CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrice autorisée

2021

2021

Lancer de rayons

Et qu'au contraire les chats voient de nuit par le moyen des rayons qui tendent de leurs yeux vers les objets.

— René Descartes, *Discours de la méthode - La dioptrique(1637)*

Le sujet présente une méthode de génération d'images en deux dimensions représentant des objets dans l'espace, éclairés par des sources lumineuses. Cette technique appelée *lancer de rayons* s'appuie sur les lois de propagation de la lumière, ce qui lui confère une grande qualité de rendu, au prix d'un temps de calcul souvent élevé. L'augmentation de la puissance des processeurs devrait bientôt pouvoir rendre cette technique compatible avec les jeux vidéo.

Figure 1 Boite en bois et boules[1]

Ce sujet propose d'illustrer les principales caractéristiques de la méthode, en se limitant à des scènes ne comportant que des sphères et des sources lumineuses ponctuelles.

De nombreuses questions sont indépendantes, il est toutefois demandé de les traiter dans l'ordre.

Les seuls langages informatiques autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question, il est possible, et souvent souhaitable, de faire appel aux fonctions définies dans les questions précédentes.

Les modules `math` et `numpy` ont été rendus accessibles grâce à l'instruction

```
import math, numpy as np
```

Dans tout le sujet, le terme « liste » désigne une valeur de type `list`. Le terme « tableau » désigne une valeur de type `np.ndarray`. Enfin le terme « séquence » désigne une suite finie *indiçable* et *itérable*.

— *Indiçable* signifie que les éléments sont accessibles par des indices, `seq[0]` désignant le premier élément de la séquence `seq`.

[1] Exemple fourni avec le logiciel libre de lancer de rayons POV-Ray. Fichier woodbox.pov, POV-Ray scene file by Dan Farmer, sous licence Creative Commons Attribution-ShareAlike 3.0

BY-NC-SA

— *Itérable* signifie que la séquence peut être parcourue dans une boucle par `for element in seq: ...`

Un tuple d'entiers, une liste d'entiers et un tableau d'entiers sont trois exemples de « séquence d'entiers ».

Les entêtes des fonctions demandées sont annotés pour préciser les types des paramètres et du résultat. Ainsi,

```
def uneFonction(n:int, X:[float], c:str, u) -> (np.ndarray, int):
```

signifie que la fonction `uneFonction` prend quatre paramètres `n`, `X`, `c` et `u`, où `n` est un entier, `X` une liste de nombres à virgule flottante, `c` une chaine de caractères et le type de `u` n'est pas précisé. Cette fonction renvoie un couple constituée d'un tableau et d'un entier.

Il n'est pas demandé aux candidats d'annoter leurs fonctions, la rédaction pourra commencer par

```
def uneFonction(n, X, c, u):
    ...
```

De façon générale, une attention particulière sera portée à la lisibilité, la simplicité et la clarté du code proposé. L'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires seront appréciés.

Une liste de fonctions potentiellement utiles est fournie à la fin du sujet.

I Géométrie

Dans tout le sujet, l'espace est muni d'un repère $(O, \vec{u}_x, \vec{u}_y, \vec{u}_z)$ orthonormé direct. $\|\vec{v}\|$ désigne la norme euclidienne du vecteur $\vec{v}$. Le produit scalaire de deux vecteurs $\vec{v}_1$ et $\vec{v}_2$ est noté $\vec{v}_1 \cdot \vec{v}_2$, leur produit terme à terme (produit de Hadamard) est noté $\vec{v}_1 \odot \vec{v}_2$: $\vec{v}_1 \odot \vec{v}_2 = v_{1x}v_{2x}\vec{u}_x + v_{1y}v_{2y}\vec{u}_y + v_{1z}v_{2z}\vec{u}_z$.

Tout point ou vecteur de l'espace est représenté en Python par le tableau de ses trois coordonnées cartésiennes, de type `float`. Pour faciliter la compréhension, un tel tableau est considéré de type `point` quand il désigne un point et de type `vecteur` quand il désigne un vecteur : `np.array([0.,0.,0.])` est considéré de type `point` quand il représente le point O et de type `vecteur` quand il représente le vecteur nul.

Beaucoup d'opérations classiques sur les vecteurs se transposent simplement dans la syntaxe de `numpy`. Si $\vec{v}_1$, $\vec{v}_2$ sont 2 vecteurs et t un réel, représentés respectivement par `v1`, `v2` et `t`, alors $\vec{v}_1 + \vec{v}_2$, $\vec{v}_1 - \vec{v}_2$, $\vec{v}_1 \odot \vec{v}_2$ et $t\vec{v}_1$ peuvent être calculés simplement par `v1 + v2`, `v1 - v2`, `v1 * v2` et `t * v1`.

Quand il n'y a pas de confusion possible, un objet mathématique est assimilé à sa représentation en Python. Ainsi, par exemple, « la fonction `f` prend en paramètre le vecteur $\vec{v}_1$ » signifie que la fonction `f` accepte des valeurs de type `vecteur` représentant le vecteur $\vec{v}_1$.

On rappelle que la valeur du cosinus de l'angle formé par deux vecteurs unitaires correspond à la valeur de leur produit scalaire. Ainsi, si $\vec{u}_1$ et $\vec{u}_2$ sont deux vecteurs unitaires, $\cos(\widehat{\vec{u}_1, \vec{u}_2}) = \vec{u}_1 \cdot \vec{u}_2$. Par ailleurs, si $\vec{u}$ est un vecteur unitaire et $\vec{v}$ un vecteur quelconque, le projeté du vecteur $\vec{v}$ dans la direction $\vec{u}$ est donné par $(\vec{u} \cdot \vec{v})\,\vec{u}$.

Q 1. Écrire une fonction d'entête

```
def vec(A:point, B:point) -> vecteur:
```

qui prend deux points en paramètre et renvoie le vecteur $\overrightarrow{AB}$.

Q 2. Écrire une fonction d'entête

```
def ps(v1:vecteur, v2:vecteur) -> float:
```

qui prend en paramètres deux vecteurs $\vec{v}_1$ et $\vec{v}_2$ et qui renvoie la valeur de leur produit scalaire $\vec{v}_1 \cdot \vec{v}_2$.

Q 3. Écrire une fonction d'entête

```
def norme(v:vecteur) -> float:
```

qui prend en paramètre un vecteur $\vec{v}$ et qui renvoie $\|\vec{v}\|$, la valeur de sa norme euclidienne.

Q 4. Écrire une fonction d'entête

```
def unitaire(v:vecteur) -> vecteur:
```

qui prend en paramètre un vecteur $\vec{v}$ non nul et qui renvoie $\dfrac{1}{\|\vec{v}\|}\vec{v}$, le vecteur unitaire correspondant.

On utilise dans ce sujet le modèle du rayon lumineux de l'optique géométrique. Un rayon lumineux issu du point S est une demi-droite d'origine S. Ce rayon est représenté en Python par le couple $(S, \vec{u})$, où S est le point de départ du rayon et $\vec{u}$ le vecteur unitaire donnant la direction de propagation du rayon lumineux. Tout point M du rayon est tel que $\overrightarrow{SM} = t\vec{u}$, avec $t \in \mathbb{R}^+$. Un tel couple est désignée dans la suite par le type `rayon`.

Ainsi, en définissant `O = np.array([0., 0., 0.]` et `u = np.array([0, 0.6, 0.8]`, le couple `(O, u)` représente le rayon issu de O dans la direction $3\vec{u}_y + 4\vec{u}_z$.

Q 5. Que font les fonctions `pt`, `dir` et `ra` ci-dessous ?

```
1  def pt(r:rayon, t:float) -> point:
2      assert t >= 0
3      (S, u) = r
4      return S + t * u
```


CC BY-NC-SA

```
5   def dir(A:point, B:point) -> vecteur:
6       return unitaire(vec(A, B))

7   def ra(A:point, B:point) -> rayon:
8       return A, dir(A, B)
```

Comme toutes les fonctions définies dans ce sujet, les fonctions `pt`, `dir` et `ra` peuvent être utilisées dans la suite.

Une sphère de centre C et de rayon $r > 0$ est l'ensemble des points de l'espace situés à la distance r du point C. Ici, elle est représentée en Python par le couple (C, r). Un tel couple est désigné par le type `sphère`.

Q 6. Écrire une fonction d'entête

```
def sp(A:point, B:point) -> sphère:
```

qui renvoie la sphère de centre A passant par B.

Q 7. Montrer qu'une droite passant par le point A de vecteur directeur $\vec{u}$ et une sphère (C, r) sont sécantes si et seulement si l'équation d'inconnue t

$$t^2 + 2t(\vec{u} \cdot \overrightarrow{CA}) + \|\overrightarrow{CA}\|^2 - r^2 = 0 \tag{1}$$

possède deux solutions réelles, éventuellement confondues.

Q 8. Écrire une fonction d'entête

```
def intersection(r:rayon, s:sphère) -> (point, float) or None:
```

qui renvoie le premier point de la sphère `s` frappé par le rayon lumineux `r` et la distance entre ce point et l'origine du rayon. La fonction renvoie `None` si le rayon ne coupe pas la sphère. On suppose que l'origine du rayon n'est pas située à l'intérieur de la sphère.

II Optique

Les couleurs seront représentées par des tableaux de 3 nombres flottants, associés dans la suite au type `couleur`. Les trois composantes codent respectivement le niveau de rouge, vert et bleu de la couleur considérée (composantes RVB). Chaque composante est comprise entre 0 et 1. Plus la valeur est élevée, plus la composante contribue fortement à la couleur. Ainsi, (1, 1, 1) correspond au blanc, (0, 0, 0) au noir, (0.5, 0.5, 0.5) désigne un gris moyen et (0.6, 1, 0.6) un vert pastel.

Pour toute la suite, on a défini les variables globales

```
noir  = np.array([0., 0., 0.])
blanc = np.array([1., 1., 1.])
```

II.A – Visibilité

Q 9. Une source lumineuse ponctuelle S ne peut être vue d'un point P d'une sphère (C, r) que si la source est au-dessus de l'horizon de P, défini ici comme le plan tangent à la sphère en P. Donner une condition géométrique pour que la source soit au-dessus de l'horizon.

Q 10. Écrire une fonction booléenne, d'entête

```
def au_dessus(s:sphère, P:point, src:point) -> bool:
```

qui détermine si la source située en `src` est au-dessus de l'horizon du point `P` de la sphère `s`.

