme géométrique comme $\sum_{i=0}^{N-1} 2^i$ pose des problèmes importants à plus d'un tiers des candidats. Bien qu'il ne s'agisse pas d'une épreuve de mathématique, on est en droit d'attendre des candidats une aisance calculatoire minimale.
- On a remarqué cette année dans beaucoup de copies un mélange entre des notations propres aux mathématiques (parties entières, racines...) et le code Python. Cela est généralement sanctionné : un algorithme en python n'est pas du pseudo-code, il faut utiliser les opérateurs du langage.
- Nous souhaitons insister sur le fait que la syntaxe **L=L+[a]** est beaucoup moins efficace pour ajouter un élément à la fin d'une liste que **L.append(a)**. Il serait souhaitable que les étudiants privilégient systématiquement cette deuxième solution au cours de leur formation.
- La gestion efficace des booléens n'est pas encore très assurée : on a par exemple croisé beaucoup plus de **if L[i]!=False** que de **if L[i]** … pourtant équivalents si **L[i]** est un booléen.
- Une remarque récurrente année après année : certains candidats semblent n'avoir aucune notion de la complexité algorithmique quand ils écrivent leurs programmes. Faire appel à une fonction de complexité significative comme **Pi** ou **erato_iter** au cœur d'une double boucle devrait pourtant leur poser un problème... surtout quand il existe une solution beaucoup plus simple et facilement accessible.
- Dans certaines copies, on peut parfois lire plusieurs solutions pour une question. Cette démarche pourrait être appréciée dans un autre contexte, mais dans le cadre d'une épreuve de concours en temps limité, le jury s'attend surtout à lire une solution qui fonctionne, si possible lisible et simple. Par ailleurs, le jury n'a pas à choisir entre une solution juste et une solution fausse : la notation se fera toujours sur la mauvaise.

Remarques particulières

Question 1. Généralement correcte, mais **numpy** et **matplotlib** n'ont rien à voir avec le module **math**...

Question 2. Beaucoup de confusions sur l'écriture des flottants… e-5, 10e-5, ou 10^-5 ne définissent pas 10^{-5} en python. Par ailleurs, l'affectation de 10^{-8} via rtol=0.00000001 n'est pas une solution élégante.

Question 3. Une application très simple de fonction récursive, généralement comprise.

Question 4. La moitié des candidats environ reconnaît la partie entière du logarithme en base b.

Question 5. Pour obtenir des points sur cette question qui abordait un point essentiel du programme d'informatique, il ne fallait pas se contenter de généralités creuses. il fallait reconnaître une manifestation des **erreurs d'arrondis** due à la **représentation des flottants sur un nombre limité de bits**. Beaucoup n'en ont pas été capables, mettant parfois en cause « l'ordinateur », « le langage python » ou prétendant que « l'addition est moins précise que la multiplication ».

Question 6. Trop de candidats sont incapables de répondre correctement à cette question, souvent à cause d'une mauvaise connaissance de la signification du préfixe giga.

Question 7. Une erreur fréquente : « un booléen doit être codé au minimum sur 2 bits, car il ne peut prendre que deux valeurs ». La notion de bit informatique est donc mal comprise de ces candidats.

Question 8 : Cette question de traduction d'un algorithme fourni en pseudo-code a été traitée de manière généralement décevante. Entre les initialisations fausses de la liste de booléens, les boucles qui ne commencent ou ne finissent pas avec la bonne valeur de l'incrément, des divisions euclidiennes sur les booléens (!) et, bien sûr, les erreurs d'indices lors de l'appel d'un élément de la liste (**erato_iter[i]** contenant l'information sur la primalité du nombre **i+1**...), on peut estimer à 10 % environ les candidats ayant proposé une solution parfaitement correcte de cet algorithme… On peut légitimement espérer mieux. Pour cela, une attention plus grande aux détails est nécessaire. Voir en annexe un exemple de bon code pour cette question.

Question 9. Pour traiter cette question correctement, il fallait bien entendu avoir proposé un parcours optimal à la question précédente.

Question 10. On a pu lire quelques très bonnes solutions pour cette question.

Question 11. Une grande majorité de candidats comprennent que $A=\sum_{i=0}^{N-1} 2^i$. Calculer correctement cette somme a par contre posé de nombreuses difficultés… cependant, on peut remarquer qu'en maîtrisant le principe du codage binaire, calculer la somme était inutile.

Question 12. On a relevé des confusions entre les opérateurs **%** et **//** . A ce propos, on a lu beaucoup trop de **if a/i==int(a/i)**, ce qui est une solution fausse pour vérifier que **a** est divisible par **i**.

Question 13. Très peu de candidats ont bien compris la valeur que devait prendre l'argument de **bbs**. Bravo à eux, ce point était subtil ! Par ailleurs, la condition de pseudo-primalité a

donné lieu à nombre de solutions fausses, dans lesquelles l'algorithme renvoyait un nombre dès que l'une des quatre conditions de divisibilité était satisfaite.

Question 14. Un seul appel à **erato_iter** judicieux évitait de reprogrammer des tests de divisibilité pour tous les nombres testés.

Question 15. La solution la plus simple consistait à faire appel une seule fois à **erato_iter**. De nombreux candidats se sont lancés dans des constructions trop complexes et coûteuses, qui ne respectaient pas la contrainte imposée par le sujet (complexité linéaire).

Question 16. On demandait de tester l'inéquation fournie à partir d'un nombre particulier, pas à partir de 0. L'appel à **Pi** dans la boucle a été sanctionné, à plus forte raison quand la liste de listes renvoyée par **Pi** était comparée à un flottant...

Question 17. Question en général bien réussie.

Question 18. Certains candidats croient que la méthode des trapèzes possède une complexité différente de celle des rectangles.

Question 19. Attention à l'utilisation de **range**. Trop de candidats ne semblent pas du tout gênés d'écrire **range(a,b,pas)** avec **a** et **b** et **pas** des flottants. De plus on a lu des solutions de la méthode des rectangles à gauche (certes celle exposée en cours le plus souvent, mais il faut savoir s'adapter).

Question 20. Les candidats ont souvent bien traité deux cas sur les trois selon la valeur de x, mais ont oublié le troisième.

Question 21. Question très peu traitée, et souvent incomprise.

Question 22. Question difficile, très rarement traitée.

Question 23. Question très rarement traitée.

Question 24. Attention à la solution tentante qui consistait à faire appel à une fonction de calcul de factorielle à chaque étape de la boucle... cela fait exploser la complexité.

Question 25. L'erreur la plus courante a été de dire que les deux tables possédant un attribut de même nom, celui-ci ne pouvait servir de clé primaire pour l'une des tables : cela n'a rien à voir avec la définition d'une clé primaire.

Question 26 : Peu de candidats ont perçu la nécessité d'une sous-requête (ou d'un **EXCEPT**) pour la deuxième requête. En revanche, la syntaxe d'une jointure semble mieux maîtrisée que les années précédentes.

Perles diverses

Comme chaque année, le jury a été ému de trouver dans les copies quelques perles inattendues, auxquelles nous avons décerné les prix suivants :

- Prix « perdu dans la ville » : « Le problème est que les erreurs d'arrondissement s'accumulent » ;
- Prix « Poésie et informatique » : « Le plus petit espace mémoire possible est le facteur spatial » ;
- Prix « le SQL Low-Cost » : « SELECT DISCOUNT FROM ordinateurs ».

ANNEXE : Proposition de code correct pour Q8

```
def erato_iter(N):
    liste_bool = N * [True]
    liste_bool[0] = False
    i = 2
    while i**2 <= N:
        if liste_bool[i - 1]:
            for k in range(2, N//i + 1):
                liste_bool[k*i - 1] = False
        i += 1
    return liste_bool
```

4.2. Informatique option — filière MP

Généralités.

Le sujet traite d'une méthode de réduction des automates.

Il fait appel, d'une part, à la notion formelle d'automate et, d'autre part, à des structures informatiques complexes que le candidat doit manipuler. L'ensemble permet de bien évaluer l'acquisition du programme des deux années de classe préparatoire.

Les candidats abordent l'ensemble des questions dans leur grande majorité.

La présentation des copies est globalement satisfaisante.

Nous avons pu constater peu d'efforts de rédaction des quelques questions théoriques.

Beaucoup de candidats ne donnent pas d'arguments, se contentent d'arguments superficiels ou ne citent pas les résultats précédemment montrés.

Première partie : Premiers exemples.

Questions 1-4. Il s'agit ici de décrire le langage accepté par un automate sous forme qualitative ou rationnelle. Ces notions sont globalement comprises.

Question 5. Les candidats montrent qu'ils ont compris la représentation informatique d'un automate choisie ici.

A2018 – INFO

ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT Atlantique, ENSAE PARISTECH.

Concours Centrale-Supélec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.

CONCOURS 2018

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée de l'épreuve : 1 heure 30 minutes

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI

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

INFORMATIQUE COMMUNE

L'énoncé de cette épreuve comporte 10 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.

Mesures de houle

Le sujet comporte des questions de programmation. Le langage à utiliser est `Python`.

On s'intéresse à des mesures de niveau de la surface libre de la mer effectuées par une bouée (représentée sur la Figure 1) [1]. Cette bouée contient un ensemble de capteurs incluant un accéléromètre vertical qui fournit, après un traitement approprié, des mesures à étudier [2].

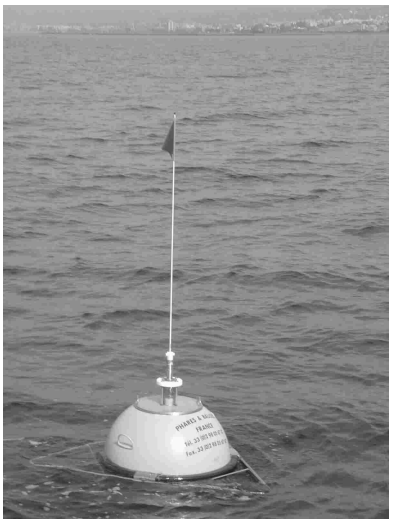


Figure 1 : bouée de mesure de houle

Les mesures réalisées à bord de la bouée sont envoyées par liaison radio à une station à terre où elles sont enregistrées, contrôlées et diffusées pour servir à des études scientifiques. Pour plus de sûreté, les mesures sont aussi enregistrées sur une carte mémoire interne à la bouée.

Partie I. Stockage interne des données

Une campagne de mesures a été effectuée. Les caractéristiques de cette campagne sont les suivantes :

- durée de la campagne : 15 jours ;
- durée d'enregistrement : 20 min toutes les demi-heures ;
- fréquence d'échantillonnage : 2 Hz.

Les relevés de la campagne de mesure sont écrits dans un fichier texte dont le contenu est défini comme suit.

Les informations relatives à la campagne sont rassemblées sur la première ligne du fichier, séparées par des points-virgules (" ;"). On y indique différentes informations importantes comme le numéro de la campagne, le nom du site, le type du capteur, la latitude et la longitude de la bouée, la date et l'heure de la séquence.

1. Cette étude utilise des résultats extraits de la base de données du Centre d'Archivage National des Données de Houle In Situ. Les acquisitions ont été effectuées par le Centre d'Etudes Techniques Maritimes Et Fluviales.

2. L'ensemble des paramètres des états de mer présent dans la base CANDHIS est calculé par les logiciels :
   - Houle5 (CETMEF) : analyse vague par vague (temporelle) ;
   - PADINES (EDF/LNHE) : analyse spectrale et directionnelle (fréquentielle).

Les lignes suivantes contiennent les mesures du déplacement vertical (m). Chaque ligne comporte 8 caractères dont le caractère de fin de ligne. Par exemple, on trouvera dans le fichier texte les 3 lignes suivantes :

```
+0.4256
+0.3174
-0.0825
...
```

❑ **Q1** – On suppose que chaque caractère est codé sur 8 bits. En ne tenant pas compte de la première ligne, déterminer le nombre d'octets correspondant à 20 minutes d'enregistrement à la fréquence d'échantillonnage de 2 Hz.

❑ **Q2** – En déduire le nombre approximatif (un ordre de grandeur suffira) d'octets contenus dans le fichier correspondant à la campagne de mesures définie précédemment. Une carte mémoire de 1 Go est-elle suffisante ?

❑ **Q3** – Si, dans un souci de réduction de la taille du fichier, on souhaitait ôter un chiffre significatif dans les mesures, quel gain relatif d'espace mémoire obtiendrait-on ?

❑ **Q4** – Les données se trouvent dans le répertoire de travail sous forme d'un fichier `donnees.txt`. Proposer une suite d'instructions permettant de créer à partir de ce fichier une liste de flottants `liste_niveaux` contenant les valeurs du niveau de la mer. On prendra garde à ne pas insérer dans la liste la première ligne du fichier.

Deux analyses sont effectuées sur les mesures : l'une est appelée "vague par vague", l'autre est appelée "spectrale".

Partie II. Analyse "vague par vague"

On considère ici que la mesure de houle est représentée par un signal $\eta(t) \in \mathbb{R}$, $t \in [0, T]$, avec η une fonction C^1.

On appelle niveau moyen m la moyenne de $\eta(t)$ sur $[0, T]$.

On définit Z_1, Z_2, ..., Z_n l'ensemble (supposé fini) des Passages par le Niveau moyen en Descente (PND) (voir Figure 2). A chaque PND, le signal traverse la valeur m en descente.