Q 11. On considère une scène contenant plusieurs sphères et une source lumineuse. Pour que la source soit visible d'un point P d'une sphère particulière, il faut que cette source soit au-dessus de l'horizon et qu'aucune autre sphère ne la cache. Écrire une fonction booléenne d'entête

```
def visible(obj:[sphère], j:int, P:point, src:point) -> bool:
```

où le paramètre `obj` est une liste contenant les sphères de la scène et `src` l'emplacement de la source lumineuse. Cette fonction détermine si la source est visible du point `P`, appartenant à la sphère `obj[j]`.

II.B – Diffusion

On considère un point P, à la surface d'une sphère, éclairé sous l'incidence θ par une source lumineuse S ponctuelle de couleur $C_s = (R_s, V_s, B_s)$. On note $\overrightarrow{N}$ le vecteur unitaire normal à la sphère en P, dirigé vers l'extérieur de la sphère, et $\vec{u}$ le vecteur unitaire du rayon lumineux qui éclaire P en provenance de la source. On considère que le point P diffuse la lumière de la source de manière isotrope dans tout le demi-espace délimité par le plan tangent à la sphère en P qui contient la source (figure 2). Autrement dit, le point P diffuse la lumière de la source dans toutes les directions $\overrightarrow{w}$ telles que $\overrightarrow{w} \cdot \overrightarrow{N} \geqslant 0$ (les vecteurs $\vec{u}$, $\overrightarrow{w}$ et $\overrightarrow{N}$ ne sont pas forcément coplanaires) et la lumière diffusée ne dépend pas de la direction d'observation $\overrightarrow{w}$, en particulier elle ne dépend pas de θ'.

BY-NC-SA

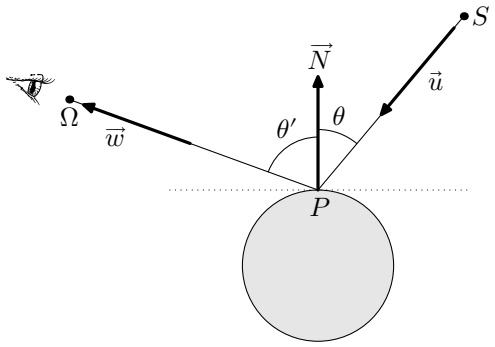

Figure 2 Parcours de la lumière de la source à l'œil

La faculté d'un objet à diffuser la lumière est modélisée par ses coefficients de diffusion k_{dr}, k_{dv} et k_{db} qui mesurent son aptitude à réémettre les composantes rouge, verte et bleue. Chaque coefficient est un nombre à virgule flottante compris entre 0 et 1. On représente les coefficients de diffusion d'un objet par un triplet $k_d = (k_{dr}, k_{dv}, k_{db})$ de type `couleur`. La couleur $C_d = (R_d, V_d, B_d)$ de la lumière diffusée par le point P éclairé par la source S suit alors la loi de Lambert :

$$C_d = (k_d \odot C_s)\cos\theta \quad \text{soit} \quad (R_d, V_d, B_d) = (k_{dr}R_s\cos\theta,\ k_{dv}V_s\cos\theta,\ k_{db}B_s\cos\theta). \tag{2}$$

Q 12. Écrire une fonction d'entête

```
def couleur_diffusée(r:rayon, Cs:couleur, N:vecteur, kd:couleur) -> couleur:
```

qui renvoie la couleur de la lumière diffusée par le point P éclairé par un rayon lumineux `r` en provenance d'une source ponctuelle de couleur `Cs`. Les paramètres `N` et `kd` représentent respectivement le vecteur unitaire normal à l'objet en P et les coefficients de diffusion de l'objet (figure 2). La source S est supposée visible de P.

II.C – Réflexion

Si la surface de l'objet est réfléchissante, au phénomène de diffusion s'ajoute le phénomène de réflexion. Lorsqu'un rayon lumineux $(S, \vec{u})$ issu d'un point source S atteint un point P de la surface, il donne naissance au rayon réfléchi $(P, \vec{w})$. Les lois de la réflexion de Descartes stipulent que, avec les notations de la figure 2,

— $\vec{u}$, $\vec{w}$ et $\vec{N}$ sont coplanaires ;

— $(\vec{u} + \vec{w}) \cdot \vec{N} = 0$, ce qui correspond à $\theta' = \theta$.

Q 13. Écrire une fonction d'entête

```
def rayon_réfléchi(s:sphère, P:point, src:point ) -> rayon:
```

qui renvoie le rayon réfléchi par le point `P` de la sphère `s` en provenance de la source S placée en `src`. Le résultat est le couple $(P, \vec{w})$ représentant le rayon émergent. La source S est supposée visible de P.

III Enregistrement des scènes

Les différentes scènes à représenter sont enregistrées dans une base de données relationnelle. La figure 3 donne sa structure physique.

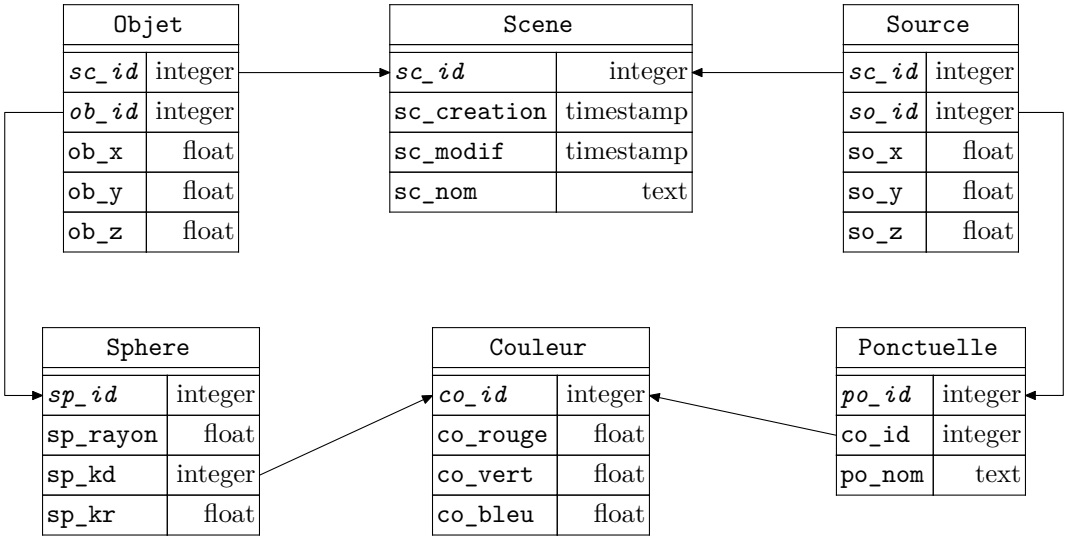

Figure 3 Structure physique de la base de données des scènes

BY-NC-SA

Cette base comporte les six tables listées ci-dessous avec la description de leurs colonnes :

— la table `Scene` répertorie les scènes
  - `sc_id` identifiant (entier arbitraire) de la scène (clé primaire)
  - `sc_creation` date de création de la scène
  - `sc_modif` date de dernière modification de la scène
  - `sc_nom` nom de la scène

— la table `Couleur` définit les couleurs utilisées
  - `co_id` identifiant (entier arbitraire) de la couleur (clé primaire)
  - `co_rouge` valeur de la composante rouge (dans l'intervalle $[0, 1]$)
  - `co_vert` valeur de la composante verte (dans l'intervalle $[0, 1]$)
  - `co_bleu` valeur de la composante bleue (dans l'intervalle $[0, 1]$)

— la table `Sphere` liste les sphères utilisées pour construire les scènes
  - `sp_id` identifiant (entier arbitraire) de la sphère (clé primaire)
  - `sp_rayon` rayon de la sphère
  - `sp_kd` coefficients de diffusion de la sphère (référence dans la table `Couleur`)
  - `sp_kr` coefficient de réflexion de la sphère (cf. partie V)

— la table `Ponctuelle` répertorie les sources ponctuelles disponibles
  - `po_id` identifiant (entier arbitraire) de la source ponctuelle (clé primaire)
  - `co_id` couleur de la source (référence dans la table `Couleur`)
  - `po_nom` nom de la source

— la table `Objet` fait le lien entre les scènes et les objets qu'elles contiennent, ses deux premières colonnes constituent sa clé primaire
  - `sc_id` identifiant de la scène
  - `ob_id` identifiant de l'objet
  - `ob_x`, `ob_y`, `ob_z` coordonnées du centre de l'objet dans la scène considérée

— la table `Source` indique quelles sont les sources qui éclairent chaque scène, ses deux premières colonnes constituent sa clé primaire
  - `sc_id` identifiant de la scène
  - `so_id` identifiant de la source
  - `so_x`, `so_y`, `so_z` coordonnées de la source dans la scène considérée

Q 14. Écrire une requête SQL qui donne le nom des scènes créées au cours de l'année 2021.

Q 15. Écrire une requête SQL qui donne, pour chaque scène, son identifiant et le nombre de sources qui l'éclairent.

Q 16. Écrire une requête SQL qui liste l'identifiant, les coordonnées du centre et le rayon de toutes les sphères contenues dans la scène dont le nom est `woodbox`. On suppose qu'une seule scène possède ce nom.

Il est possible en SQL de définir des fonctions utilisateur qui s'utilisent comme les fonctions SQL pré-définies.

On dispose d'une fonction utilisateur booléenne de signature `OCCULTE(sc_id, objr_id, so_id, objo_id)` où `sc_id`, `objr_id`, `so_id` et `objo_id` sont les identifiants respectifs d'une scène, d'un objet de cette scène dit « récepteur », d'une source de cette scène et d'un autre objet de la scène dit « occultant ». La fonction renvoie `TRUE` si l'objet occultant projette son ombre sur l'objet récepteur lorsqu'il est éclairée par la source considérée.

Q 17. Écrire une requête SQL qui, pour la scène `woodbox`, renvoie tous les triplets `objr_id, so_id, objo_id` pour lesquels l'objet d'identifiant `objo_id` occulte la source d'identifiant `so_id` pour l'objet d'identifiant `objr_id`.

IV Lancer de rayons

Cette partie implante l'algorithme qui génère l'image 2D de la scène 3D à visualiser. Pour représenter une scène en Python, on utilise les quatre variables globales suivantes :

— `Objet` est une liste des n_o objets (sphères de type `sphère`) contenues dans la scène à représenter ;

— `KdObj` est une liste de n_o tableaux de type `couleur` représentant les coefficients de diffusion des sphères, `KdObj[i]` est associé à la sphère `Objet[i]` ;

— `Source` est une liste de n_s points (de type `point`) donnant l'emplacement des n_s sources lumineuses (ponctuelles) qui éclairent la scène à représenter ;

— `ColSrc` est une liste de n_s couleurs, `ColSrc[i]` représente la couleur de la source placée en `Source[i]`.

Pour des raisons d'efficacité, il est d'usage de construire le parcours de la lumière de l'œil vers les sources : on « lance » des rayons. Cela est possible grâce au principe du retour inverse de la lumière : un rayon parcourt le

BY-NC-SA

même chemin pour joindre 2 points A et B, qu'il aille de B vers A ou de A vers B. Dans la suite, les vecteurs unitaires caractérisant les rayons sont donc orientés dans le sens opposé au sens réel de propagation de la lumière.

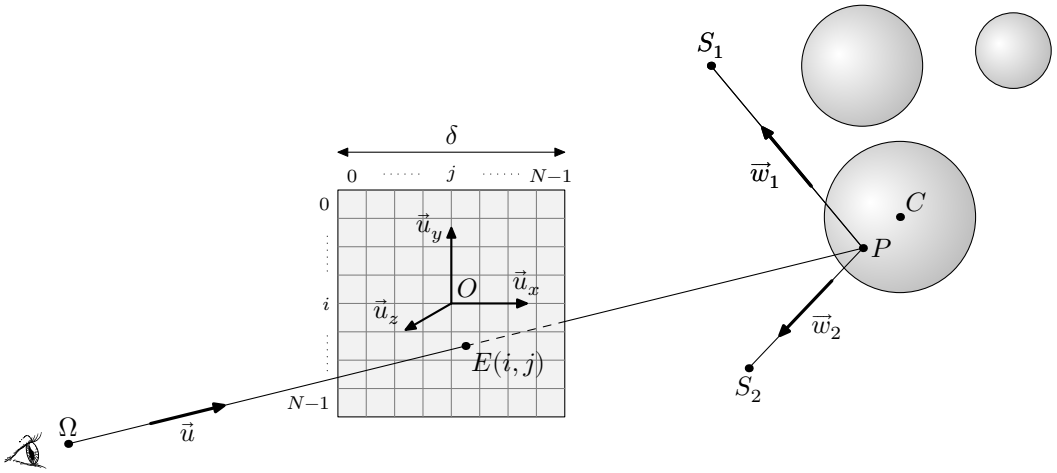

Figure 4 Une scène avec 3 sphères et 2 sources ponctuelles

Un écran translucide est placé dans le plan $(O, \vec{u}_x, \vec{u}_y)$. Le point O coïncide avec le centre de l'écran.
Il sépare les objets de la scène, placés dans le demi espace $z < 0$, de l'œil Ω placé de l'autre côté : $z_\Omega > 0$.

IV.A – Écran

L'écran est un carré de côté δ, divisé en $N \times N$ cases (N pair) qui représentent les pixels de l'image finale. Les variables globales `Delta` et `N` contiennent respectivement les valeurs de δ et N.

Q 18. Écrire une fonction d'entête

```
def grille(i:int, j:int) -> point:
```

qui renvoie les coordonnées cartésiennes du point E, centre de la case repérée par les indices (i, j) de la grille (figure 4).

Q 19. Écrire une fonction d'entête

```
def rayon_écran(omega:point, i:int, j:int) -> rayon:
```

qui renvoie le rayon issu du point `omega` et passant par $E(i, j)$.

IV.B – Couleur d'un pixel

Dans un premier temps, les objets de la scène à représenter sont supposés parfaitement mats, ils se contentent de diffuser la lumière des sources sans la réfléchir.

On considère un point P d'un objet de la scène atteint par un rayon issu de l'œil. On note E le point de l'écran coupé par ce rayon allant de l'œil au point P. La couleur $C_d = (R_d, V_d, B_d)$ diffusée par P est la somme des couleurs diffusées par ce point, dues à l'éclairement des sources visibles du point P.

$$C_d = \sum_{S_i \text{ visibles}} (k_d \odot C_{s_i}) \cos\theta_i. \tag{3}$$

On suppose que les couleurs ne saturent pas : toutes les composantes de C_d restent inférieures à 1. La couleur C_d est affectée au point E de l'écran.

Q 20. Écrire une fonction d'entête

```
def interception(r:rayon) -> (point, int) or None:
```

qui prend en paramètre un rayon `r` et qui renvoie le premier point matériel de la scène atteint par ce rayon ainsi que l'indice de la sphère concernée dans la liste `Objet`. Si le rayon n'intercepte aucune sphère, la fonction renvoie `None`.

Q 21. Écrire une fonction d'entête

```
def couleur_diffusion(P:point, j:int) -> couleur:
```

qui renvoie la couleur diffusée par le point `P` appartenant à la sphère `Objet[j]`. Si aucune source n'éclaire P, la fonction renvoie `noir`.

IV.C – Constitution de l'image

L'image de la scène que l'on souhaite obtenir est construite dans un tableau à 3 dimensions, de taille $N \times N \times 3$, associé au type `image`. L'affectation de la couleur `c` de type `couleur` au pixel (i, j) de l'image `im` s'écrira simplement `im[i, j] = c`.

BY-NC-SA

Q 22. Écrire une fonction d'entête

```
def lancer(omega:point, fond:couleur) -> image:
```

qui génère l'image associée à la scène. Si un rayon n'intercepte aucun objet, le pixel correspondant est de couleur `fond`.

IV.D – Complexités

Les complexités asymptotiques seront exprimées en fonction du nombre N de lignes de l'image, du nombre n_o d'objets et du nombre n_s de sources.

Q 23. Calculer la complexité temporelle de la fonction `lancer` dans le meilleur des cas en caractérisant la situation correspondante.

Q 24. Calculer la complexité temporelle de la fonction `lancer` dans le pire des cas en caractérisant la situation correspondante.

V Améliorations

V.A – Prise en compte de la réflexion

Désormais les sphères sont supposées au moins partiellement réfléchissantes. Un rayon issu de l'œil peut alors se réfléchir successivement sur plusieurs sphères.

Q 25. Écrire une fonction d'entête

```
def réflexions(r:rayon, rmax:int) -> [(point, int)]:
```

qui renvoie une liste de couples (P_k, i_k) correspondant aux points successivement rencontrés par le rayon `r` au fur et à mesure de ses réflexions. Dans chaque couple, P_k est le point où a lieu la $(k+1)^{\text{ème}}$ réflexion et i_k est l'indice de l'objet contenant P_k. Le résultat comporte au plus `rmax` éléments, les éventuelles réflexions ultérieures ne sont pas prises en compte.

Le pouvoir réfléchissant d'un objet est caractérisé par un coefficient de réflexion k_r propre à chaque objet, $0 \leqslant k_r \leqslant 1$. Les coefficients de réflexion des objets de la scène (de type `float`) sont stockés dans une liste globale `KrObj` à n_o éléments, telle que `KrObj[i]` est le coefficient de réflexion de l'objet `Objet[i]`.

En tenant compte des phénomènes de diffusion et de réflexion, la couleur $C_k = (R_k, V_k, B_k)$ du point P_k vaut

$$C_k = C_{dk} + k_r C_{k+1} \tag{4}$$

où C_{dk} désigne la couleur diffusée par le point P_k (cf. IV.B) et C_k vaut `noir` si k est supérieur au nombre de réflexions considérées.

Q 26. Écrire une fonction d'entête

```
def couleur_perçue(r:rayon, rmax:int, fond:couleur) -> couleur:
```

qui renvoie la couleur du premier point de la scène rencontré par le rayon `r` en tenant compte d'au maximum `rmax` réflexions de ce rayon. Si le rayon ne rencontre aucun objet, la fonction renvoie la couleur `fond`.

Q 27. Écrire une fonction d'entête

```
def lancer_complet(omega:point, fond:couleur, rmax:int) -> image:
```

qui construit l'image de la scène en tenant compte des diffusions et des réflexions.

Q 28. Exprimer la nouvelle complexité dans le pire cas.

V.B – Une optimisation

On construit, à partir de la base de données, les deux listes globales :

— `IdObj` telle que `IdObj[i]` soit l'identifiant dans la base de données (`ob_id`) de la sphère `Objet[i]` ;

— `IdSrc` telle que `IdSrc[i]` soit l'identifiant dans la base de données (`so_id`) de la source `Source[i]`.

Q 29. Écrire la fonction d'entête

```
def table_risque(risque:[[int, int, int]]) -> [[[int]]]:
```

qui prend en paramètre une liste de triplets correspondant au résultat de la requête SQL de la question 17 et construit une liste de listes de listes d'entiers telle que, si `res` est le résultat de la fonction, `res[i][j]` donne la liste (éventuellement vide) des indices des objets susceptibles de masquer la source `Source[j]` pour un point de l'objet `Objet[i]`.

Q 30. Le résultat de la fonction `table_risque` est conservé dans la variable globale `TableRisque`. Écrire la fonction `visible_opt` d'entête

```
def visible_opt(j:int, k:int, P:point) -> bool:
```

qui prend en paramètres l'indice `j` d'une source, l'indice `k` d'une sphère et un point `P` de cette sphère et qui, comme la fonction `visible` de la question 11, détermine si la source `Source[j]` est visible à partir du point `P`.

CC BY-NC-SA

Opérations et fonctions disponibles en Python et en SQL

Constantes

— `math.inf`, `np.inf` correspondent à $+\infty$, n'importe quel nombre est strictement inférieur à cette valeur.

Fonctions Python diverses

— `range(n)` itérateur sur les `n` premiers entiers ($[\![0, n-1]\!]$).
`list(range(5))` → `[0, 1, 2, 3, 4]`.

— `range(d, f, p)` où `d`, `f` et `p` sont des entiers, itérateur sur les entiers $(r_i = d + ip \mid r_i < f)_{i\in\mathbb{N}}$ si $p > 0$ et $(r_i = d + ip \mid r_i > f)_{i\in\mathbb{N}}$ si $p < 0$. Le paramètre `p` est optionnel avec une valeur par défaut de 1.
`list(range(1, 5))` → `[1, 2, 3, 4]` ; `list(range(20, 10, -2))` → `[20, 18, 16, 14, 12]`.

— `math.sqrt(x)` calcule la racine carrée du nombre `x`.

Opérations sur les listes

— `len(L)` donne le nombre d'éléments de la liste `L`.

— `L1 + L2` construit une liste constituée de la concaténation des listes `L1` et `L2`.

— `e in L` et `e not in L` déterminent si l'objet `e` figure dans la liste `L`. Ces opérations ont une complexité temporelle en $O($`len(L)`$)$.

— `L.append(e)` ajoute l'élément `e` à la fin de la liste `L`.

— `L.index(e)` renvoie le plus petit entier `i` tel que `L[i] == e`. Lève l'exception `ValueError` si l'élément `e` n'apparait pas dans la liste. Cette opération a une complexité temporelle en $O($`len(L)`$)$.

— `L.sort()` trie en place la liste `L` (qui est donc modifiée) en réordonnant ses éléments dans l'ordre croissant.