On suppose $\eta(0) > m$, et $\dfrac{\mathrm{d}\eta}{\mathrm{d}t}(0) > 0$.

On en déduit que $\eta(t) - m \geq 0$ sur $[0, Z_1]$.

Les hauteurs de vagues H_i sont définies par les différences :

$$\begin{cases} H_1 = \max\limits_{t\in[0,\ Z_1]} \eta(t) - \min\limits_{t\in[Z_1,\ Z_2]} \eta(t) \\ H_i = \max\limits_{t\in[Z_{i-1},\ Z_i]} \eta(t) - \min\limits_{t\in[Z_i,\ Z_{i+1}]} \eta(t) \ \text{ pour } 2 \leq i < n \end{cases}$$

.

On définit les *périodes de vagues* par $T_i = Z_{i+1} - Z_i$.

❑ **Q5** – Pour le signal représenté sur la Figure 2, que valent approximativement H_1, H_2 et H_3 ? Que valent approximativement T_1 et T_2 ?

On adopte désormais une représentation en temps discret du signal. On appelle *horodate*, un ensemble (fini) des mesures réalisées sur une période de 20 minutes à une fréquence d'échantillonnage de 2 Hz. Les informations de niveau de surface libre d'un horodate sont stockées dans une

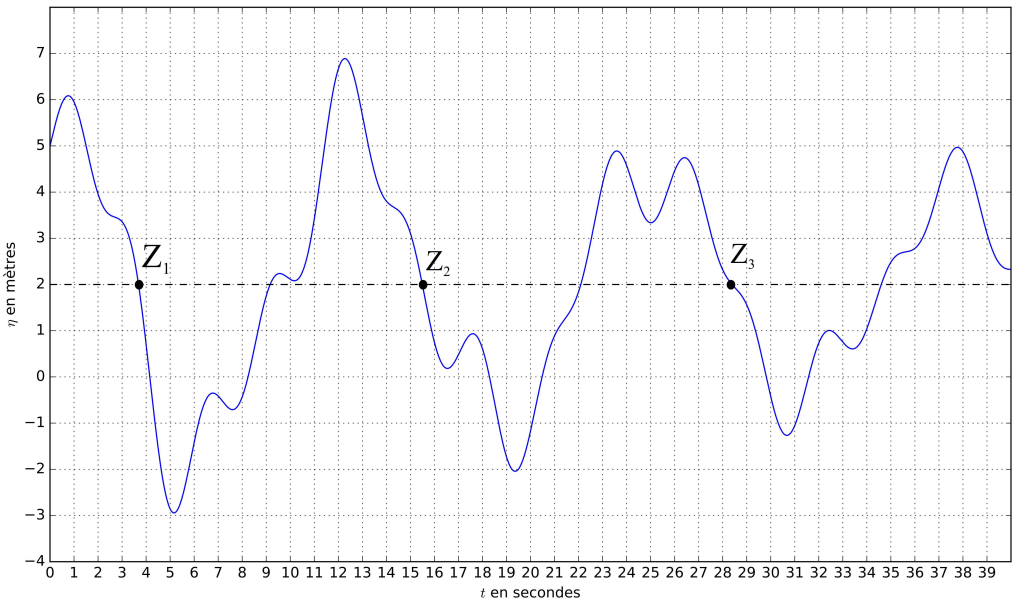


FIGURE 2 : Passages par le Niveau moyen en Descente (PND). Ici la moyenne m vaut 2

liste de flottants `liste_niveaux`. On suppose qu'aucun des éléments de cette liste n'est égal à la moyenne.

❑ **Q6** – Proposer une fonction `moyenne` prenant en argument une liste non vide `liste_niveaux`, et retournant sa valeur moyenne.

❑ **Q7** – Proposer une fonction `integrale_precise` prenant en argument une liste non vide `liste_niveaux`, et retournant la valeur approchée de l'intégrale de η sur une période de 20 minutes. On demande d'utiliser la méthode des trapèzes. En déduire une fonction `moyenne_precise` prenant en argument une liste non vide `liste_niveaux` et retournant une estimation de la moyenne de η sur une période de 20 minutes.

❑ **Q8** – Proposer une fonction `ind_premier_pzd(liste_niveaux)` retournant, s'il existe, l'indice du premier élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner -1 si aucun élément vérifiant cette condition n'existe.

❑ **Q9** – Proposer une fonction retournant l'indice i du *dernier* élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner -2 si aucun élément vérifiant cette condition n'existe. On cherchera à proposer une fonction de complexité $O(1)$ dans le meilleur des cas.

On souhaite stocker dans une liste `successeurs`, les indices des points succédant (strictement) aux PND (voir Figure 3).

❑ **Q10** – On propose la fonction `construction_successeurs` en annexe (algorithme 1). Elle retourne la liste `successeurs`. Compléter (sur la copie) les lignes 6 et 7.

❑ **Q11** – Proposer une fonction `decompose_vagues(liste_niveaux)` qui permet de décomposer une liste de niveaux en liste de vagues. On omettra les données précédant le premier PND et celles

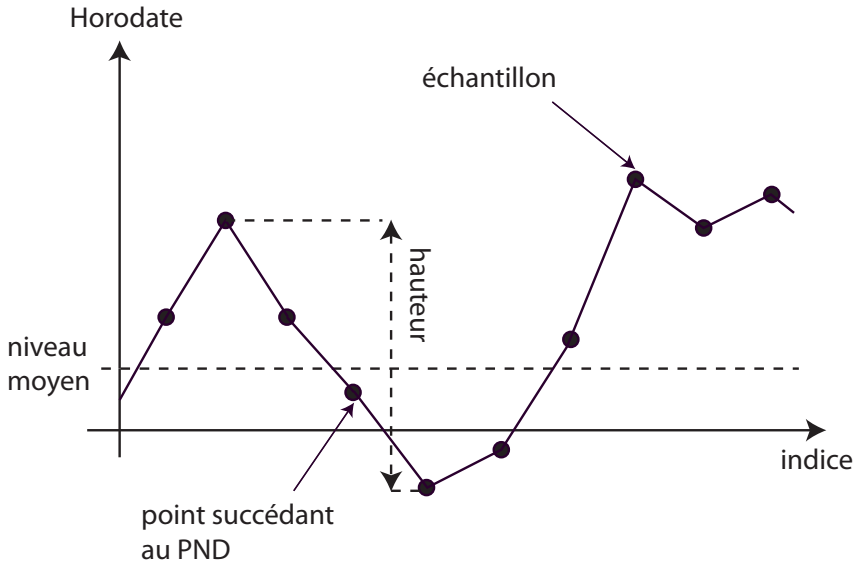


FIGURE 3 : propriétés d'une vague

succédant au dernier PND. Ainsi `decompose_vagues([1, -1, -2, 2, -2, -1, 6, 4, -2, -5])` (noter que cette liste est de moyenne nulle) retournera `[[-1, -2, 2], [-2, -1, 6, 4]]`.

On désire maintenant caractériser les vagues.

Ainsi, on cherche à concevoir une fonction `proprietes(liste_niveaux)` retournant une liste de listes à deux éléments `[Hi,Ti]` permettant de caractériser *chacune des vagues i* par ses attributs :

- `Hi`, sa hauteur en mètres (m) (voir Figure 3),
- `Ti`, sa période en secondes (s).

❑ **Q12** – Proposer une fonction `proprietes(liste_niveaux)` réalisant cet objectif. On pourra utiliser les fonctions de Python `max(L)` et `min(L)` qui retournent le maximum et le minimum d'une liste `L`, respectivement.

Partie III. Contrôle des données

Plusieurs indicateurs sont couramment considérés pour définir l'état de la mer. Parmi eux, on note :

- H_{max} : la hauteur de la plus grande vague observée sur l'intervalle d'enregistrement $[0, T]$;
- $H_{1/3}$: la valeur moyenne des hauteurs du tiers supérieur des plus grandes vagues observées sur $[0, T]$;
- $T_{H1/3}$: la valeur moyenne des périodes du tiers supérieur des plus grandes vagues observées sur $[0, T]$.

❑ **Q13** – Proposer une fonction prenant en argument la liste `liste_niveaux` de la question 12 et retournant H_{max}.

Afin de déterminer $H_{1/3}$ et $T_{H1/3}$, il est nécessaire de trier la liste des propriétés des vagues. La méthode utilisée ici est un *tri rapide* (*quick sort*). On donne en annexe un algorithme possible pour la fonction `triRapide` (algorithme 2). Trois arguments sont nécessaires : une liste `liste`, et deux indices `g` et `d`.

❑ **Q14** – Préciser les valeurs que doivent prendre les arguments `g` et `d` au premier appel de la fonction `triRapide`. Compléter (sur la copie) la ligne 2.

Lorsque le tri rapide est utilisé et que le nombre de données à traiter devient petit dans les sous-listes (de l'ordre de 15), il peut être avantageux d'utiliser un "tri par insertion". On appelle `triInsertion` la fonction qui permet d'effectuer un tri par insertion. Elle admet en argument une liste `liste`, et deux indices `g` et `d`. Ces deux indices permettent de caractériser la sous-partie de la liste à trier (indices de début et de fin inclus).

❑ **Q15** – Donner les modifications à apporter à la fonction `triRapide` pour que, lorsque le nombre de données dans une sous-liste devient inférieur ou égal à 15, la fonction `triInsertion` soit appelée pour terminer le tri.

Le code incomplet de la fonction `triInsertion` est donné en annexe : algorithme 3.

❑ **Q16** – La fonction `triInsertion` admet trois arguments : une liste de données `liste`, et deux indices `g` et `d`. Elle trie dans l'ordre croissant la partie de la liste comprise entre les indices `g` et `d` inclus. Compléter cette fonction (avec sur la copie le nombre de lignes de votre choix).

La distribution des hauteurs de vague (voir Figure 4) lors de l'analyse *vague par vague* est réputée être gaussienne. On peut contrôler ceci par des tests de *skewness* (variable désignée par S) et de *kurtosis* (variable désignée par K) définis ci-après. Ces deux tests permettent de quantifier respectivement l'asymétrie et l'aplatissement de la distribution.

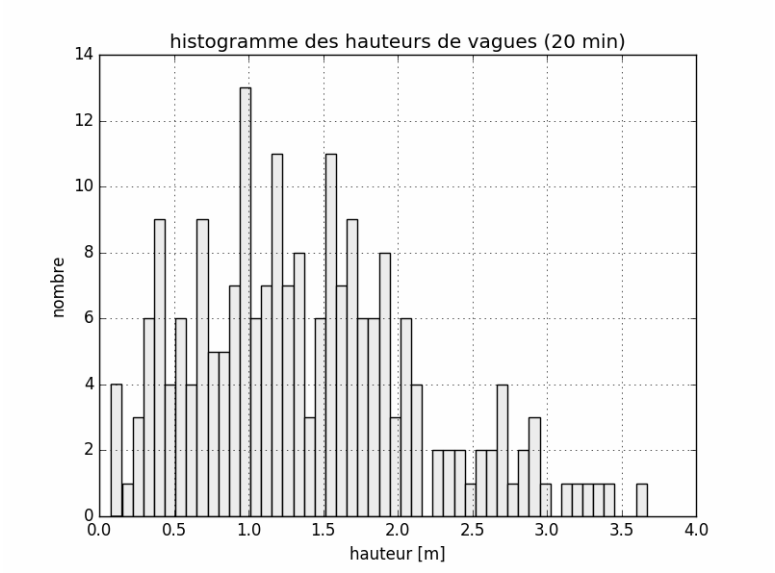


FIGURE 4 : histogramme des hauteurs de vague

On appelle $\overline{H}$ et σ^2 les estimateurs non biaisés de l'espérance et de la variance, n le nombre d'éléments H_1, H_2, ..., H_n.

On définit alors

$$S = \frac{n}{(n-1)(n-2)} \times \left(\frac{1}{\sigma^3}\right) \times \sum_{i=1}^{n} \left(H_i - \overline{H}\right)^3$$

$$K = \frac{n}{(n-1)(n-2)(n-3)} \times \left(\frac{1}{\sigma^4}\right) \times \sum_{i=1}^{n} \left(H_i - \overline{H}\right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

Le test suivant est appliqué :

– si la valeur absolue de S est supérieure à 0,3 alors l'horodate est déclaré non valide ;
– si la valeur de K est supérieure à 5 alors l'horodate est déclaré non valide.

On utilise la fonction `moyenne` pour estimer la valeur de $\overline{H}$, et on suppose disposer de la fonction `ecartType` qui permet de retourner la valeur de l'écart type non biaisé σ.

❑ **Q17** – Un codage de la fonction `skewness` pour une liste ayant au moins 3 éléments est donné en annexe (algorithme 4). Le temps d'exécution est anormalement long. Proposer une modification simple de la fonction pour diminuer le temps d'exécution (sans remettre en cause le codage des fonctions `ecartType` et `moyenne`).

❑ **Q18** – Doit-on s'attendre à une différence de type de la complexité entre une fonction évaluant S et une fonction évaluant K ?

Partie IV. Base de données relationnelle

On dispose d'une base de données relationnelle `Vagues`.