Opérations sur les tableaux (`np.ndarray`)

— `np.array(s, dtype)` crée un nouveau tableau contenant les éléments de la séquence `s`. La taille de ce tableau est déduite du contenu de `s`. Le paramètre optionnel `dtype` précise le type des éléments du tableau créé.

— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un tableau à une dimension de `n` éléments et un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype`. Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float`.

— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.

— `np.sum(a)` ou `a.sum()` renvoie la somme des éléments du tableau `a`.

— `np.inner(a, b)` calcule la somme des produits terme à terme dans le cas où `a` et `b` sont deux tableaux à une dimension de même taille.

— `np.all(a)` vaut `True` si tous les éléments du tableau `a` ont une valeur logique « vrai ».

— `np.any(a)` vaut `True` si au moins un des éléments du tableau `a` a une valeur logique « vrai ».

SQL

— `SELECT … FROM t1 JOIN t2 ON …` effectue une requête sur le résultat du produit cartésien entre les tables `t1` et `t2` restreint par la condition indiquée. Par exemple `t1.a = t2.b` permet de limiter le résultat aux lignes pour lesquelles la colonne `a` de la table `t1` est égale à la colonne `b` de la table `t2`.

— `SELECT … FROM t AS t1 JOIN t AS t2 ON …` effectue une requête sur le résultat du produit cartésien de la table `t` avec elle-même restreint par la condition indiquée. Le premier exemplaire de la table `t` est désigné par `t1` et le second par `t2`.

— La fonction `EXTRACT(`*part* `FROM t)` extrait un élément de `t`, expression de type `date`, `time`, `timestamp` (jour et heure) ou `interval` (durée). *part* peut prendre les valeurs `year`, `month`, `day` (jour dans le mois), `doy` (jour dans l'année), `dow` (jour de la semaine), `hour`, etc.

— Les fonctions d'agrégation `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(e)`, `COUNT(*)` calculent respectivement la somme, la moyenne arithmétique, le maximum, le minium, le nombre de valeurs non nulles de l'expression `e` et le nombre de lignes pour chaque groupe de lignes défini par la cause `GROUP BY`. Si la requête ne comporte pas de clause `GROUP BY` le calcul est effectué pour l'ensemble des lignes sélectionnées par la requête.

• • • FIN • • •

CC BY-NC-SA

Concours Centrale-Supélec 2020 filière PC

Informatique

Présentation du sujet

Le sujet est construit autour d'un des thèmes du programmes de seconde année, le traitement des images. Il s'intéresse à la mise en œuvre de méthodes numériques visant à concevoir des photomosaïques, images composées à la manière d'une mosaïque d'une multitude de petites images appelées vignettes. Le sujet comporte 32 questions réparties sur 4 parties et fait largement appel aux connaissances algorithmiques et pratiques du programme de première année :

— la première partie traite du codage des images en termes de pixels et de codage RGB pour se terminer par l'écriture d'une fonction de conversion d'une image en niveaux de gris ;

— la deuxième partie étudie plusieurs solutions algorithmiques de redimensionnement d'images de complexités temporelles différentes. La partie se termine par une synthèse discutant des usages respectifs de ces solutions ;

— la troisième partie aborde le thème des bases des données par l'écriture de requêtes sélectionnant une image source et des vignettes ;

— la quatrième partie aboutit à la construction d'une photomosaïque. Les deux dernières questions laissent une part importante à l'initiative des candidats.

Outre la maitrise des connaissances informatiques du programme, l'écriture syntaxiquement correcte de codes et l'analyse de leurs performances, le sujet évalue l'aptitude des candidats à porter un regard critique sur des propositions de codes. Ce sujet a très largement permis au jury d'évaluer la qualité et le niveau de compétences de chaque candidat.

Analyse globale des résultats

Au regard de la longueur et de la difficulté de l'épreuve, le jury est satisfait du niveau général des copies. Les connaissances informatiques semblent globalement acquises, les langages Python et SQL convenablement maitrisés. Quelques rares candidats ont visiblement négligé la formation, tentant de répondre aux questions ne relevant pas immédiatement du domaine de l'informatique. Ces copies conduisent à des notes généralement très faibles.

La moitié des candidats de la filière PC traite pratiquement 65 % des questions. Un très faible pourcentage ne traite que moins de 20 % des questions.

De nombreux candidats ont fourni des copies d'excellente qualité. Le jury regrette le niveau parfois très bas d'autres copies. Il serait souhaitable que les candidats mesurent l'importance de la formation initiale en informatique pour la suite de leurs études mais également pour leurs cultures d'ingénieur et de citoyen.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Les compétences en matière de programmation élémentaire semblent acquises chez le plus grand nombre de candidats. Néanmoins, le jury souhaite attirer l'attention des futurs candidats sur les points suivants.

— La notion de *type* est essentielle en informatique. Il convient d'en tenir compte lors de la manipulation d'objets informatiques. En particulier, les candidats doivent s'interroger sur la pertinence et les limites de certaines opérations effectuées sur ou entre objets de même type.

— La *complexité* est souvent estimée au regard du nombre d'opérations effectuées dans tout ou partie d'un code. Un minimum d'explications est attendu pour justifier le résultat qui doit, en outre, être exprimé

avec les notations strictes de l'énoncé. Ainsi, affirmer que la présence de deux boucles imbriquées induit une complexité quadratique, souvent notée $O(n^2)$ sans préciser la nature de n, est insuffisant. Ces questions attendent une argumentation fondée menant à l'écriture de complexités sous la forme, par exemple, $O(h \times w)$ puis $O(n)$ après avoir rappelé que $n = h \times w$ (cf. question 9).

— La lecture et l'*analyse de codes* comptent parmi les activités de tout futur ingénieur. Elles requièrent plus qu'un simple survol du code et plus encore qu'un commentaire de type *paraphrase*. Il convient d'abord de préciser le rôle et les objectifs d'une fonction ou d'un bout de code puis d'identifier des blocs structurels importants du code et d'en expliquer leur fonction.

— Les *requêtes SQL* doivent faire l'objet d'une attention particulière. Il s'agit de répondre exactement à la question, sans oublier d'attributs dans la réponse, sans oublier les jointures, etc. Si les candidats maitrisent l'écriture de requêtes très élémentaires, de nombreuses réponses sont souvent incomplètes, voire mal écrites, en raison d'une lecture incomplète ou erronée des questions.

— Les codes sont globalement syntaxiquement corrects et lisibles. Leur lecture révèle toutefois une écriture au fil de l'eau. Il serait souhaitable qu'avant même d'écrire une fonction, chaque candidat s'interroge sur l'*organisation* et la *structure logiques des codes* qu'il propose. À cela s'ajoute la présence de commentaires parfois inutiles dans le corps du code. Un commentaire n'a de sens que s'il apporte une information utile et pertinente pour comprendre le code. Les docstrings (documentation placée immédiatement après la définition d'une fonction) sont toujours utiles en pratique mais généralement pas attendues sur une copie dans le cadre d'une épreuve de concours en temps limité.

— Des points dits *transversaux* ont été attribués pour valoriser la clarté des explications, la qualité rédactionnelle, le respect de la syntaxe de Python, la lisibilité des codes et les commentaires pertinents. Le jury est particulièrement attentif à ces compétences transversales.

Signalons par ailleurs quelques erreurs de syntaxe générales :

— écriture *à l'envers* des affectations, `10 = a` ;

— mauvaise gestion des `range` en ajoutant 1 à la valeur finale pour parcourir toute la liste ;

— l'incrémentation avec `+=` devient parfois `±` ;

— le symbole `*` de la multiplication est souvent omis.

I Pixels et images

Q1. Cette première question a mené à des réponses de qualité variable. Près d'une fois sur deux, le décompte du nombre de couleurs est incorrect et l'application numérique simple est souvent omise.

Q2. De nombreuses réponses sont erronées en raison essentiellement d'un manque de rigueur. La réponse attendait d'une part un objet de type clairement défini, d'autre part une proposition qui respecte les contraintes liées au codage RGB.

Q3. Très peu de candidat ont traité convenablement cette question. La principale erreur est liée à l'absence de prise en compte du type des objets manipulés, conduisant inévitablement à des erreurs dans les calculs demandés. Beaucoup de copies tentent d'écrire les opérations sous forme binaire.

Q4. Cette question est globalement bien traitée. Une attention particulière doit, là encore, être portée sur le type du résultat renvoyé. Beaucoup d'erreurs sont liées à un manque de rigueur dans le suivi des consignes. L'énoncé demandait *la meilleure approximation entière* (qui n'est ni la partie entière, ni le quotient euclidien d'une division) d'une moyenne, retournée sous le type `np.uint8`.

Q5. Cette question est comprise par l'ensemble des candidats mais les réponses sont souvent imprécises ou incomplètes. Deux points structuraient la réponse : un premier point détaillait le contenu de `source.shape`, un second point précisait la signification et le contenu de `source[0,0]`. La rédaction est souvent trop vague. Parler de *largeur* et de *longueur* d'une image est ambigu.

Q6. Cette question de codage est bien réussie par les candidats. Certaines réponses utilisant `a.shape` oublient parfois qu'en raison de la nature même de `a`, cette instruction renvoie un triplet. Le type des éléments du nouveau tableau renvoyé par la fonction est parfois oublié. Enfin, l'énoncé demandant de retourner une image en niveaux de gris qui est un tableau à deux dimensions, il est incorrect de modifier le tableau `a` passé en argument.

II Redimensionnement d'images

Q7. Dans cette question globalement comprise par les candidats, les explications sont parfois confuses même si les résultats sont corrects. Le jury est attentif à la qualité rédactionnelle et aux explications fournies. Même s'il fait preuve de bienveillance, les réponses succinctes, de type *steno*, sont pénalisées.

Q8. Cette question est globalement bien traitée. Une erreur récurrente est observée dans les arguments calculés qui permettent de sélectionner les éléments du tableau `A`. Certaines réponses ont tendance à utiliser `W` et `H` sans les définir dans la fonction. Des confusions sont également faites dans l'utilisation des dimensions `w` et `h`, parfois interverties.

Q9. Bien qu'à priori relativement simple, cette question a révélé la difficulté de nombreux candidats à argumenter leurs calculs de complexité. Une telle question attend une réponse détaillée : opérations prises en compte, décomptes du nombre de ces opérations, expressions du résultat final en respectant les notations strictes de l'énoncé. La seule réponse à la question ne suffit pas à obtenir tous les points.