La première table est `Bouee`. On se limite aux attributs suivants : le numéro d'identification `idBouee`, le nom du site `nomSite`, le nom de la mer ou de l'océan `localisation`, le type du capteur `typeCapteur` et la fréquence d'échantillonnage `frequence`.

`Bouee`

idBouee	nomSite	localisation	typeCapteur	frequence
831	Porquerolles	Mediterranee	Datawell non directionnelle	2.00
291	Les pierres noires	Mer d'iroise	Datawell directionnelle	1.28
...	...	...	...	...

La seconde table est `Campagne`.

On se limite aux attributs suivants : le numéro d'identification `idCampagne`, le numéro d'identification de la bouée `idBouee`, la date de début `debutCampagne` et la date de fin `finCampagne`.

`Campagne`

idCampagne	idBouee	debutCampagne	finCampagne
08301	831	01/01/2010 00h00	15/01/2010 00h00
02911	291	15/10/2005 18h30	18/10/2005 08h00
...	...	...	...

La troisième table est `Tempete`. Les informations fournies relatives à un événement "tempête" sont les suivantes :

– date de début et fin de tempête,
– évolution des paramètres $H_{1/3}$ et H_{max} en fonction du temps,
– le détail de certains paramètres non définis ici, obtenus au pic de tempête.

On se limite aux attributs suivants : le numéro d'identification de la tempête `idTempete`, le numéro d'identification de la bouée `idBouee`, la date de début `debutTempete`, la date de fin `finTempete`, la valeur maximale de hauteur de vague `Hmax`.

`Tempete`

idTempete	idBouee	debutTempete	finTempete	Hmax
083010	831	07/01/2010 20h00	09/01/2010 15h30	5.3
029012	291	16/10/2005 08h30	18/10/2005 09h00	8.5
...	...	...	...	...

Le schéma de la base de données est donc : `Vagues`={`Bouee`, `Campagne`, `Tempete`}.

❑ **Q19** – Formuler les requêtes SQL permettant de répondre aux questions suivantes :
– "Quels sont le numéro d'identification et le nom de site des bouées localisées en Méditerranée ?"

– “Quel est le numéro d’identification des bouées où il n’y a pas eu de tempêtes ?”
– “Pour chaque site, quelle est la hauteur maximale enregistrée lors d’une tempête ?”

Partie V. Analyse “spectrale”

L’analyse spectrale (fréquentielle) du niveau, permet elle aussi de caractériser l’état de la mer, qui peut, en première approximation, être modélisé par une superposition linéaire d’ondes sinusoïdales indépendantes.

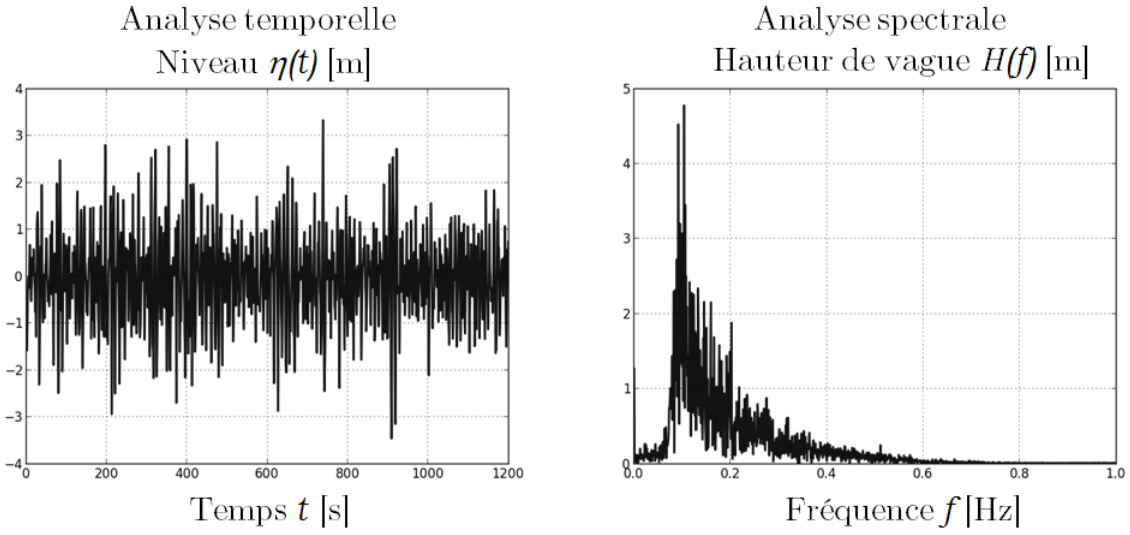


FIGURE 5 : analyses temporelle et spectrale

Des coefficients estimateurs de l’état de la mer issus de l’analyse spectrale ont donc été définis. Parmi eux, on note par exemple H_{m0} la hauteur significative spectrale des vagues ou T_P la période de pic barycentrique.

Pour leur calcul, il est nécessaire d’introduire la Transformation de Fourier Discrète (TFD).

Sa définition pour un signal numérique x de N échantillons, est la suivante :

$$X_k = \sum_{i=0}^{N-1} x_i \times e^{-2\pi j k \frac{i}{N}} \ , \ 0 \leq k < N \text{ et } j^2 = -1 \qquad (1)$$

Il existe plusieurs méthodes dites de “transformée de Fourier rapide”. On étudie dans la suite l’algorithme de Cooley – Tukey adapté de celui de Gauss. On propose ici une réécriture de (1) appelée entrelacement temporel (DIT decimation-in-time).

Dans toute la suite, on suppose que N **est une puissance de 2**.

On note $w = e^{-\frac{2\pi j}{N}}$ (qui est une racine N-ième de l’unité).

On pose

$$P_k = \sum_{i=0}^{N/2-1} x_{2i} \times e^{-\frac{2\pi j}{N/2} ik}$$

$$I_k = \sum_{i=0}^{N/2-1} x_{2i+1} \times e^{-\frac{2\pi j}{N/2} ik}$$

P_k : TFD des indices pairs, I_k : TFD des indices impairs

On montre alors que pour $0 \leq k < \frac{N}{2}$, $\left| \begin{array}{l} X_k = P_k + w^k I_k \\ X_{k+N/2} = P_k - w^k I_k \end{array} \right.$

L'algorithme est de type "diviser pour régner" : le calcul d'une TFD pour N éléments se fait à l'aide de deux TFD de $N/2$ éléments.

❑ **Q20** – Quelle est la complexité en temps de cet algorithme en fonction de N ? Justifier en une ou deux lignes.

❑ **Q21** – Écrire une fonction *récursive* prenant en argument la liste de données `x` et retournant la liste `X` obtenue par transformée de Fourier discrète rapide. La longueur de `x` est une puissance de 2.

Annexe

Algorithme 1

```
1  def construction_successeurs(liste_niveaux):
2      n=len(liste_niveaux)
3      successeurs=[]
4      m=moyenne(liste_niveaux)
5      for i in range(n-1):
6          if                    # A completer
7                                # A completer
8      return successeurs
```

Algorithme 2

```
1  def triRapide(liste,g,d):
2      pivot=                          # A completer
3      i=g
4      j=d
5      while True:
6          while i<=d and liste[i][0]<pivot:
7              i=i+1
8          while j>=g and liste[j][0]>pivot:
9              j=j-1
10         if i>j:
11             break
12         if i<j:
13             liste[i],liste[j]=liste[j],liste[i]
14         i=i+1
15         j=j-1
16     if g<j:
17         triRapide(liste,g,j)
18     if i<d:
19         triRapide(liste,i,d)
```

Algorithme 3

```
1  def triInsertion(liste,g,d):
2      for i in range(g+1,d+1):
3          j=i-1
4          tmp = liste[i]
5          while                          # A completer
6
7
8
9
10
11
12
13         liste[j+1]=tmp
```

Algorithme 4

```
1 def skewness(liste_hauteurs):
2     n=len(liste_hauteurs)
3     et3=(ecartType(liste_hauteurs))**3
4     S=0
5     for i in range(n):
6         S+=(liste_hauteurs[i]-moyenne(liste_hauteurs))**3
7     S=n/(n-1)/(n-2)*S/et3
8     return S
```

Fin de l'épreuve.

4. INFORMATIQUE

4.1. Informatique pour tous

Remarques générales

Le jury souhaite attirer l'attention des candidats et de leurs formateurs sur quelques points généraux relevés cette année.

- Les copies sont en général bien présentées, et sauf rares exceptions, l'indentation est assez claire. Rappelons cependant que celle-ci est capitale pour juger de la validité d'un programme, et qu'un doute sur celle-ci est défavorable au candidat. Des espaces clairs utilisant les carreaux ou quelques barres verticales bien placées semblent un bon compromis, et il ne faut pas hésiter à commencer bien à gauche de la copie pour ne pas se retrouver coincé en fin de ligne.
- Certaines copies utilisent des notions ou des fonctions spécifiques à PYTHON, mais imparfaitement maîtrisées. Il est préférable de bien maîtriser ce que demande le programme, plutôt que de tenter une syntaxe hasardeuse utilisant des notions ou fonctions pas clairement maîtrisées, qui rendent le programme invalide.
- L'utilisation de range n'est pas suffisamment maîtrisée. On a l'impression que beaucoup de candidats ne prennent pas le temps de se demander entre quelles bornes exactes les éléments de l'itérateur doivent prendre leurs valeurs, et un parcours d'éléments décroissants donne lieu à des syntaxes souvent fausses. Rappelons que range(a,b,-1) avec a>b permet de parcourir l'ensemble des valeurs de a inclus à b **exclu** par pas de -1. Dans le même ordre d'idée, beaucoup trop de candidats écrivent for i in range(len(L)) : avant de faire appel à L[i+1] dans la boucle sans paraître conscients du problème.
- La question 4 faisait appel explicitement à une notion du programme : l'exploitation de données dans un fichier texte. Les candidats traitant cette question sont rares, et ceux qui la réussissent entièrement sont une infime minorité. Le jury peut s'attendre à ce que cette technique pourtant fondamentale dans les applications pratiques soit traitée de manière plus satisfaisante. La syntaxe nécessaire n'est pas lourde, et elle doit être mieux maîtrisée.
- Il est dommage que dans de nombreuses copies, on trouve des appels à des fonctions complexes au sein d'une boucle, alors qu'un seul appel avant la boucle suffit. Le jury redemande explicitement aux candidats de s'entraîner à éviter ces appels multiples inutiles qui augmentent souvent artificiellement la complexité d'un programme.

Commentaires spécifiques à chaque question

Q1 : Peu de bonnes réponses parfois totalement fausses pour un calcul élémentaire.
Q2 : Il est très surprenant de constater que beaucoup de candidats ne maîtrisent pas le sens du préfixe giga, le confondant avec méga. Cela n'est pas acceptable à ce niveau de formation.
Q3 : La notion de gain **relatif** est trop mal maîtrisée. Le nombre de candidats incapables de conclure simplement que la suppression d'un caractère sur 8 permet d'obtenir un gain relatif d'espace de 1/8 est malheureusement élevé.
Q4 : Question très souvent non traitée (voir les commentaires généraux).
Q5 : Question bien traitée en général.
Q6 : Question bien traitée, sauf par les candidats divisant la somme par len(L)-1, et nous avons été surpris de rencontrer parfois l'opérateur // pour faire la division par len(L), qui n'était pas du tout adapté à cette question. On a par ailleurs constaté un certain nombre de fois l'utilisation d'un compteur pour compter le nombre d'éléments de la liste, alors que la boucle liste était parcourue avec range(len(L))…

Q7 : Question pas toujours bien traitée, les erreurs les plus courantes étant l'oubli du pas, et surtout l'appel à liste_niveaux[i+1] dans la boucle alors que la boucle était construite avec range(len(liste_niveaux)).
Quelques distinctions de cas (if L[i]>L[i+1]… elif L[i+1]>L[i]…) inutiles nous ont surpris, ainsi que de très nombreux cas de décomposition des trapèzes en un rectangle et un triangle.
Q8 : Cette question était un bon test pour la maîtrise de la syntaxe élémentaire du langage. Les erreurs les plus courantes étaient un range allant un entier trop loin et un return -1 mal indenté qui terminait l'exécution de la fonction dès le premier test.
Q9 : Il y avait une petite erreur dans le sujet, puisque le calcul de la valeur moyenne de la liste était nécessairement de complexité linéaire. Il fallait entendre que la seule recherche du dernier PND devait être de complexité unitaire dans le meilleur des cas, ce que les candidats ont compris sans difficulté en général, mais souvent raté à cause d'une maîtrise insuffisante des bornes de recherches (par un range mal utilisé ou un while avec une condition inexacte).
Certains ont tenté d'inverser la liste pour utiliser le résultat de la question 8. Il fallait prendre garde à ne pas renvoyer un passage par la valeur moyenne en montée, ce qui n'était pas le but recherché.
Q10 : Question simple, pas toujours bien traitée.
Q11 : Question de mise en forme plus difficile. La solution la plus simple et élégante utilisait bien entendu la fonction construction_successeurs et le « slicing » de liste_niveaux, ce que les meilleurs candidats ont compris sans difficulté. À l'inverse, nous avons pu lire des solutions mal pensées, voire très alambiquées. On pourrait conseiller à certains candidats un petit temps de réflexion avant de partir tête baissée dans l'écriture d'un algorithme non trivial.
Q12 : Question assez facile, mais il fallait penser que la période de la vague devait être donnée en secondes. Attention également à la structure du résultat renvoyé : une liste de listes à deux éléments n'est pas une liste de deux listes.
Q13 : Cette question classique de recherche de maximum a été souvent bien traitée.
Q14 : Le tri était fait explicitement sur un élément de la sous-liste, il fallait initialiser le pivot en conséquence.
Q15 : Question peu traitée, la distinction des cas qu'elle nécessitait n'a pas toujours été bien comprise.
Q16 : Quelques confusions sur le tri par insertion : le rôle de la variable tmp n'a pas toujours été perçue clairement par les candidats. Il s'agit d'un algorithme au programme, une meilleure maîtrise de celui-ci est souhaitable.
Q17 : Question en général bien réussie. Il est plus surprenant que des candidats qui répondent correctement à cette question soient tombés, au cours de leurs programmes précédents, dans le travers qu'elle souligne.
Q18 : Question assez bien traitée dans l'ensemble, mais les réponses ont parfois manqué de précision dans la justification, et on a parfois dû subir des réponses délirantes, certains candidats allant jusqu'à évoquer des complexités en O(1/n)…
Q19 : La première requête SQL ne pose pas de problème, sauf à ceux qui ne se sont manifestement pas entraînés sur ce langage. Les autres requêtes, plus complexes, ont eu des succès variables. Parmi les erreurs rencontrées, l'appel à un attribut via **attribut.table** à la place de **table.attribut** . De plus, la condition de jointure après ON est bien une condition, et non seulement le nom d'un attribut.
Rappelons que la seule syntaxe exigible du programme pour effectuer une jointure est table1 JOIN table2 ON condition (jointure symétrique simple). Trop de candidats ont voulu se lancer dans des NATURAL JOIN et autres FULL JOIN sans maîtriser précisément leurs effets.
Q20 : Peu de bonnes réponses sur cette question. Mis à part les habituelles réponses de complexité délirantes (O(n!), $O(2^n)$… les candidats ont-ils conscience de ce qu'une telle complexité signifie concrètement?) on a aussi vu beaucoup de complexités en O(ln(n)). Rappelons qu'une complexité logarithmique implique que l'on n'ait pas besoin d'accéder à l'intégralité des données.

Q21 : Question difficile très rarement traitée.

Perles diverses

Comme chaque année, le jury a été ému de trouver dans les copies quelques perles inattendues, auxquelles nous avons décerné cette année les prix suivants :

- Prix « L'informatique pour les nuls » :
  « Q4 : On utilise Word pour ouvrir le fichier. Il y a une tab "insérer tableau". On copie les données, on les colle et on nomme la tab : liste niveaux. On s'assure qu'on choisit une place après la première ligne pour le tableau. »

- Prix « Piège à correcteur »
  « Q14 : Voir la copie. » (écrit sur la copie)

- Prix « SQL, WHAT ? »
  « Q19 : WHAT Hmax FROM tempete »

Nous souhaitons une bonne préparation à cette épreuve aux futurs candidats !

4.2. Informatique — filière MP

Remarques générales

Le sujet traite de la recherche des positions d'un motif dans un texte. Il fait appel, d'une part, à la notion formelle d'automate et, d'autre part, à des structures informatiques complexes que le candidat doit manipuler. L'ensemble permet de bien évaluer l'acquisition du programme des deux années de classe préparatoire.
Les candidats abordent l'ensemble des questions dans leur grande majorité. Ils finissent parfois le sujet (en passant les questions difficiles). Quelques (rares) excellentes copies ont pu être lues.
La présentation des copies est globalement satisfaisante.
Nous avons pu constater peu d'efforts de rédaction des quelques questions théoriques.
Beaucoup de candidats ne donnent pas d'arguments, ou se contentent d'arguments superficiels.
Pour ce qui est de la manipulation des objets de type élaboré, une certaine aisance a pu être globalement appréciée.
Certains codes sont parfois bien trop compliqués ou difficiles à comprendre.
Il est rappelé que les codes doivent être clairs. Utiliser l'indentation est un excellent moyen d'y parvenir.

Première partie : recherche naïve dans un texte.

Questions 1-2

Le but de ces deux questions est de faire découvrir la possibilité du recouvrement des positions du motif dans le texte. Beaucoup de candidats passent à côté de cette finesse et se trouvent ensuite pénalisés dans l'écriture des codes.

Questions 3 - 4

A2017 – INFO

ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT Atlantique (ex Télécom Bretagne),
ENSAE PARISTECH.

Concours Centrale-Supelec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.

CONCOURS 2017

ÉPREUVE D'INFORMATIQUE COMMUNE

Durée de l'épreuve : 1 heure 30 minutes

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI

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

INFORMATIQUE COMMUNE

L'énoncé de cette épreuve comporte 7 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.

Étude de trafic routier

Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figure 1(a) et 1(b)). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

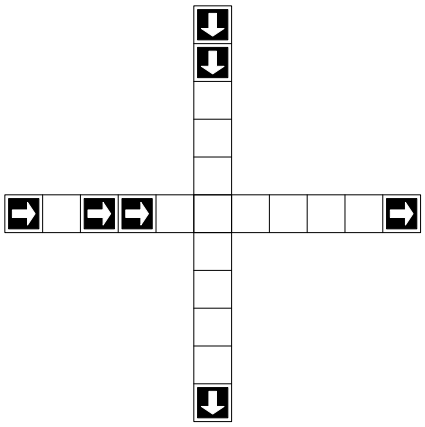
Le sujet comporte des questions de programmation. Le langage à utiliser est `Python`.

Notations

Soit L une liste,

- on note $len(L)$ sa longueur ;
- pour i entier, $0 \leq i < len(L)$, l'élément de la liste d'indice i est noté $L[i]$;
- pour i et j entiers, $0 \leq i < j \leq len(L)$, $L[i : j]$ est la sous-liste composée des éléments $L[i]$, …, $L[j-1]$;
- $p * L$, avec p entier, est la liste obtenue en concaténant p copies de L. Par exemple, $3 * [0]$ est la liste [0, 0, 0].

(a) Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10.



(b) Configuration représentant deux files de circulation à sens unique se croisant en une case. Les voitures sont représentées par un carré noir.

Figure 1 – Files de circulation

Partie I. Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1(a). 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 1(a)) et sont indifférenciées.

❑ **Q1** – Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

❑ **Q2** – Donner une ou plusieurs instructions `Python` permettant de définir une liste A représentant la file de voitures illustrée par la Figure 1(a).

❑ **Q3** – 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.

❑ **Q4** – Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse.

❑ **Q5** – Écrire une fonction `egal(L1, L2)` retournant un booléen permettant de savoir si deux listes $L1$ et $L2$ sont égales.

❑ **Q6** – Que peut-on dire de la complexité de cette fonction ?

❑ **Q7** – Préciser le type de retour de cette fonction.

Partie II. Déplacement de voitures dans la file

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 2 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 2(a) permet d'obtenir soit la liste illustrée par la Figure 2(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 2(c) lorsque l'on considère qu'une voiture nouvelle est introduite.

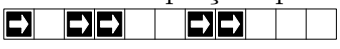
(a) Liste initiale A

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

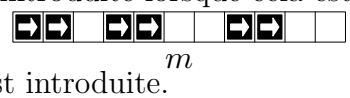
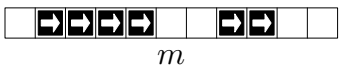
(c) $C =$ `avancer`$(A,$ `True`$)$

FIGURE 2 – Étape de simulation

❑ **Q8** – Étant donnée A la liste définie à la question 2, que renvoie `avancer(avancer(`A`, False), True)` ?

❑ **Q9** – On considère L une liste et m l'indice d'une case de cette liste ($0 \leq m < len(L)$). On s'intéresse à une *étape partielle* où seules les voitures situées sur la case d'indice m ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.
Par exemple, la file  devient 
m m
Définir en `Python` la fonction `avancer_fin`$(L,\ m)$ qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L.

❑ **Q10** – Soient L une liste, b un booléen et m l'indice d'une case *inoccupée* de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice m se déplacent, les autres voitures ne se déplacent pas. Le booléen b indique si une nouvelle voiture est introduite sur la case la plus à gauche.
Par exemple, la file  devient  lorsque aucune
m m
nouvelle voiture n'est introduite.
Définir en `Python` la fonction `avancer_debut`$(L,\ b,\ m)$ qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L.

❑ **Q11** – On considère une liste L dont la case d'indice $m > 0$ est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice m ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen b indique si une nouvelle voiture est introduite lorsque cela est possible.
Par exemple, la file  devient  lorsque aucune nouvelle voiture n'est introduite.

Définir en `Python` la fonction `avancer_debut_bloque`(L, b, m) qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

On considère dorénavant deux files $L1$ et $L2$ de même longueur impaire se croisant en leur milieu ; on note m l'indice de la case du milieu. La file $L1$ est toujours prioritaire sur la file $L2$. Les voitures ne peuvent pas quitter leur file et la case de croisement ne peut être occupée que par une seule voiture. Les voitures de la file $L2$ ne peuvent accéder au croisement que si une voiture de la file $L1$ ne s'apprête pas à y accéder. Une *étape de simulation à deux files* se déroule en deux temps. Dans un premier temps, on déplace toutes les voitures situées sur le croisement ou après. Dans un second temps, les voitures situées avant le croisement sont déplacées en respectant la priorité. Par exemple, partant d'une configuration donnée par la Figure 3(a), les configurations successives sont données par les Figures 3(b), 3(c), 3(d), 3(e) et 3(f) en considérant qu'*aucune nouvelle voiture n'est introduite*.

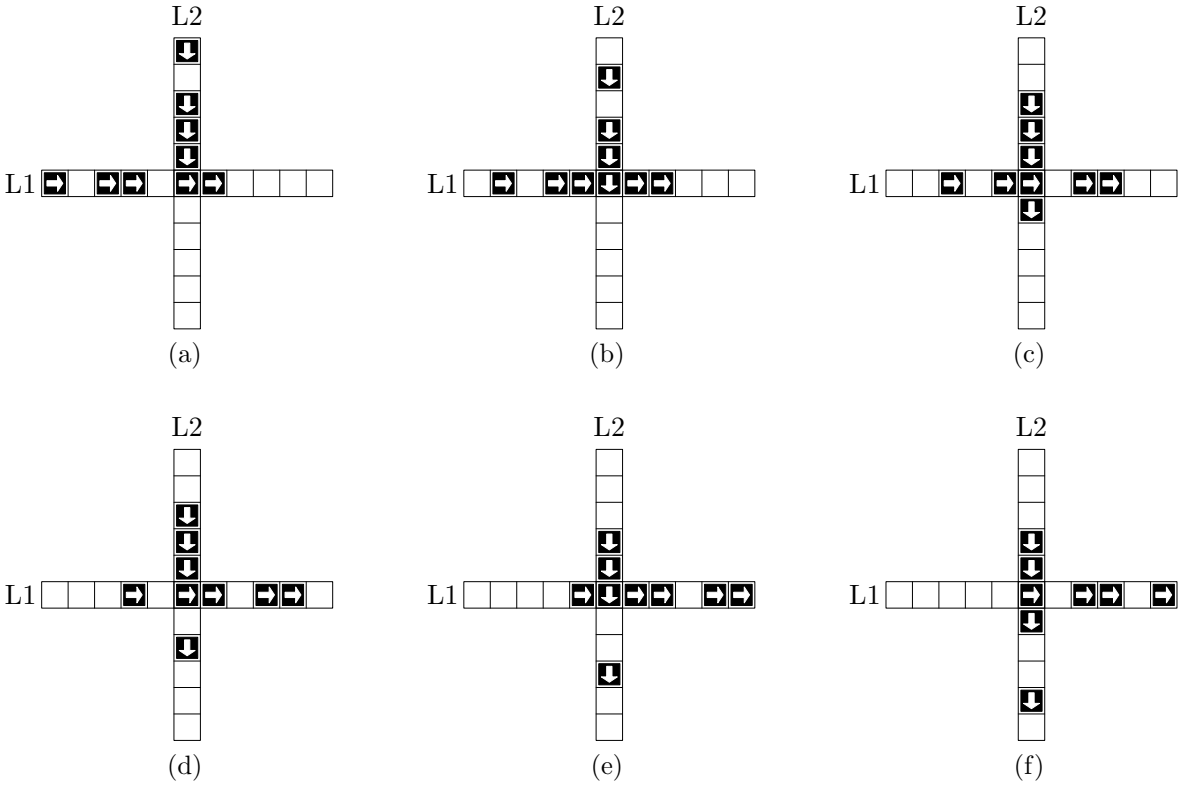


FIGURE 3 – Étapes de simulation à deux files

Partie III. Une étape de simulation à deux files

L'objectif de cette partie est de définir en `Python` l'algorithme permettant d'effectuer une étape de simulation pour ce système à deux files.

❑ **Q12** – En utilisant le langage `Python`, définir la fonction `avancer_files`($L1$, $b1$, $L2$, $b2$) qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée [$R1$, $R2$] sans changer les listes $L1$ et $L2$. Les booléens $b1$ et $b2$ indiquent respectivement si une nouvelle voiture est introduite dans les files $L1$ et $L2$. Les listes $R1$ et $R2$ correspondent aux listes après déplacement.