Q10. Trop souvent l'explication se résume à une simple paraphrase des lignes du code, sans montrer une compréhension de celles-ci. Parler des valeurs « autour de `A[i,j]` » est bien trop vague. L'exercice de lecture et d'analyse d'un code attend bien évidemment plus qu'une lecture ligne à ligne. Comme le signale le début de ce rapport, le rôle du code analysé doit être précisé. S'agissant dans le cas présent d'une fonction, étant données des informations en entrée, le résultat renvoyé et son type doivent être indiqués. Ensuite, il convient d'identifier les blocs structurels importants et les étapes clés des calculs effectués dans le corps de la fonction. Ainsi, par son argumentation, le candidat montre sa capacité à prendre du recul par rapport à la question.

Q11. Comme pour la question 9, cette question a révélé les difficultés d'une présentation claire et rigoureuse des nombres d'opérations menant à l'établissement d'une complexité temporelle. Cette question est peu réussie.

Q12. Cette question a été soit très bien réussie, soit pas réussie du tout ou non traitée. Elle s'appuie sur des connaissances de cours simples.

Q13. Cette question plus délicate nécessitait une réflexion préalable à l'écriture de la fonction. Malheureusement, ce travail de préparation, trop souvent clairement absent au vu des productions, a abouti à l'écriture de fonctions incomplètes ou fausses. Quelques trop rares copies ont proposé de bonnes solutions. Une lecture attentive de l'énoncé aurait également pu éviter certaines erreurs comme par exemple le dimensionnement incorrect `H*W` d'un tableau alors qu'il était attendu `(H+1)*(W+1)`.

Q14. Cette question a fait l'objet d'un traitement variable. Les réponses présentent les mêmes défauts que ceux énoncés pour la question 10.

Q15. Cette question amène les mêmes commentaires que ceux des questions 9 et 11.

Q16. Cette question amène les mêmes commentaires que ceux de la question 10.

Q17. Quand elle traitée, cette question est peu réussie. Elle nécessitait de prendre du recul par rapport à l'ensemble des questions de la deuxième partie.

Q18. Cette question marquant la fin de la deuxième partie, quelques candidats ont fourni des réponses partielles en discutant la qualité des images obtenues.

III Sélection des images de la banque

Q19. Cette question est globalement très bien traitée par l'ensemble des candidats.

Q20. Cette question est bien traitée par l'ensemble des candidats. De nombreuses copies utilisent `USING` proposé en annexe. Signalons malgré tout quelques erreurs liées à une mauvaise écriture de la jointure. Attention également aux erreurs de syntaxe comme l'utilisation de `==` dans la clause `WHERE`.

Q21. Cette question nécessitait l'écriture de deux jointures. À ce sujet, les réponses sont inégales. Un nombre non négligeable de candidats ne maitrise pas l'écriture de jointures multiples. Par ailleurs, certaines clauses `WHERE` de fin de requête sont maladroitement écrites, avec un `OR` parfois hasardeux. Plusieurs candidats ont pensé que `PH_auteur` était le prénom de l'auteur alors que le type `integer` est clairement mentionné dans la table `Photo`. Néanmoins, plus de la moitié des candidats apporte une réponse tout à fait satisfaisante à cette question.

Q22. Cette question nécessitait de jointer convenablement trois tables avec deux conditions entre les tables `Photo` et `Present` par exemple. Une erreur fréquente réside dans l'oubli d'une de ces deux conditions. L'extraction de l'année, expliquée dans l'annexe, est souvent fausse.

Q23. Cette question a été peu traitée. Une réponse pouvait être formulée en un `INTERSECT` avec un `EXCEPT` ou un `INTERSECT` avec un `COUNT`.

Q24. Cette question laissait une place importante à la prise d'initiative. Néanmoins, la réponse proposée devait respecter le cahier des charges exprimé par les points a. et b. de l'énoncé, ce qui a conduit à des réponses parfois incomplètes.

Q25. Suite naturelle de la question 24, cette question est convenablement traitée par moins de la moitié des candidats. Peu de copies proposent une réponse satisfaisante et rigoureuse.

IV Placement des vignettes

Q26. Le plus simple était d'utiliser convenablement la fonction `procheVoisin` définie à la question 8. Les arguments passés à cette fonction sont parfois incorrects. Le traitement global de cette question est donc très variable.

Q27. Cette question est peu réussie en raison essentiellement de la non prise en compte du dépassement de capacité. Les entiers utilisés n'étant pas signés, `abs(a-b)` est généralement différent de `abs(b-a)`. Une solution consistait à redéfinir les images `a` et `b` en tableaux de type `np.int64` ou bien à calculer des différences de la forme `int(a[i,j])-int(b[i,j])`.

Q28. Cette question a globalement été réussie.

Q29. Cette question abordée par beaucoup de candidats nécessitait l'écriture d'un code organisé, appelant des fonctions définies aux questions précédentes. Les réponses sont variables en raison d'un manque d'organisation préliminaire des idées qui aurait permis une écriture plus aisée du code.

Q30. Peu de candidats ont abordé cette question avec succès.

Q31. et **Q32**. Les deux dernières questions, relativement ouvertes, laissaient la part belle aux propositions des candidats. Si la question 31 a permis de lire quelques propositions pertinentes de stratégie, son implantation dans la question 32 n'a fait l'objet que de très rares réponses correctes et complètes.

Conclusion

Les résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences certaines en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrice autorisée

2020

Photomosaïque

Une *photomosaïque* (figure 1) est une image composée à la manière d'une mosaïque, où les fragments sont eux-mêmes des petites images, appelées *vignettes*. Elle est créée à partir d'une image appelée *image source*. Chaque vignette remplace une zone de même forme dans l'image source appelée *pavé*. Les vignettes sont fabriquées à partir d'une collection d'images appelée *banque* d'images.

L'intérêt est essentiellement artistique : vue de loin, une photomosaïque ressemble à l'image source ; en se rapprochant, on reconnait les vignettes.

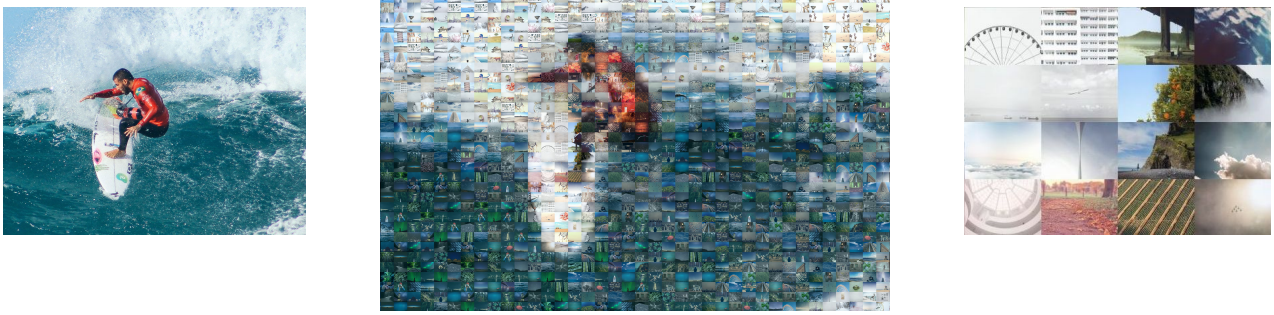

Figure 1 Photomosaïque d'un surfer composée de 1600 vignettes — de gauche à droite : l'image source[1], la photomosaïque et 16 vignettes[2] (détail du pied)

De nombreux paramètres régissent la construction d'une photomosaïque et la qualité du résultat :

— la structure du pavage utilisé (nombre, forme et arrangement des vignettes) ;

— le nombre et la diversité des images de la banque ;

— les algorithmes mis en œuvre pour :

- sélectionner les bonnes images dans la banque (partie III) ;
- redimensionner les images (partie II) ;
- placer les vignettes (partie IV).

Dans la suite du sujet, les photomosaïques sont construites sur des pavages rectangulaires réguliers, c'est-à-dire, constitués de vignettes rectangulaires, toutes de mêmes dimensions et juxtaposées bord à bord.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet, on suppose que les modules `math`, `numpy`, `matplotlib.pyplot` et `random` ont été rendus accessibles grâce à l'instruction

```
import math, numpy as np, matplotlib.pyplot as plt, random
```

Si les candidats font appel à des fonctions d'autres bibliothèques, ils doivent préciser les instructions d'importation correspondantes.

Dans tout le sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type `list`. Les termes « vecteur » et « tableau » désignent des objets numpy de type `np.ndarray`, respectivement à une dimension ou de dimension quelconque. Enfin le terme « séquence » représente une suite itérable et indiçable, indépendamment de son type Python, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers ».

[1] Photo par « Sincerely Media », issue de `https://unsplash.com`.

[2] Vignettes issues de la banque `https://picsum.photos/images`.

BY-NC-SA

Les entêtes des fonctions demandées sont annotés pour préciser les types des paramètres et du résultat. Ainsi,

```
def uneFonction(n:int, X:[float], c:str, u) -> np.ndarray:
```

signifie que la fonction `uneFonction` prend quatre paramètres `n`, `X`, `c` et `u`, où `n` est un entier, `X` une liste de nombres à virgule flottante, `c` une chaine de caractères et le type de `u` n'est pas précisé. Cette fonction renvoie un tableau numpy.

Il n'est pas demandé aux candidats d'annoter leurs fonctions, la rédaction pourra commencer par

```
def uneFonction(n, X, c, u):
    ...
```

De façon générale, une attention particulière sera portée à la lisibilité, la simplicité et la clarté du code proposé. L'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires seront appréciés.

Une liste de fonctions potentiellement utiles est fournie à la fin du sujet.

I Pixels et images

I.A – Pixels

Un pixel (contraction de l'anglais *picture element*) est un élément de couleur homogène utilisé pour représenter une image sous forme numérique. La teinte d'un pixel peut être représentée de plusieurs façons. Une méthode courante, basée sur la synthèse additive, consiste à la décomposer en trois composantes qui correspondent aux couleurs rouge, vert et bleu. On parle de représentation RGB (pour *red*, *green* et *blue*). Chacune des trois composantes donne l'intensité de la couleur correspondante dans la teinte finale, 0 indiquant l'absence de cette couleur. Ainsi, le triplet $(0, 0, 0)$ désigne un pixel noir.

Q 1. On suppose que chacune des trois composantes RGB d'un pixel est représentée par un nombre entier positif ou nul, codé sur 8 bits. Combien de couleurs différentes peut-on représenter avec un tel pixel ?

Dans la suite, on représente un pixel par un vecteur (tableau numpy à une dimension) d'entiers de type `np.uint8` (entier non signé codé sur 8 bits) à trois éléments, correspondant respectivement à chacune des composantes RGB du pixel ; on utilise dans toute la suite le type `pixel` pour désigner un tel vecteur.

Q 2. Donner une instruction permettant de créer un vecteur correspondant à un pixel blanc.