❑ **Q13** – On considère les listes

$$D = [\texttt{ False}, \texttt{True}, \texttt{False}, \texttt{True}, \texttt{False}], \qquad E = [\texttt{False}, \texttt{True}, \texttt{True}, \texttt{False}, \texttt{False}]$$

Que renvoie l'appel avancer_files(D, `False`, E, `False`) ?

Partie IV. Transitions

❑ **Q14** – En considérant que de nouvelles voitures peuvent être introduites sur les premières cases des files lors d'une étape de simulation, décrire une situation où une voiture de la file $L2$ serait indéfiniment bloquée.

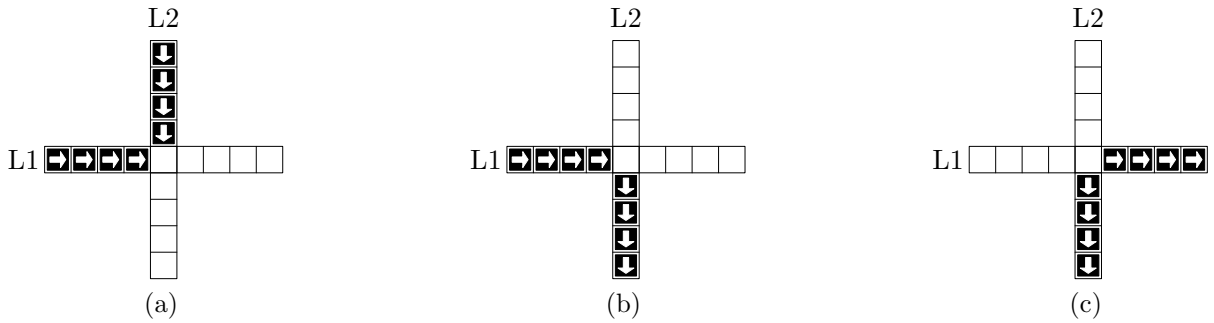


FIGURE 4 – Étude de configurations

❑ **Q15** – Étant données les configurations illustrées par la Figure 4, combien d'étapes sont nécessaires (on demande le nombre minimum) pour passer de la configuration 4(a) à la configuration 4(b) ? Justifier votre réponse.

❑ **Q16** – Peut-on passer de la configuration 4(a) à la configuration 4(c) ? Justifier votre réponse.

Partie V. Atteignabilité

Certaines configurations peuvent être néfastes pour la fluidité du trafic. Une fois ces configurations identifiées, il est intéressant de savoir si elles peuvent apparaître. Lorsque c'est le cas, on dit qu'une telle configuration est *atteignable*.

Pour savoir si une configuration est atteignable à partir d'une configuration initiale, on a écrit le code *incomplet* donné en annexe.

Le langage `Python` sait comparer deux listes de booléens à l'aide de l'opérateur usuel $<$, on peut ainsi utiliser la méthode `sort` pour trier une liste de listes de booléens.

❑ **Q17** – Écrire en langage `Python` une fonction `elim_double(L)` non récursive, de complexité linéaire en la taille de L, qui élimine les éléments apparaissant plusieurs fois dans une liste triée L et renvoie la liste triée obtenue. Par exemple `elim_double([1, 1, 3, 3, 3, 7])` doit renvoyer la liste `[1, 3, 7]`.

On dispose de la fonction suivante :

```
1  def doublons(liste):
2      if len(liste)>1:
3          if liste[0] != liste[1]:
4              return [liste[0]] + doublons(liste[1:])
5          del liste[1]
6          return doublons(liste)
7      else:
8          return liste
```

❑ **Q18** – Que retourne l'appel suivant ?

```
doublons([1, 1, 2, 2, 3, 3, 3, 5])
```

❑ **Q19** – Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste *non* triée ? Justifier.

❑ **Q20** – La fonction `recherche` donnée en annexe permet d'établir si la configuration correspondant à `but` est atteignable en partant de l'état `init`. Préciser le type de retour de la fonction `recherche`, le type des variables `but` et `espace`, ainsi que le type de retour de la fonction `successeurs`.

❑ **Q21** – Afin d'améliorer l'efficacité du test `if but in espace`, ligne 10 de l'annexe, on propose de le remplacer par `if in1(but, espace)` ou bien par `if in2(but, espace)`, avec `in1` et `in2` deux fonctions définies ci-dessous. On considère que le paramètre `liste` est une liste triée par ordre croissant.
Quel est le meilleur choix ? Justifier.

```
1  def in1(element,liste):
2      a = 0
3      b = len(liste)-1
4      while a <= b and element >= liste[a]:
5          if element == liste[a]:
6              return True
7          else:
8              a = a + 1
9      return False
10
11
12 def in2(element,liste):
13     a = 0
14     b = len(liste)-1
15     while a < b:
16         pivot = (a+b) // 2 # l'opérateur // est la division entière
17         if liste[pivot] < element:
18             a = pivot + 1
19         else:
20             b = pivot
21     if element == liste[a]:
22         return True
23     else:
24         return False
```

❑ **Q22** – Afin de comparer plus efficacement les files représentées par des listes de booléens on remarque que ces listes représentent un codage binaire où `True` correspond à 1 et `False` à 0. Écrire la fonction `versEntier(L)` prenant une liste de booléens en paramètre et renvoyant l'entier correspondant. Par exemple, l'appel `versEntier([True, False, False])` renverra 4.

❑ **Q23** – On veut écrire la fonction inverse de `versEntier`, transformant un entier en une liste de booléens. Que doit être au minimum la valeur de `taille` pour que le codage obtenu soit satisfaisant ? On suppose que la valeur de `taille` est suffisante. Quelle condition booléenne faut-il écrire en ligne 4 du code ci-dessous ?

```
1 def versFile(n, taille):
2     res = taille * [False]
3     i = taille - 1
4     while ...:
5         if (n % 2) != 0: # % est le reste de la division entière
6             res[i] = True
7         n = n // 2 # // est la division entière
8         i = i - 1
9     return res
```

❑ **Q24** – Montrer qu'un appel à la fonction `recherche` de l'annexe se termine toujours.

❑ **Q25** – Compléter la fonction `recherche` pour qu'elle indique le nombre minimum d'étapes à faire pour passer de `init` à `but` lorsque cela est possible. Justifier la réponse.

Partie VI. Base de données

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.
La base de données du réseau routier est constituée des relations suivantes :

- Croisement(<u>id</u>, longitude, latitude)
- Voie(<u>id</u>, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (`id`) d'un croisement donné.

❑ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

❑ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.

❑ **Q28** – Que renvoie la requête SQL suivante ?

```
1 SELECT V2.id_croisement_fin
2 FROM   Voie as V1
3 JOIN   Voie as V2
4 ON     V1.id_croisement_fin = V2.id_croisement_debut
5 WHERE  V1.id_croisement_debut = c
```

Annexe

```
def recherche(but, init):
    espace = [init]
    stop = False
    while not stop:
        ancien = espace
        espace = espace + successeurs(espace)
        espace.sort() # permet de trier espace par ordre croissant
        espace = elim_double(espace)
        stop = egal(ancien,espace) # fonction définie à la question 5
        if but in espace:
            return True
    return False


def successeurs(L):
    res = []
    for x in L:
        L1 = x[0]
        L2 = x[1]
        res.append( avancer_files(L1, False, L2, False) )
        res.append( avancer_files(L1, False, L2, True) )
        res.append( avancer_files(L1, True,  L2, False) )
        res.append( avancer_files(L1, True,  L2, True) )
    return res

# dans une liste triée, elim_double enlève les éléments apparaissant plus d'une fois
# exemple : elim_double([1, 1, 2, 3, 3]) renvoie [1, 2, 3]
def elim_double(L):
    # code à compléter

# exemple d'utilisation
# debut et fin sont des listes composées de deux files de même longueur impaire,
# la première étant prioritaire par rapport à la seconde
debut = [5*[False], 5*[False]]
fin  = [3*[False]+2*[True], 3*[False]+2*[True]]
print(recherche(fin,debut))
```

Fin de l'épreuve.

Le candidat idéal capable de répondre à toutes ses exigences a été rencontré, mais force est de constater que plusieurs candidats présentent des difficultés dans l'analyse d'une courbe de titrage, la gestion du temps et l'organisation de la paillasse.

Ce concours est une épreuve où l'énergie, la ténacité et le sens de l'initiative sont demandés, le jury ne pénalisera pas les éventuelles questions des candidats et y répondra si possible.

4. INFORMATIQUE

4.1. Informatique pour tous

- Remarques générales

Le sujet d'informatique commune portait sur la modélisation d'une situation de trafic routier. Le sujet balayait un spectre assez large des notions abordées lors des deux années de CPGE.

Le sujet, assez long, mais d'une difficulté raisonnable, a permis un classement très efficace des candidats, tout le spectre de notes possibles étant occupé. Certaines copies nous impressionnent favorablement par leur maîtrise et leur rapidité, d'autres nous impressionnent beaucoup moins favorablement.

Les clefs pour obtenir une note honorable à cette épreuve sont de bien maîtriser la syntaxe de base de Python et du SQL de s'assurer que les programmes font exactement - et pas « à peu près » - ce qui est demandé par l'énoncé, et de proposer des programmes concis et bien présentés/indentés.

Cette année, le jury croit utile d'attirer l'attention des futurs candidats et de leurs formateurs sur les points suivants :

- Un défaut particulièrement sensible cette année a été le **manque de concision des réponses et des programmes proposés, même pour des fonctions élémentaires**. Nous avons trop souvent été confrontés à des programmes surréalistes pour tester par exemple si une case est occupée ou non, ou bien pour vérifier que deux listes sont égales... mais nous avons trop rarement vu des programmes simples, qui étaient pourtant parfaitement accessibles grâce aux notions du programme (voir les commentaires de la question Q3, symptomatique de ce problème). Quant aux questions sur l'évolution de la file de voitures, on a parfois dû subir des programmes de 20 lignes ou plus avec des disjonctions de cas inutiles alors que 2 lignes suffisaient en maîtrisant la syntaxe de base de Python. Notre message principal aux candidats sera donc, concernant les programmes qu'ils proposent :

  ALLEZ AU PLUS SIMPLE !

- Dans le même esprit, le jury demande explicitement aux candidats d'éviter de dépenser leur précieux temps d'épreuve à commenter leurs programmes, tout particulièrement les programmes relativement simples. Ce qui est parfaitement compréhensible et souhaitable avec des programmes complexes dans un cadre industriel ou de recherche ne l'est plus dans le cadre contraint d'une épreuve de concours en temps limité. La perte de temps qui résulte de commentaires superflus a pénalisé de nombreux candidats : nous tenions donc à faire cette mise au point.

- Concernant le détail de la syntaxe, de nombreux candidats accordent une attention trop limitée aux bornes des itérateurs. Toute erreur sur ce point est sanctionnée, notamment sur les questions simples où il n'est pas acceptable de lire **for i in range(len(L)+1) :** pour parcourir une liste de longueur **len(L)**. Attention également à **range(p,q,-1)** souvent mal maîtrisé (on rappelle que **list(range(p,q,-1))** renvoie **[p,p-1,… ,q+1]** et qu'il est nécessaire que **p>q** si on veut une liste non vide).

- Une faute grave a été vue dans un nombre appréciable de copies : l'utilisation de **while** à la place de **if**. Cette confusion est sanctionnée lourdement.

- Une erreur de base vue un certain nombre de fois concerne la gestion des listes et de l'affectation des éléments. On a lu de trop nombreuses fois des instructions telles que :
  L=[]
  L[0]=b
  qui ne fonctionne évidemment pas.
  Quant à l'ajout d'un élément en fin de liste, nous avons trop souvent vu **L=L+a** à la place de **L=L+[a]**. De plus, trop de candidats pensent à tort que **L.append(L1)** est équivalent à **L+L1 :** seule cette dernière est correcte.

- Les candidats doivent savoir que l'instruction **L2=L** ne suffit pas à créer une liste **L2** indépendante de **L**.
- Enfin, sur la présentation, le jury insiste tout particulièrement sur la lisibilité de l'écriture - il arrive quelquefois que nous renoncions à tenter de déchiffrer une réponse si elle est très mal écrite - et sur la clarté de l'indentation (les lignes verticales de la feuille sont souvent largement suffisantes si le candidat les utilise bien), celle-ci étant cruciale pour vérifier la validité des programmes.

- Commentaires par question

Q 1 : Rappelons aux candidats que 0 et 1 ne sont pas des valeurs de variable booléenne.
Q 2 : Que de réponses compliquées ou même fausses sur cette question élémentaire ! En particulier, de nombreux candidats pensent que définir une fonction suffit à créer une liste. Ce n'est le cas que si on appelle la fonction…
Q 3 : La solution la plus simple, **def occupe(L,i) : return L[i]** n'a été proposée que par une minorité de candidats. De très nombreux candidats proposent :

```
def occupe(L,i) :
    if L[i]==True :
        return True
    elif L[i]==False :
        return False
```

certes correcte, mais qui montre qu'il y a encore une compréhension imparfaite de la puissance du langage. Parmi les réponses fausses, on a souvent vu une boucle for complètement hors de propos, qui nous montrait dès la troisième question que le candidat ne comprenait pas bien ce qu'il faisait.
Q 4 : Question en général correctement traitée, mais qui a donné lieu à des réponses parfois surprenantes… on a ainsi vu des candidats faisant de complexes raisonnement de dénombrement, allant même parfois jusqu'à un raisonnement par récurrence, qui s'assimilait ici à une perte de temps au vu de la simplicité de la question et de la justification attendue.

Q 5 : Parmi les fautes récurrentes relevées sur cette question, un **return True** mal placé qui faisait sortir de la boucle avant d'avoir testé tous les éléments, une double boucle **for** inutile, et quelques (heureusement rares) candidats qui semblent penser que deux listes de mêmes longueurs sont forcément égales…
Certains utilisent volontiers **assert len(L1)==len(L2)** en début de programme. Cela sert au débogage d'un programme en renvoyant une exception (notion hors programme), mais ne renvoie pas un booléen comme demandé.
Q 6 : Question bien traitée en général, même si on a vu quelques complexités curieuses ou irréalistes.
Q 7 : Question assez mal traitée au regard de sa faible difficulté. Le sujet de l'an dernier proposait pourtant une question similaire. Nous rappelons aux candidats que « True ou False » n'est pas un type de variable.
Q 8 : De nombreux candidats ont répondu par un dessin. Cette fonction ne renvoyait pas un dessin, mais une liste de booléens.
QQ 9 -10 : Certains candidats proposent des réponses exactes et concises. À l'inverse, d'autres proposent des solutions extrêmement complexes, avec des indices dans les itérateurs et les listes difficiles à interpréter. On passe sur les quelques candidats qui ont fait reculer les voitures au lieu de les faire avancer... Dans ces deux questions, il fallait créer une nouvelle liste indépendante de **L** (**L2=L** ne suffisait donc pas), et faire bien attention aux indices dans les découpages de listes (non, le dernier élément de **L[0:m]** n'est pas **L[m]**, et **L[len(L)]** n'existe pas…).
Q 11 : Question plutôt difficile que certains bons candidats ont remarquablement traitée. Même si on percevait la nécessité de rechercher la première case accessible et faire avancer les voitures situées avant, il fallait également faire attention à la gestion du booléen b : le premier élément de la liste ne prenait pas automatiquement la valeur de b.
Q 12 : Pour traiter cette question de manière raisonnable, il fallait bien entendu judicieusement utiliser les fonctions définies dans les questions 9, 10 et 11. Les candidats qui ont voulu faire autrement se sont souvent embourbés dans des programmes trop complexes, ou dans des disjonctions de cas déraisonnables.
Q 13 : Question assez simple, attention toutefois à la syntaxe correcte de la réponse : cette fonction renvoie non pas deux listes, mais une seule liste constituée de deux sous-listes.
Q 14 : Question qualitative bien traitée.
Q 15 : De nombreux candidats ont répondu 8 étapes ou 10 étapes en allant trop vite.
Q 16 : Le jury apprécie des réponses claires et non ambiguës sur ces questions qualitatives. Le doute ne bénéficie pas au candidat.
Q 17 : Question en apparence simple, mais qui contenait de nombreux pièges dans lesquels beaucoup de candidats sont tombés. La précision de l'énoncé « de complexité linéaire » a échappé à beaucoup, y compris à ceux qui pensent s'en sortir en utilisant **in**, **del** ou **.pop** ou… qui ne sont certainement pas de complexité unitaire ! Attention également à la gestion des indices : on a souvent vu des fonctions qui renvoyaient une liste ne contenant pas le premier ou le dernier élément de la liste primaire.
QQ 18-19 : Questions assez bien traitées.
Q 20 : Un certain nombre de candidats ont vu en but un entier, ce n'était pas le cas.
Q 21 : Pour obtenir tous les points sur cette question, il fallait clairement reconnaître un algorithme de recherche par dichotomie et citer les complexités des deux algorithmes proposés. Les rares candidats qui ont fait valoir que « **in1** est plus optimisé, car il y a moins d'instructions » nous ont laissés songeurs.
Q 22 : Quelques candidats ont traité correctement cette question, mais il fallait faire attention aux puissances de 2 employées.
Q 23 : Question rarement traitée.
Q 24 : Trop de candidats écrivent « il y a un **while** donc la boucle s'arrête nécessairement »… encore faut-il le prouver !

Q 25 : Question rarement traitée, mais souvent de manière satisfaisante.
Q 26 : Cette requête simple, qui ne nécessitait pas de jointure, a été relativement mal traitée.
Q 27 : Les candidats semblent maîtriser plutôt mieux que l'an dernier la syntaxe d'une jointure, mais ne maîtrisent pas encore assez la notion : combien de fois a-t-on vu **ON Croisement.id=Voie.id** dans la condition de jointure ?
Q 28 : Question assez peu traitée.

Quelques distinctions honorifiques :
Prix « rencontre du troisième type » :
« Cette fonction renvoie un booléen qui est une chaîne de caractère de type str. »
Prix de l'extrapolation douteuse :
« Les voitures se sont montées dessus dans l'intersection. »
Prix Jean-Claude Van Damme :
« Il n'existe qu'une seule file de longueur n car toute file de longueur n est contenue dans elle-même. »

4.2. Informatique — filière MP

- Remarques générales

Le sujet est constitué d'un problème en deux parties : la première partie traite des automates et la seconde porte sur la programmation. L'ensemble permet de bien évaluer l'acquisition du programme des deux années de classe préparatoire.
Les candidats abordent les deux parties dans leur grande majorité. Ils finissent parfois le sujet (en passant les questions difficiles). Quelques (rares) excellentes copies ont pu être lues.
La présentation des copies est globalement satisfaisante.
Nous avons pu constater peu d'efforts de rédaction des questions théoriques (automate ou calcul de complexité).
Beaucoup de candidats ne donnent pas d'arguments ou se contentent d'arguments superficiels.

- Partie automate

Le but de cette partie est de manipuler les automates en relation avec une certaine classe de langages.
Le sujet permet d'évaluer les connaissances des candidats en matière d'automates et leur capacité à identifier le langage accepté par un automate.
Des exemples sont traités dans un premier temps. Ceci permet de mettre en place une approche formelle dans le cas général.
La difficulté est essentiellement dans les justifications formelles attendues pour certaines questions.

- Quelques remarques

QQ 1-6
De très rares erreurs. L'association langage-automate est bien maîtrisée.
QQ 7-8
Des erreurs plus fréquentes sans doute dues à une mauvaise interprétation des opérations ensemblistes utilisées.
QQ 9-12
Ces questions nécessitent une argumentation formelle fine et rigoureuse.