Il est rappelé qu'en Python, comme dans beaucoup de langages de programmation, les opérations d'addition, soustraction, multiplication, division entière, modulo et élévation à la puissance (opérateurs `+`, `-`, `*`, `//`, `%`, `**`) appliquées à deux opérandes de même type fournissent un résultat du type de leurs opérandes. Cela peut conduire à un dépassement de capacité et à une erreur de calcul car, les dépassements de capacité étant par défaut « silencieux », ils ne produisent pas d'erreur lors de l'exécution du programme.

L'opérateur division (`/`) entre deux entiers produit toujours un résultat sous forme de nombre à virgule flottante même si la division est exacte (`12 / 2` → `6.0`). Il en est de même pour toute fonction faisant implicitement appel à cet opérateur comme `np.mean`.

Q 3. On pose `a = np.uint8(280)` et `b = np.uint8(240)`. Que valent `a`, `b`, `a+b`, `a-b`, `a//b` et `a/b` ?

Les fonctions numpy qui effectuent de manière répétitive des opérations élémentaires, si elles ne garantissent pas l'absence de dépassement de capacité, prennent la précaution d'utiliser pour leurs calculs intermédiaires et leur résultat un type compatible avec le type de base de la plus grande capacité possible. Par exemple le résultat de `np.sum(np.array([100, 200], np.uint8))` est de type `np.uint64` (entier non signé codé sur 64 bits) et vaut bien 300.

Pour représenter une image en niveau de gris, on peut se contenter d'une valeur par pixel, représentant l'intensité du gris entre le noir et le blanc. Pour convertir une image en couleurs en niveaux de gris, on peut remplacer chaque pixel par un seul entier, dont la valeur correspond à la meilleure approximation entière de la moyenne des trois composantes RGB du pixel.

Q 4. Écrire une fonction d'entête

```
def gris(p:pixel) -> np.uint8:
```

qui calcule le niveau de gris correspondant au pixel `p`.

I.B – Images

Une image en niveaux de gris de taille $w \times h$ (w pixels de large, h pixels de haut) est associée à un tableau d'octets (type `np.uint8`) à deux dimensions, à h lignes et w colonnes. Chaque élément de ce tableau représente le niveau de gris du pixel correspondant. Ainsi le tableau à deux dimensions `img1`, défini par :

```
img1 = np.array([[ 85,   0, 127, 170,  85, 150],
                 [119, 102, 102, 123,  81, 170],
                 [255, 170,  90, 112,  63,  97],
                 [171, 212, 225, 186, 162, 171]], np.uint8)
```

définit une image de taille 6×4, représentée figure 2.

Dans toute la suite, on utilise le type `image` pour désigner un tableau d'octets à deux dimensions.

BY-NC-SA

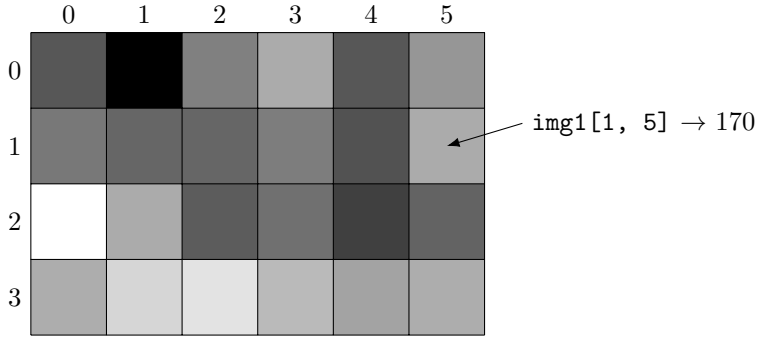

Figure 2 Visualisation de l'image `img1`

Pour les images en couleurs, on ajoute une dimension pour représenter les trois composantes d'un pixel. L'instruction `source = plt.imread("surfer.jpg")` charge dans un tableau numpy l'image en couleurs contenue dans le fichier `surfer.jpg`. Les expressions `source.shape` et `source[0,0]` valent alors respectivement :

`(3000, 4000, 3)` et `np.array([144, 191, 221], np.uint8)`.

Q 5. Interpréter ces valeurs.

Q 6. Écrire une fonction d'entête

```
def conversion(a:np.ndarray) -> image:
```

qui génère une image en niveaux de gris correspondant à la conversion de l'image en couleurs `a`.

II Redimensionnement d'images

On s'intéresse dans cette partie à plusieurs algorithmes de redimensionnement d'une image `A`, de taille $W \times H$ (W pixels de large par H pixels de haut, on note $N = HW$ son nombre total de pixels), en une image `a` de taille $w \times h$ (on pose $n = hw$). Nous nous intéresserons dans la suite uniquement à des images en niveau de gris.

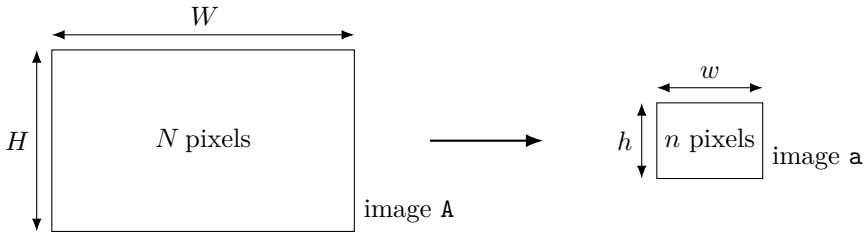

Figure 3 Redimensionnement d'image

II.A – Le contexte

À l'occasion du mariage d'Alice et de Bernard, leurs amis souhaitent réaliser plusieurs photomosaïques sur des thèmes variés. Ils ont pour cela accumulé un grand nombre de photos au ratio 4:3, ce qui signifie que le rapport W/H vaut *exactement* 4/3. Les photomosaïques mesureront chacune deux mètres de large et seront constituées de $40 \times 40 = 1600$ vignettes, toutes de même taille et au même ratio 4:3. Pour garder une bonne qualité d'impression, ils choisissent une résolution de 10 pixels par millimètre.

Q 7. Quelle taille de vignette ($w \times h$, en pixels) faut-il choisir ? Quelle sera alors la taille en pixels de la photomosaïque ?

II.B – Algorithme d'interpolation au plus proche voisin

Cette interpolation est définie par la formule $a(i,j) = A\left(\left\lfloor \frac{iH}{h} \right\rfloor, \left\lfloor \frac{jW}{w} \right\rfloor\right)$ où $\lfloor x \rfloor$ désigne la partie entière de x.

Q 8. Écrire une fonction d'entête

```
def procheVoisin(A:image, w:int, h:int) -> image:
```

qui renvoie une nouvelle image correspondant au redimensionnement de l'image `A` à la taille `w` × `h` en utilisant l'interpolation au plus proche voisin.

Q 9. Quelle est sa complexité temporelle asymptotique ?

II.C – Algorithme de réduction par moyenne locale

On suppose ici que les dimensions de l'image `a` divisent celles de l'image `A` : H/h et W/w sont entiers. Afin d'améliorer la qualité de la réduction, on propose la fonction `moyenneLocale`.

BY-NC-SA

```
1   def  moyenneLocale(A:image, w:int, h:int) -> image:
2       a = np.empty((h, w), np.uint8)
3       H, W = A.shape
4       ph, pw =  H // h, W // w
5       for I in range(0, H, ph):
6           for J in range(0, W, pw):
7               a[I // ph, J // pw] = round(np.mean(A[I:I+ph, J:J+pw]))
8       return a
```

Q 10. Expliquer en quelques lignes son principe de fonctionnement.

Q 11. Donner sa complexité temporelle asymptotique.

II.D – Optimisation de la réduction par moyenne locale

Afin d'accélérer le calcul de la moyenne locale, on précalcule pour chaque image sa table de sommation. La table de sommation d'une image `A` de N pixels, représentée par le tableau A à H lignes et W colonnes, est le tableau `S` à $H+1$ lignes et $W+1$ colonnes, défini par

$$\forall l \in [\![0, H]\!], \quad \forall c \in [\![0, W]\!], \qquad S(l,c) = \sum_{\substack{0 \leqslant i < l \\ 0 \leqslant j < c}} A(i,j),$$

la somme étant prise nulle quand elle ne comporte aucun terme.

Q 12. Le type `np.uint32` (entier non signé codé sur 32 bits) est-il suffisant pour stocker les éléments de S si l'image `A` comporte 50 millions de pixels ? Justifier.

Q 13. Écrire une fonction, de complexité temporelle asymptotique $O(N)$, d'entête

```
def tableSommation(A:image) -> np.ndarray:
```

qui calcule la table de sommation de l'image `A`.

On suppose à nouveau que les dimensions de l'image `A` divisent celles de l'image `a` : H/h et W/w sont entiers. On propose alors la fonction `réductionSommation1`, qui prend en paramètre l'image `A` et sa table de sommation `S` (`S = tableSommation(A)`), ainsi que les dimensions de l'image que l'on souhaite obtenir.

```
1   def réductionSommation1(A:image, S:np.ndarray, w:int, h:int) -> image:
2       a = np.empty((h, w), np.uint8)
3       H, W = A.shape
4       ph, pw =  H // h, W // w
5       nbp = ph * pw
6       for I in range(0, H, ph):
7           for J in range(0, W, pw):
8               X = (S[I+ph, J+pw] - S[I+ph, J]) - (S[I, J+pw] - S[I, J])
9               a[I // ph, J // pw] = round(X / nbp)
10      return a
```

Q 14. Expliquer en quelques lignes le principe de fonctionnement de `réductionSommation1`.

Q 15. Donner sa complexité temporelle asymptotique.

Q 16. Montrer que la fonction `réductionSommation2` dont le code est fourni ci-dessous donne le même résultat que `réductionSommation1`.

```
1   def réductionSommation2(A:image, S:np.ndarray, w:int, h:int) -> image:
2       H, W = A.shape
3       ph, pw =  H // h, W // w
4       sred = S[0:H+1:ph, 0:W+1:pw]
5       dc = sred[:, 1:] - sred[:, :-1]
6       dl = dc[1:, :] - dc[:-1, :]
7       d = dl / (ph * pw)
8       return np.uint8(d.round())
```

Q 17. Comparer les complexités asymptotiques en temps et en mémoire des deux versions de la fonction `réductionSommation`. Quel est l'avantage de la seconde version ?

II.E – Synthèse

Q 18. Discuter des cas d'usage respectifs de `procheVoisin`, `moyenneLocale` et `réductionSommation` pour redimensionner une image.

17

CC BY-NC-SA

III Sélection des images de la banque

Une première étape dans la conception d'une photomosaïque est le choix d'une image source et de vignettes. Cette partie est consacrée à la sélection d'images dans la banque.