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)
```

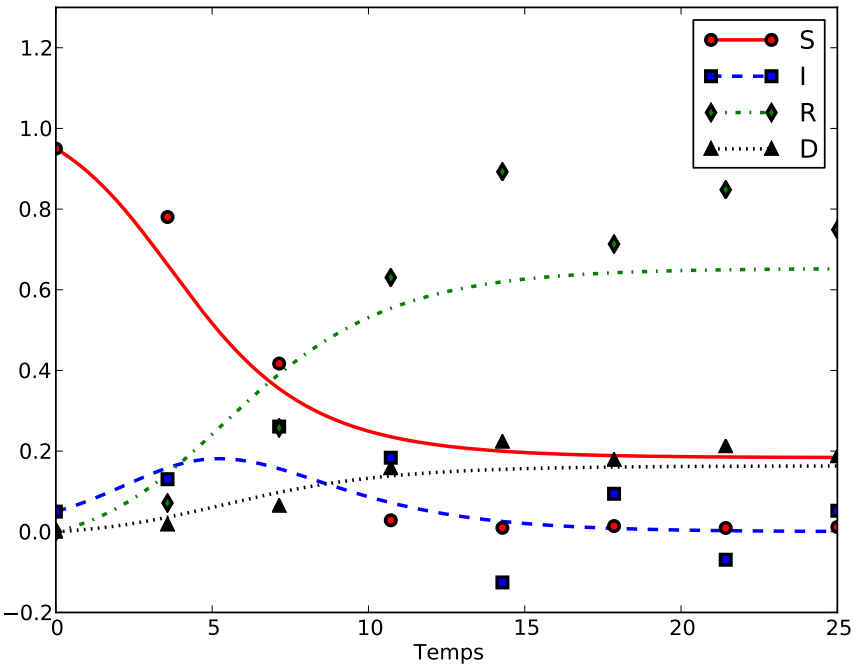


FIGURE 1 – Représentation graphique des quatre catégories S, I, R et D en fonction du temps pour $N = 7$ (points) et $N = 250$ (courbes).

❑ **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 :

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

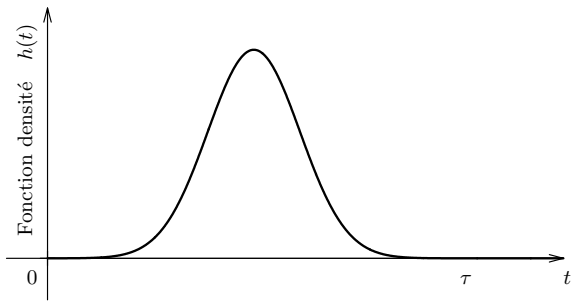


FIGURE 2 – Exemple d'une fonction de densité.

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 ?

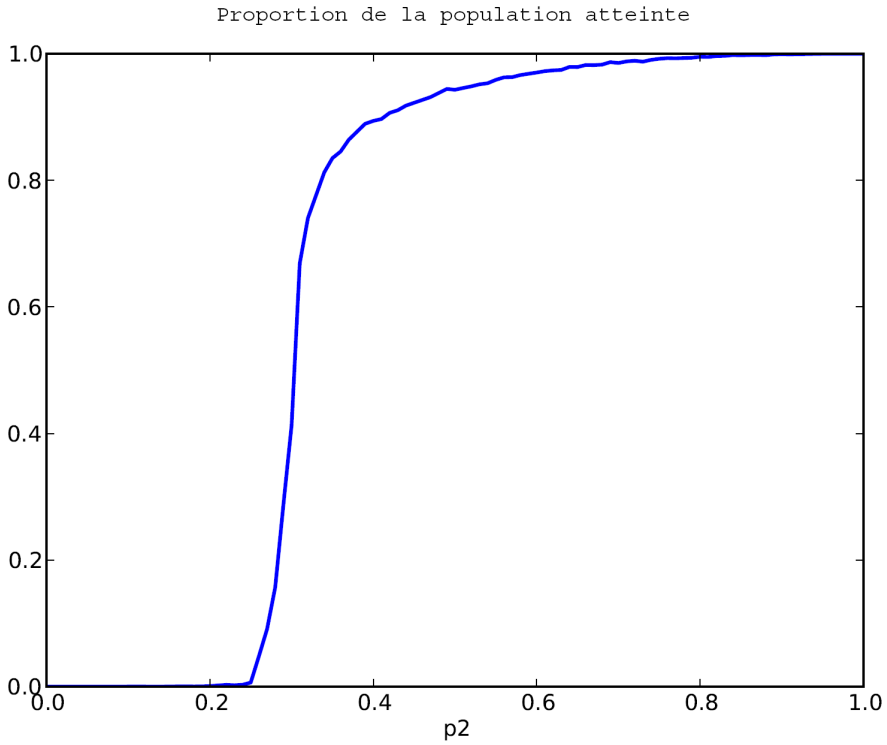


FIGURE 3 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité p_2.

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 !

A 2015 INFO.

ÉCOLE DES PONTS PARISTECH.
SUPAERO (ISAE), ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH
MINES DE SAINT-ÉTIENNE, MINES NANCY,
TÉLÉCOM BRETAGNE, ENSAE PARISTECH (Filière MP).
ÉCOLE POLYTECHNIQUE (Filière TSI).

CONCOURS 2015

ÉPREUVE D'INFORMATIQUE

(Durée de l'épreuve : 1h30 heures)
L'usage de l'ordinateur ou de la calculatrice est interdit.

Sujet mis à la disposition des concours :
Cycle international, Écoles des Mines, TELECOM INT, TPE-EIVP.

L'énoncé de cette épreuve comporte 10 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.

Tests de validation d'une imprimante

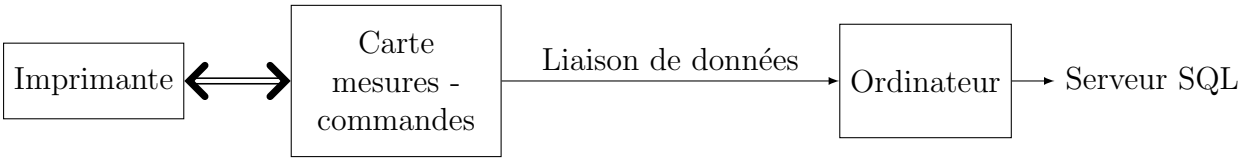
Le sujet comporte des questions de programmation. *Le langage à utiliser est* `Python` *dans les parties I à IV. Dans la partie V, on demande d'utiliser* `Scilab`.

Introduction

Les imprimantes sont des systèmes mécatroniques fabriqués en grande série dans des usines robotisées. Pour améliorer la qualité des produits vendus, il a été mis en place différents tests de fin de chaîne pour valider l'assemblage des produits. Pour un de ces tests, un opérateur connecte l'outil de test sur la commande du moteur de déplacement de la tête d'impression et sur la commande du moteur d'avance papier. Une autre connexion permet de récupérer les signaux issus des capteurs de position.

Différentes commandes et mesures sont alors exécutées. Ces mesures sont envoyées par liaison de données sous la forme d'une suite de caractères ASCII vers un ordinateur.

Cet ordinateur va effectuer différentes mesures pour valider le fonctionnement de l'électromécanique de l'imprimante. L'ensemble des mesures et des analyses est sauvegardé dans un fichier texte. Cette sauvegarde s'effectue dans une base de données et, afin de minimiser l'espace occupé, les fichiers sont compressés. La base de données permet à l'entreprise d'améliorer la qualité de la production par diverses études statistiques.



Rappels et définitions :

- Une liste commence par un crochet `[` et se termine par un crochet `]`. Les éléments d'une liste sont ordonnés (indexés).
- On pourra utiliser la surcharge de l'opérateur + : `['a']+['b']=['a','b']`.
- Un dictionnaire définit une relation une à une entre des clés et des valeurs. Celui-ci se note entre accolades `{}`.
- Un tuple est une collection d'éléments ordonnés comme dans une liste mais une fois le tuple créé, ses éléments ne peuvent pas être modifiés indépendamment les uns des autres. Il se note entre parenthèses `()`. Exemple : (4,'e',[1,3])

I Réception des données issues de la carte d'acquisition

Les mesures sont faites à l'aide de convertisseurs analogique/numérique (CAN). Le résultat de conversion est codé sur 10 bits signés en complément à 2.

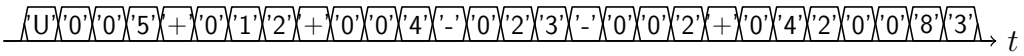
Q1. Quelle plage de valeurs entières pourra prendre le résultat de la conversion ?

Q2. Si on considère que les valeurs analogiques converties s'étendent en pleine échelle de -5V à 5V, quelle est la résolution de la mesure en volt ?

Une liaison série asynchrone permet la communication entre la carte de commande/acquisition et le PC. Les échantillons correspondant à une mesure sont envoyés par la carte électronique sous la forme d'une trame (suite de caractères ASCII). Cette suite de caractères se présente sous la forme suivante :

- un entête qui permet d'identifier la mesure sur un caractère ('U' tension moteur, 'I 'courant moteur, 'P' position absolue),
- le nombre de données envoyées (3 caractères),
- les données constituées des mesures brutes issues de la conversion analogique-numérique, chaque mesure étant codée à l'aide du caractère '+' ou '-' suivi de 3 caractères pour la valeur absolue,
- un checksum, somme des valeurs absolues des données précédentes modulo 10000 sur 4 caractères. Le nombre de données transmises n'est pas inclus dans le checksum.

Exemple : Mesure de la tension sur 5 échantillons.

Caractères reçus : 

La commande `carac_recus=com.read(nbre_car)` permet de récupérer *nbre_car* caractères reçus sous la forme d'une chaîne de caractères. En supposant que les caractères reçus correspondent à l'exemple précédent, après l'exécution de `carac_recus=com.read(5)`, la variable `carac_recus` contiendra la chaîne "U005+".

Après une nouvelle exécution de `carac_recus=com.read(3)`, la variable `carac_recus` contiendra la chaîne "012".

Q3. Écrire une fonction `lect_mesures()` en langage `Python` qui retourne une liste contenant : le type de la mesure ('U','I' ou 'P'), une liste contenant l'ensemble des valeurs des mesures reçues et le checksum. Exemple : ['U',[12,4,-23,-2,42],83]. Cette fonction doit attendre que le caractère d'entête reçu soit correct ('U','I' ou 'P') avant de réaliser le stockage des informations dans la liste qui sera retournée.

Q4. On suppose que toutes les mesures sont disponibles dans la liste `mesures[]`, et le checksum reçu dans la variable `CheckSum`. Écrire une fonction `check(mesure,CheckSum)` en langage `Python` qui retourne True si la transmission présente un checksum valide et False sinon.

Q5. Les mesures étant dans la liste `mesures`, écrire une fonction `affichage(mesure)` en langage `Python` qui produit un affichage graphique comme représenté en Figure 1, sachant que la résolution de la conversion analogique-numérique du courant est de 4 mA et que les mesures ont été effectuées toutes les 2 ms. On ne demande pas de légender les axes ni de donner un titre à la figure. On suppose que les bibliothèques nécessaires ont été importées.

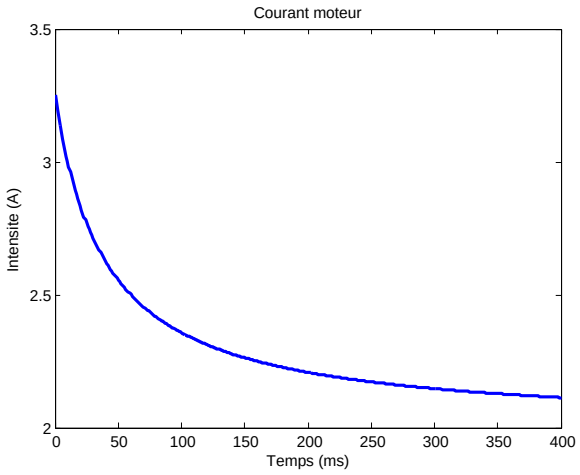


FIGURE 1 – Affichage de la mesure.

II Analyse des mesures

La suite des valeurs de mesure du courant en Ampère du moteur de la tête d'impression est contenue dans une liste. Les mesures ont été effectuées toutes les 2 ms. Ces mesures sont disponibles dans la liste `mesure`. Deux traitements permettent de valider le fonctionnement de l'imprimante :

- Le calcul de la valeur moyenne I_{moy} du signal $I(t)$ sur la durée d'acquisition.

$$I_{moy} = \frac{1}{t_{final}} \int_0^{t_{final}} I(t)\,\mathrm{d}t$$

- Le calcul de l'écart type I_{ec} du signal $I(t)$ sur la durée d'acquisition.

$$I_{ec} = \sqrt{\frac{1}{t_{final}} \int_0^{t_{final}} (I(t) - I_{moy})^2\,\mathrm{d}t}$$

Q6. Écrire une fonction en langage `Python` qui retourne I_{moy} après l'avoir calculée par la méthode des trapèzes.

Q7. Écrire une fonction en langage `Python` qui retourne I_{ec} après l'avoir calculé en utilisant la fonction précédente.

III Base de données

Une représentation simplifiée de deux tables de la base de données qu'on souhaite utiliser est donnée ci-dessous :

testfin

nSerie	dateTest	...	Imoy	Iec	...	fichierMes
230-588ZX2547	2012-04-22 14-25-45		0.45	0.11		mesure31025.csv
230-588ZX2548	2012-04-22 14-26-57		0.43	0.12		mesure41026.csv
⋮	⋮	⋮	⋮	⋮	⋮	⋮

production

Num	nSerie	dateProd	type
20	230-588ZX2547	2012-04-22 15-52-12	JETDESK-1050
21	230-588ZX2549	2012-04-22 15-53-24	JETDESK-3050
⋮	⋮	⋮	⋮

Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque imprimante. A la fin des tests de chaque imprimante, les résultats d'analyse ainsi que le fichier contenant l'ensemble des mesures réalisées sur l'imprimante sont rangés dans la table `testfin`. Lorsqu'une imprimante satisfait les critères de validation, elle est enregistrée dans la table `production` avec son numero de série, la date et l'heure de sortie de production ainsi que son type.

Q8. Rédiger une requête SQL permettant d'obtenir les numéros de série des imprimantes ayant une valeur de Imoy comprise strictement entre deux bornes Imin et Imax.

Q9. Rédiger une requête SQL permettant d'obtenir les numéros de série, la valeur de l'écart type et le fichier de mesures des imprimantes ayant une valeur de Iec strictement inférieure à la valeur moyenne de la colonne Iec.

Q10. Rédiger une requête SQL qui permettra d'extraire à partir de la table `testfin` le numéro de série et le fichier de mesures correspondant aux imprimantes qui n'ont pas été validées en sortie de production.

IV Préparation du fichier texte avant envoi : la compression

Le fichier de résultat va être stocké sous la forme d'un fichier binaire. Une des étapes de l'algorithme de compression utilise le codage de Huffman.

IV.1 Présentation :

Le codage de Huffman utilise un code à longueur variable pour représenter un symbole de la source (par exemple un caractère dans un fichier). Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents. La première étape du codage de Huffman consiste à créer un

dictionnaire contenant la liste des caractères présents dans le texte, associé à leur fréquence dans ce texte. Exemple : "AABCDCCEF" donnera `{'A':2,'B':1,'C':3,'D':1,'E':1,'F':1}`. La deuxième étape consiste à construire un arbre de Huffman qui permet ensuite de coder chaque caractère.

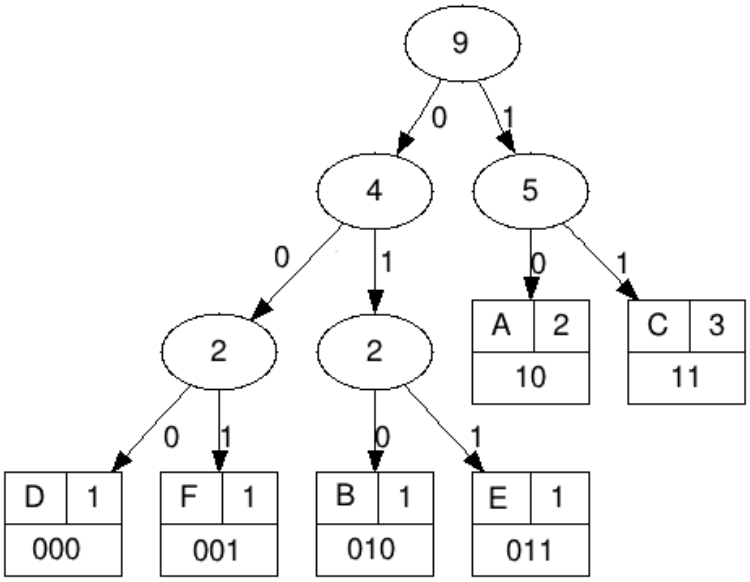


FIGURE 2 – Arbre de Huffman.

Notre texte "AABCDCCEF" de 9 caractères ASCII (72 bits) sera ainsi codé en binaire "10 10 010 11 000 11 11 011 001" (22 bits).

Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe un poids valant son nombre d'occurrences. Puis l'arbre est créé suivant un principe simple : on associe à chaque fois les deux nœuds de plus faibles poids pour créer un nœud dont le poids équivaut à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine. On associe ensuite par exemple le code 0 à la branche de gauche et le code 1 à la branche de droite.

Pour obtenir le code binaire de chaque caractère, on descend sur la Figure 2 de la racine jusqu'aux feuilles en ajoutant à chaque fois au code un 0 ou un 1 selon la branche suivie.

Pour pouvoir être décodé par l'ordinateur, l'arbre doit aussi être transmis.

Le code `Python` permettant de construire un arbre de Huffman et de coder chaque caractère est fourni en *annexe*.

Les feuilles de l'arbre de Huffman (leaf) sont codées sous la forme de tuple avec comme premier élément le caractère et comme deuxième élément le poids. Pour l'arbre donné en exemple en Figure 2, on aura 6 tuples : `('A',2), ('B',1), ('C',3), ('D',1), ('E',1) et ('F',1).`

Q11. La documentation de l'instruction `isinstance(object,classinfo)` décrit le fonctionnement suivant : "Return True if the object is an instance of the classinfo argument. If object is not a class instance of the given type, the function returns False." Décrire succinctement le rôle des fonctions suivantes et indiquer le type de la variable retournée :

- `test()`
- `get1()`
- `get2()`

IV.2 Analyse des fonctions `make_huffman_tree()` et `freq_table()`

Q12. Donner le contenu des variables `node1` et `node2` suite à l'exécution des commandes
`node1=make_huffman_tree(('F',1),('E',1))`.
`node2=make_huffman_tree(('D',1),('B',1))`.

Q13. De même, donner le contenu de la variable `node3` suite à l'éxécution de la commande `node3=make_huffman_tree(node1,node2)`.

Q14. Donner le contenu de la variable `f` suite à l'éxécution de la commande `f=freq_table('AABBCB')`.

IV.3 Analyse de la fonction `insert_item()`

Cette fonction permet d'insérer un nœud ou une feuille dans une liste de nœuds et de feuilles triés par poids croissant, en conservant l'ordre de tri.

Q15. Quelle est la particularité de cette fonction ?

Q16. Montrer qu'un invariant d'itération est "tous les éléments de la sous liste `lst[0 à pos-1]` ont un poids inférieur à celui de la variable item". On démontrera qu'il est vrai à l'initialisation, puis à chaque itération, sachant que cette fonction doit être appelée avec l'argument d'entrée `pos=0` produisant ainsi une liste vide.

IV.4 Analyse de `build_huffman_tree()`

Q17. D'après la description précédente et le résultat de la question Q13, commenter la fonction de manière à expliquer comment l'arbre de Huffman est construit. On demande de proposer des rédactions pour les commentaires correspondant aux lignes ## 5, ## 6, ## 7, ## 8 dans la fonction `build_huffman_tree()`.

Q18. Donner le nombre de tests dans le meilleur des cas et dans le pire des cas effectués dans la fonction `insert_item()` pour une liste `lst` contenant n éléments. Les résultats devront être justifiés.

Q19. Donner la complexité en temps dans le meilleur des cas et dans le pire des cas de la fonction `build_huffman_tree` pour une liste `lst` contenant n éléments en tenant compte de la fonction `insert_item`. On négligera le coût en complexité de la fonction `lst.sort`. Les résultats devront être justifiés.

Q20. Donner le contenu de la variable `htree` après l'éxécution de la commande `htree=build_huffman_tree("ZBBCB")`.

Q21. Dessiner l'arbre de Huffman correspondant à `htree` de la question précédente en vous inspirant de la Figure 2.

V Simulation physique

Une possible source de pannes dans chaque imprimante est la défectuosité d'un moteur particulier. On suppose disposer d'enregistrements du signal d'entrée e et du signal de sortie s de cet élément. Ces variables sont supposées satisfaire une équation différentielle linéaire du premier ordre

$$\frac{d}{dt}s = -k\frac{s-e}{10}$$

où k est un paramètre réel strictement positif. On va chercher à vérifier le bon fonctionnement du moteur en analysant des simulations.

Q22. Écrire en langage Scilab une fonction qui calcule de manière approchée la solution de l'équation différentielle précédente pour le signal $e(t) = \sin(t)$, avec $s(0) = 0$, sur l'intervalle $t \in$ [0,10] pour une valeur quelconque de k. La fonction doit retourner une estimation de $s(t)$ aux instants [0, 0.1, 0.2,..., 10].

Q23. Trois valeurs sont possibles pour le paramètre k : 0.5, 1.1 et 2. On décide de déclarer le moteur défectueux si aucune de ces valeurs ne permet d'expliquer l'enregistrement s réalisé expérimentalement aux instants [0, 0.1, 0.2,..., 10]. Définir un code Scilab utilisant la fonction précédemment définie pour réaliser une validation ou une invalidation de la défectuosité du moteur suivant un critère à proposer.

Annexe

Huffman tree :

```
##############################################################################
### Auto Generation of Huffman Trees
##############################################################################

def make_leaf(symbol, weight):
    return (symbol, weight)

def test(x):
    return isinstance(x, tuple) and \
            len(x) == 2 and \
            isinstance(x[0], str) and \
            isinstance(x[1], int)

def get1(x):
    return x[0]

def get2(x):
    return x[1]

def get3(huff_tree):
    return huff_tree[0]

def get4(huff_tree):
    return huff_tree[1]

def get5(huff_tree):
    if test(huff_tree):
        return [get1(huff_tree)] #Attention le symbole est dans une liste
    else:
        return huff_tree[2]

def get6(huff_tree):
    if test(huff_tree):
        return get2q(huff_tree)
    else:
        return huff_tree[3]

def make_huffman_tree(left_branch, right_branch):
    return [left_branch,
            right_branch,
            get5(left_branch) + get5(right_branch),
            get6(left_branch) + get6(right_branch)]

### entr\'{e}e : string txt et retourne un dictionnaire contenant chaque
### caractere avec son occurrence dans le string txt
def freq_table(txt):
    ftble = {}
    for c in txt:
        if c not in ftble:
            ftble[c] = 1
        else:
            ftble[c] += 1
    return ftble
```

```
### Fonction de comparaison qui permet de comparer
### les noeuds de Huffman entre eux selon leur occurrence.
def freq_cmp(node1, node2):
    freq1, freq2 = get6(node1), get6(node2)
    if freq1 < freq2:
        return -1
    elif freq1 > freq2:
        return 1
    else:
        return 0

### ins\'{e}re un item \'{a} sa place appropri\'{e}e dans une liste de
   noeuds et feuilles
def insert_item(item, lst, pos):
    if pos == len(lst):
        lst.append(item)
    elif freq_cmp(item, lst[pos]) <= 0 :
        lst.insert(pos, item)
    else:
        insert_item(item, lst, pos+1)
    return

### Construction de l'arbre de Huffman
def build_huffman_tree(txt):
    ### 1. construire une table des occurrences \'{a} partir de txt
    ftble = freq_table(txt)
    ### 2. obtenir la liste des feuilles de Huffman
    lst = list(ftble.items())
    ### 3. classer leaf_lst par occurrence de la plus petite \'{a} la plus
    ###    grande
    lst.sort(key=lambda lst: lst[1])
    ### 4. construction de l'arbre de huffman
    if len(lst) == 0:
        return None
    elif len(lst) == 1:
        return lst[0]
    else:
        ## 5. ------------------------------------------------------------
          while len(lst) > 2:
            ## 6. ------------------------------------------------------------
            new_node = make_huffman_tree(lst[0], lst[1])
            ## 7. ------------------------------------------------------------
            del lst[0]
            del lst[0]
            ## 8. ------------------------------------------------------------
            insert_item(new_node, lst, 0)
        else:
            return make_huffman_tree(lst[0], lst[1])
```

Manual and Auto Generation of Huffman Trees in Python, Vladimir Kulyukin http://vkedco.blogspot.fr/2012/02/manual-and-auto-generation-of-huffman.html
Huffman tree generator http://huffman.ooz.ie/

Informatique pour tous

PRÉSENTATION DU SUJET

Le sujet concerne des tests de validation d'une imprimante. L'étude est menée en s'appuyant sur la démarche suivante :

I) Réception des données issues de la carte d'acquisition

Afin de pouvoir réaliser des tests sur les imprimantes, des mesures sont faites à l'aide de cartes d'acquisition. Ces mesures étant codées sur un nombre limité de bits, il est demandé aux candidats de déterminer la précision des mesures.
Les informations sont alors envoyées à un PC par la carte d'acquisition sous la forme d'une trame (suite de caractères). Les candidats doivent créer une fonction permettant de lire les caractères de cette trame pour les stocker sous forme de liste. Un « checksum » est alors réalisé pour vérifier la validité de la transmission avant de faire un affichage des résultats d'acquisition.

II) Analyse des mesures

L'objectif de cette partie est de mettre en place le traitement des mesures obtenues nécessaires à la validation ultérieure des imprimantes. Il est demandé aux candidats de calculer, par la méthode des trapèzes, la moyenne des valeurs et l'écart-type.

III) Base de données

Il s'agit ici d'interroger une base de données. Il est demandé de rédiger les requêtes SQL permettant d'identifier les imprimantes satisfaisant des critères de moyenne et d'écart-type puis d'identifier les imprimantes non valides.

IV) Préparation du fichier texte avant envoi : la compression

Le fichier de résultats est compressé avant envoi par le codage de Huffman. La fonction de codage est donnée. Les candidats doivent analyser cette fonction et évaluer sa complexité.

V) Simulation physique

Afin de détecter les pannes d'un moteur, on suppose ici qu'on dispose d'enregistrements du signal d'entrée et du signal de sortie. L'équation différentielle du modèle que doit respecter le système est donnée. Les candidats doivent résoudre numériquement cette équation différentielle, puis écrire une fonction permettant de comparer les solutions de cette équation modèle avec les enregistrements afin de déterminer la défectuosité du moteur.

ANALYSE DES COPIES DES CANDIDATS ET DES RÉSULTATS

Q1 - Cette question a été globalement mal traitée. Peu d'élèves maîtrisent la notion de « signe » d'un nombre entier et une grande majorité a répondu à cette question sans prendre en compte cet aspect. De plus, à la question « Quelle plage de valeurs... », beaucoup ont compris « Combien de valeurs... », alors que c'étaient bien les bornes qui étaient attendues.

Ceux qui répondent correctement mais sans exprimer numériquement 2^9 n'ont pas le maximum des points.

Q2 - Bon nombre de réponses exactes (ou presque, en divisant par 2^{10} et non par $2^{10} – 1$).

Q3 - Les élèves ont souvent mal compris la commande `com.read(nb_car)` et l'ont mal utilisée. Ils ont souvent aussi essayé de calculer la valeur de `checksum` au lieu de lire cette valeur. À signaler aussi que, pour tester si le premier caractère lu appartient à {'U', 'I', P'}, beaucoup d'élèves confondent `and` et `or`. Enfin, beaucoup d'élèves ont fait des opérations compliquées pour transformer une chaîne de caractères en un `int`.

Q4 - La question est bien traitée quand elle est comprise. Une ambiguïté entre `mesures` et `mesure` a gêné certains élèves. Beaucoup de candidats ne connaissent pas les opérateurs de base permettant de calculer la valeur absolue et le modulo. Dans ce cas, un certain nombre ont proposé l'écriture de fonctions correspondantes, ce qui, même en étant compté juste, leur a fait perdre du temps.

Q5 - Parmi les élèves qui ont abordé cette question, une grande majorité a oublié de prendre en compte la résolution de conversion analogique-numérique.

Q6 - Un grand nombre d'élèves connaissent la méthode des trapèzes mais n'arrivent pas à la mettre en œuvre dans le contexte du problème. Les élèves auraient sans doute mieux su répondre si le paramètre (`mesure`) de la fonction à définir avait été indiqué.

Q7 - Dans cette question, on demandait aux étudiants de réutiliser la fonction précédente pour calculer un écart-type. Cette question a été mieux traitée que la précédente. Un certain nombre de candidats ont proposé des solutions pertinentes sans avoir fait la précédente. Si le calcul de la moyenne a été bien fait dans l'ensemble, la différence entre les valeurs de la mesure et la moyenne, en revanche, a fait l'objet d'une erreur courante : on ne peut pas soustraire une variable de type « float » d'un objet de type « liste ». Quelques rares étudiants ont contourné le problème en utilisant les tableaux du module *Numpy*.

Q8 - Cette question ainsi que les deux suivantes ont été généralement abordées. Les réponses sont globalement pertinentes. Un certain nombre de candidats confondent les requêtes SQL (« `SELECT … FROM…` ») avec l'importation des modules python (« `from… import…` »). Certains élèves sont gênés par le fait d'exprimer la requête en fonction de *Imin* et *Imax* sous forme littérale.

Q9 - De nombreux candidats connaissent la fonction d'agrégation `AVG()`, cependant peu savent la mettre en œuvre. On rappelle que les fonctions d'agrégation sont des fonctions s'appliquant sur un ensemble d'enregistrements. Deux requêtes imbriquées étaient attendues ici.

Q10 - Dans cette question, l'objet était de récupérer les données d'une table qui ne figurent pas dans une autre. Beaucoup de candidats ont bien compris qu'il fallait faire une différence,

mais la structure globale de la requête proposée est généralement incorrecte. Les candidats ne semblent pas maîtriser le résultat renvoyé par une jointure.

Q11 - Certains élèves ne comprennent par l'instruction `isinstance`. Par ailleurs, ils ne donnent pas toujours les types des valeurs retournées, alors que l'énoncé les demande.

Q12 et Q13 - Ces questions sont souvent bien traitées par les élèves qui les ont atteintes et qui ont réussi à comprendre les définitions et le principe de l'algorithme de Huffman.

Q14 - On attendait dans la réponse une association entre les trois lettres 'A', 'B' et 'C' et leur nombre d'occurrences. Trop d'élèves répondent {2, 3, 1}, ce qui n'est pas pertinent, même sans connaître la définition d'un dictionnaire (définition donnée dans le sujet).

Q15 - La question étant ouverte, toute réponse « pertinente » a été acceptée (la fonction est récursive, ou la fonction est de complexité linéaire, ou la fonction ne renvoie rien).

Q16 et Q18 - L'écriture récursive de la fonction `insert_item` (et non en utilisant une boucle) a rendu ces questions délicates.

Q17 - Ici, on demandait aux étudiants de commenter des lignes de codes. Trop d'étudiants se sont contentés de paraphraser le code (exemple : « tant que la taille de la liste est supérieure à 2... »). En dépit de l'indentation, beaucoup se sont trompés de ligne à commenter.

Q18 - La question est ambiguë car d'une part la réponse semble dépendre du paramètre `pos` et d'autre part l'énoncé n'indique pas qu'on attend un ordre de grandeur du nombre de tests ; on attendait en fait l'ordre de grandeur du nombre d'opérations pour l'insertion d'une donnée dans une liste triée.

Q19 à Q21 - Peu d'élèves atteignent ces questions.

Q22 - Les quelques candidats qui se sont lancés dans cette question s'en sont plutôt bien sorti. Cependant, comme pour la question Q.6, un certain nombre a réécrit des méthodes apprises par cœur, sans les adapter au problème du sujet (on demandait ici de travailler sur des valeurs discrètes). Les réponses données dans le langage Python ont été acceptées. Comme pour la question suivante, il aurait d'ailleurs été préférable de laisser aux candidats le choix de Python ou Scilab plutôt que d'imposer Scilab dans l'énoncé.

Q23 - Question quasiment non traitée. Le critère de sélection était laissé au libre choix du candidat. Certains l'ont mentionné (sans le définir).

CONCLUSION

Le sujet, un peu long, couvre une bonne partie du programme et, dans l'ensemble, il permet de classer correctement les élèves sur cette matière. Il comportait cependant quelques erreurs résiduelles qui ont échappé aux tests, sans conséquence trop importante sur les candidats.
En moyenne, les connaissances des élèves sont approximatives et la syntaxe du langage est mal assimilée.

Les concepteurs feront en sorte que le sujet de l’an prochain soit plus abordable et corresponde mieux aux attentes exprimées par certains professeurs, en espérant aussi que les connaissances des élèves seront plus précises.