Les images de la banque sont répertoriées dans une base de données dont le modèle physique est présenté figure 4, dans laquelle les clés primaires sont notées en italique.

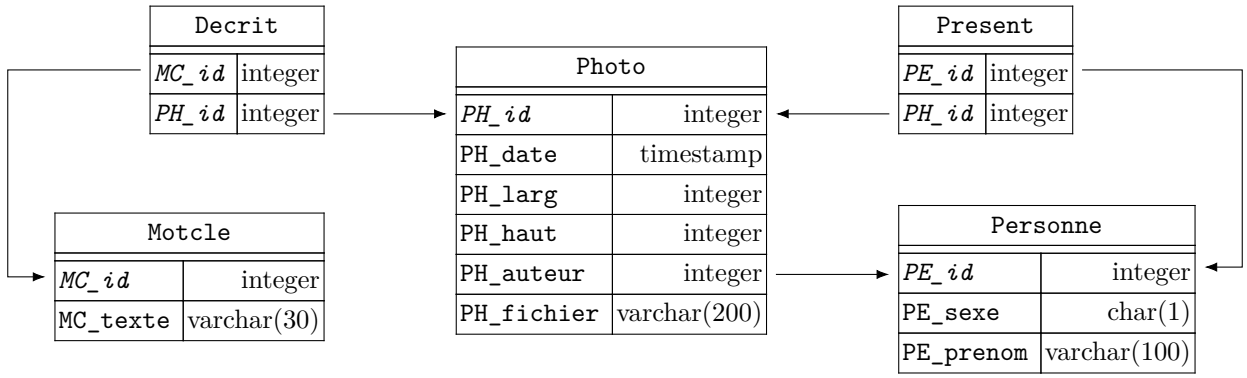

Figure 4 Structure physique de la base de données de photographies.

Cette base comporte les cinq tables listées ci-dessous avec la description de leurs colonnes :

— la table `Photo` répertorie les photographies
- `PH_id` identifiant (entier arbitraire) de la photographie (clé primaire)
- `PH_date` date et heure de la prise de vue
- `PH_larg`, `PH_haut` largeur et hauteur de la photographie en pixels
- `PH_auteur` identifiant de l'auteur de la photographie
- `PH_fichier` nom du fichier contenant la photographie

— la table `Personne` des modèles et des photographes
- `PE_id` identifiant (entier arbitraire) de la personne (clé primaire)
- `PE_sexe` sexe de la personne (`'M'` ou `'F'`)
- `PE_prenom` prénom de la personne

— la table `Motcle` des mots-clés utilisés pour décrire une photographie
- `MC_id` identifiant (entier arbitraire) du mot-clé (clé primaire)
- `MC_texte` le mot-clé lui-même

— la table `Decrit` fait le lien entre les photographies et les mots-clés qui les décrivent, ses deux colonnes constituent sa clé primaire
- `MC_id` identifiant du mot-clé (décrivant la photographie)
- `PH_id` identifiant de la photographie (décrite par le mot-clé)

— la table `Present` fait le lien entre les photographies et les personnes qui y figurent, ses deux colonnes constituent sa clé primaire
- `PE_id` identifiant de la personne (figurant sur la photographie)
- `PH_id` identifiant de la photographie (représentant la personne)

III.A – Quelques requêtes

Pour réaliser les photomosaïques du mariage d'Alice et Bernard, on dispose de plus de 20 000 photographies répertoriées dans une base de données dont le modèle est celui de la figure 4.

Q 19. Écrire une requête SQL donnant les identifiants de toutes les photographies au ratio 4:3, c'est-à-dire dont le rapport largeur sur hauteur vaut exactement 4/3.

Q 20. Écrire une requête qui compte le nombre de photos prises par « Alice » ou « Bernard ».

Q 21. Écrire une requête qui fournit l'identifiant et la date des photographies prises avant 2006 et associées au mot-clé « surf ».

Q 22. Écrire une requête qui donne le prénom de l'auteur et l'identifiant de tous les selfies, c'est-à-dire les photographies sur lesquelles l'auteur est présent.

Q 23. Écrire une requête qui sélectionne toutes les photographies sur lesquelles sont présents « Alice » et « Bernard », à l'exclusion de toute autre personne.

CC BY-NC-SA

III.B – Internationalisation des mots-clés

Afin de partager et d'enrichir la banque d'images, il a été décidé de faire évoluer la structure de la base de données afin de gérer les mots-clés dans différentes langues. Le cahier des charges de cette évolution stipule :

a. l'ensemble des photographies sélectionnées à l'aide de mots-clés ne doit pas dépendre de la langue utilisée pour exprimer ces mots-clés ; autrement dit, les photographies décrites par le mot-clé « montagne » exprimé en français doivent être les mêmes que celles sélectionnées par les mots-clés « mountain » si la langue choisie est l'anglais, « Berg » pour l'allemand, « montaña » pour l'espagnol, etc. ;

b. il doit être possible, pour cette nouvelle base de données, d'écrire une requête de recherche de photographies par mot-clef en spécifiant la langue utilisée pour exprimer le mot-clé de telle sorte que changer de langue se fasse en modifiant uniquement des constantes dans la clause `WHERE`.

Q 24. Proposer un nouveau modèle de base de données répondant à cette évolution du cahier des charges en ne détaillant que ce qui change (tables modifiées, nouvelles tables).

Q 25. Avec cette nouvelle base de données, écrire une requête qui permet de sélectionner les identifiants des photographies associées au mot-clé « mountain » exprimé en anglais.

IV Placement des vignettes

IV.A – Préparatifs

On envisage ici le cas où la photomosaïque est homothétique de l'image source et constituée de p vignettes de haut sur p vignettes de large. Le nombre total de vignettes est donc $r = p^2$.

Q 26. Écrire une fonction d'entête

```
def initMosaïque(source:image, w:int, h:int, p:int) -> image:
```

qui prend en paramètre l'image source, les dimensions w et h d'une vignette et le nombre p de vignettes par coté. Cette fonction renvoie une version redimensionnée de `source`, de même taille que la photomosaïque finale. On rappelle qu'il est possible d'utiliser les fonctions définies précédemment.

On appelle désormais *pavé* chaque zone de cette image source redimensionnée, de taille $w \times h$, qui doit être remplacé par une vignette. Afin de comparer les vignettes et les pavés, on définit la distance L_1 entre deux images a et b de même taille $w \times h$ par :

$$L_1(a,b) = \sum_{\substack{0 \leqslant i < h \\ 0 \leqslant j < w}} \big|a(i,j) - b(i,j)\big|.$$

Q 27. Écrire une fonction d'entête

```
def L1(a:image, b:image) -> int:
```

qui calcule la distance L_1 entre deux images de même taille, en prenant garde aux dépassements de capacité.

Q 28. Écrire une fonction d'entête

```
def choixVignette(pavé:image, vignettes:[image]) -> int:
```

qui prend en paramètre une image correspondant à un pavé et une liste de vignettes et qui renvoie l'indice `i` tel que `L1(pavé, vignettes[i])` est minimal (ou l'un d'entre eux si plusieurs vignettes conviennent). Cette fonction ne doit pas modifier la liste des vignettes.

IV.B – Méthode sans restriction du choix des vignettes

Q 29. Écrire, à l'aide de ce qui précède, une fonction d'entête

```
def construireMosaïque(source:image, vignettes:[image], p:int) -> image:
```

qui construit une photomosaïque homothétique de `source` comportant `p` vignettes par côté.

Q 30. Déterminer sa complexité temporelle asymptotique en fonction de la taille $n = hw$ des vignettes, du nombre r de vignettes dans la mosaïque et de la longueur q de la liste `vignettes`.

IV.C – Améliorations

Cette sous-partie demande de l'initiative de la part du candidat, qui peut être amené à définir de nouvelles variables, structures de données et fonctions. Il est demandé d'expliciter clairement la démarche utilisée, de préciser le rôle de chaque nouvelle fonction et variable introduite et de les illustrer, le cas échéant, par un schéma. Toute démarche pertinente, même non aboutie, sera valorisée. Le barème prend en compte le temps nécessaire à la résolution de cette sous-partie.

La méthode sans restriction proposée précédemment peut conduire à sélectionner répétitivement les mêmes vignettes et à mal les répartir. En particulier, une plage uniforme de l'image source conduit à l'accumulation de la même vignette dans cette zone de la photomosaïque.

CC BY-NC-SA

Q 31. Proposer une stratégie de construction de photomosaïque permettant de sélectionner un maximum de vignettes différentes et, au cas où une vignette serait réutilisée, d'éviter que les différentes apparitions de la même vignette se retrouvent trop proches.

Q 32. Implanter cette stratégie sous la forme d'une fonction `belleMosaïque`, version améliorée de la fonction `construireMosaïque`, dont on définira les éventuels paramètres supplémentaires.

Opérations et fonctions disponibles en Python et SQL

Fonctions Python diverses

— `range(n)` itérateur sur les `n` premiers entiers ($[\![0, n-1]\!]$).
`list(range(5))` → `[0, 1, 2, 3, 4]`.

— `range(d, f, p)` où `d`, `f` et `p` sont des entiers, itérateur sur les entiers $(r_i = d + ip \mid r_i < f)_{i\in\mathbb{N}}$ si $p > 0$ et $(r_i = d + ip \mid r_i > f)_{i\in\mathbb{N}}$ si $p < 0$. Le paramètre `p` est optionnel avec une valeur par défaut de 1.
`list(range(1, 5))` → `[1, 2, 3, 4]` ; `list(range(20, 10, -2))` → `[20, 18, 16, 14, 12]`.

— `s[d:f:p]` où `s` est une séquence et `d`, `f` et `p` sont des entiers, désigne la séquence des éléments de `s` dont les indices correspondent à `range(d, f, p)`. Si `s` est d'un type de base (liste ou tuple), `s[d:f:p]` effectue une copie, si `s` est un tableau numpy, `s[d:f:p]` est une vue sur les éléments de `s` et peut être utilisé pour modifier `s`.
`[0, 1, 2, 3, 4, 5][2:6:2]` → `[2, 4]` ; `(0, 1, 2, 3, 4, 5)[5:2:-2]` → `(5, 3)`.

— `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers).

— `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme.

— `random.choice(s)` renvoie un élément pris au hasard dans la séquence non vide `s`.

— `random.shuffle(L)` permute aléatoirement les éléments de la liste `L` (modifie `L`).

— `random.sample(s, n)` renvoie une liste constituée de `n` éléments distincts de la séquence `s` choisis aléatoirement, si `n` $\leqslant$ `len(s)` lève l'exception `ValueError`.

— `math.sqrt(x)` calcule la racine carrée du nombre `x`.

— `round(n)` arrondit le nombre `n` à l'entier le plus proche. Le résultat est de type `int` pour les types numériques de base. Pour les types de la bibliothèque numpy, le résultat a le même type que l'argument.

— `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`.

— `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à `x`.

Opérations sur les listes

— `len(L)` donne le nombre d'éléments de la liste `L`.

— `L1 + L2` construit une liste constituée de la concaténation des listes `L1` et `L2`.

— `n * L` construit une liste constituée de la liste `L` concaténée `n` fois avec elle-même.

— `e in L` et `e not in L` déterminent si l'objet `e` figure dans la liste `L`. Cette opération a une complexité temporelle en $O($`len`$(L))$.
`2 in [1, 2, 3]` → `True` ; `2 not in [1, 2, 3]` → `False`.

— `L.append(e)` ajoute l'élément `e` à la fin de la liste `L`.

— `L.pop(i)` : renvoie l'élément à l'indice `i` de la liste `L` et le supprime de la liste.

— `L.remove(e)` supprime de la liste `L` le premier élément qui a pour valeur `e`, s'il existe. Cette opération a une complexité temporelle en $O($`len`$(L))$.

— `L.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `L` (en décalant les éléments suivants) ; si `i >= len(L)`, `e` est ajouté en fin de liste.

— `L.sort()` trie en place la liste `L` (qui est donc modifiée) en réordonnant ses éléments dans l'ordre croissant.

Opérations sur les tableaux (`np.ndarray`)

— `np.array(s, dtype)` crée un nouveau tableau contenant les éléments de la séquence `s`. La taille de ce tableau est déduite du contenu de `s`. Le paramètre `dtype` précise le type des éléments du tableau créé.

— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un tableau à une dimension de `n` éléments et un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype`. Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float`.

— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.

— `np.full(n, v, dtype)`, `np.full((n, m), v, dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur `v`.

— `a.ndim` nombre de dimensions du tableau `a`.

CC BY-NC-SA

— `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions.

— `len(a)` taille du tableau `a` dans sa première dimension, équivalent à `a.shape[0]`.

— `a.size` nombre total d'éléments du tableau `a`.

— `a.dtype` type des éléments du tableau `a`.

— `a.flat` itérateur sur tous les éléments du tableau `a`.

— `np.ndenumerate(a)` itérateur sur tous les couples `(ind, v)` du tableau `a` où `ind` est un tuple de `a.ndim` entiers donnant les indices de l'élément `v`.

— `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O($`a.size`$)$.

— `a.sum()` ou `np.sum(a)` calcule la somme de tous les éléments du tableau `a` ; cette opération a une complexité temporelle en $O($`a.size`$)$.

— `a.sum(d)` ou `np.sum(a, d)` effectue la somme des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.
`a.sum(0)` → somme par ligne, `a.sum(1)` → somme par colonne, etc.

— `a.mean()` ou `np.mean(a)` renvoie la valeur moyenne de tous les éléments du tableau `a` ; le résultat est de type `np.float64`. Cette opération a une complexité temporelle en $O($`a.size`$)$.

— `a.mean(d)` ou `np.mean(a, d)` effectue la moyenne des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.
`a.mean(0)` → moyenne par ligne, `a.mean(1)` → moyenne par colonne, etc.

— `a.round()`, `np.around(a)` crée un nouveau tableau de même forme et type que `a` en arrondissant ses éléments à l'entier le plus proche.

SQL

— `T1 JOIN T2 USING (c1, c2, …)` joint les deux tables `T1` et `T2` sur les colonnes `c1`, `c2`… qui doivent exister dans les deux tables ; équivalent à `T1 JOIN T2 ON T1.c1 = T2.c1 AND T1.c2 = T2.c2 AND …`, sauf que les colonnes `c1`, `c2`… n'apparaissent qu'une fois dans le résultat.

— Les requêtes
- (SELECT ... FROM ... WHERE ...) INTERSECT (SELECT ... FROM ... WHERE ...)
- (SELECT ... FROM ... WHERE ...) UNION (SELECT ... FROM ... WHERE ...)
- (SELECT ... FROM ... WHERE ...) EXCEPT (SELECT ... FROM ... WHERE ...)

sélectionnent respectivement l'intersection, l'union et la différence des résultats des deux requêtes, qui doivent être compatibles : même nombre de colonnes et mêmes types.

— `EXTRACT(`*part* `FROM t)` extrait un élément de `t`, expression de type `date`, `time`, `timestamp` (jour et heure) ou `interval` (durée). *part* peut prendre les valeurs `year`, `month`, `day` (jour dans le mois), `doy` (jour dans l'année), `dow` (jour de la semaine), `hour`, etc.

— Les fonctions d'agrégation `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(e)`, `COUNT(*)` calculent respectivement la somme, la moyenne arithmétique, le maximum, le minium, le nombre de valeurs non nulles de l'expression `e` et le nombre de lignes pour chaque groupe de lignes défini par la cause `GROUP BY`. Si la requête ne comporte pas de clause `GROUP BY` le calcul est effectué pour l'ensemble des lignes sélectionnées par la requête.

• • • FIN • • •

CC BY-NC-SA

Concours Centrale-Supélec 2020 filière PC

Informatique

Présentation du sujet

Le sujet est construit autour d'un des thèmes du programmes de seconde année, le traitement des images. Il s'intéresse à la mise en œuvre de méthodes numériques visant à concevoir des photomosaïques, images composées à la manière d'une mosaïque d'une multitude de petites images appelées vignettes. Le sujet comporte 32 questions réparties sur 4 parties et fait largement appel aux connaissances algorithmiques et pratiques du programme de première année :

— la première partie traite du codage des images en termes de pixels et de codage RGB pour se terminer par l'écriture d'une fonction de conversion d'une image en niveaux de gris ;

— la deuxième partie étudie plusieurs solutions algorithmiques de redimensionnement d'images de complexités temporelles différentes. La partie se termine par une synthèse discutant des usages respectifs de ces solutions ;

— la troisième partie aborde le thème des bases des données par l'écriture de requêtes sélectionnant une image source et des vignettes ;

— la quatrième partie aboutit à la construction d'une photomosaïque. Les deux dernières questions laissent une part importante à l'initiative des candidats.

Outre la maitrise des connaissances informatiques du programme, l'écriture syntaxiquement correcte de codes et l'analyse de leurs performances, le sujet évalue l'aptitude des candidats à porter un regard critique sur des propositions de codes. Ce sujet a très largement permis au jury d'évaluer la qualité et le niveau de compétences de chaque candidat.

Analyse globale des résultats

Au regard de la longueur et de la difficulté de l'épreuve, le jury est satisfait du niveau général des copies. Les connaissances informatiques semblent globalement acquises, les langages Python et SQL convenablement maitrisés. Quelques rares candidats ont visiblement négligé la formation, tentant de répondre aux questions ne relevant pas immédiatement du domaine de l'informatique. Ces copies conduisent à des notes généralement très faibles.

La moitié des candidats de la filière PC traite pratiquement 65 % des questions. Un très faible pourcentage ne traite que moins de 20 % des questions.

De nombreux candidats ont fourni des copies d'excellente qualité. Le jury regrette le niveau parfois très bas d'autres copies. Il serait souhaitable que les candidats mesurent l'importance de la formation initiale en informatique pour la suite de leurs études mais également pour leurs cultures d'ingénieur et de citoyen.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Les compétences en matière de programmation élémentaire semblent acquises chez le plus grand nombre de candidats. Néanmoins, le jury souhaite attirer l'attention des futurs candidats sur les points suivants.

— La notion de *type* est essentielle en informatique. Il convient d'en tenir compte lors de la manipulation d'objets informatiques. En particulier, les candidats doivent s'interroger sur la pertinence et les limites de certaines opérations effectuées sur ou entre objets de même type.

— La *complexité* est souvent estimée au regard du nombre d'opérations effectuées dans tout ou partie d'un code. Un minimum d'explications est attendu pour justifier le résultat qui doit, en outre, être exprimé

avec les notations strictes de l'énoncé. Ainsi, affirmer que la présence de deux boucles imbriquées induit une complexité quadratique, souvent notée $O(n^2)$ sans préciser la nature de n, est insuffisant. Ces questions attendent une argumentation fondée menant à l'écriture de complexités sous la forme, par exemple, $O(h \times w)$ puis $O(n)$ après avoir rappelé que $n = h \times w$ (cf. question 9).

— La lecture et l'*analyse de codes* comptent parmi les activités de tout futur ingénieur. Elles requièrent plus qu'un simple survol du code et plus encore qu'un commentaire de type *paraphrase*. Il convient d'abord de préciser le rôle et les objectifs d'une fonction ou d'un bout de code puis d'identifier des blocs structurels importants du code et d'en expliquer leur fonction.

— Les *requêtes SQL* doivent faire l'objet d'une attention particulière. Il s'agit de répondre exactement à la question, sans oublier d'attributs dans la réponse, sans oublier les jointures, etc. Si les candidats maitrisent l'écriture de requêtes très élémentaires, de nombreuses réponses sont souvent incomplètes, voire mal écrites, en raison d'une lecture incomplète ou erronée des questions.

— Les codes sont globalement syntaxiquement corrects et lisibles. Leur lecture révèle toutefois une écriture au fil de l'eau. Il serait souhaitable qu'avant même d'écrire une fonction, chaque candidat s'interroge sur l'*organisation* et la *structure logiques des codes* qu'il propose. À cela s'ajoute la présence de commentaires parfois inutiles dans le corps du code. Un commentaire n'a de sens que s'il apporte une information utile et pertinente pour comprendre le code. Les docstrings (documentation placée immédiatement après la définition d'une fonction) sont toujours utiles en pratique mais généralement pas attendues sur une copie dans le cadre d'une épreuve de concours en temps limité.

— Des points dits *transversaux* ont été attribués pour valoriser la clarté des explications, la qualité rédactionnelle, le respect de la syntaxe de Python, la lisibilité des codes et les commentaires pertinents. Le jury est particulièrement attentif à ces compétences transversales.

Signalons par ailleurs quelques erreurs de syntaxe générales :

— écriture *à l'envers* des affectations, `10 = a` ;

— mauvaise gestion des `range` en ajoutant 1 à la valeur finale pour parcourir toute la liste ;

— l'incrémentation avec `+=` devient parfois `±` ;

— le symbole `*` de la multiplication est souvent omis.

I Pixels et images

Q1. Cette première question a mené à des réponses de qualité variable. Près d'une fois sur deux, le décompte du nombre de couleurs est incorrect et l'application numérique simple est souvent omise.

Q2. De nombreuses réponses sont erronées en raison essentiellement d'un manque de rigueur. La réponse attendait d'une part un objet de type clairement défini, d'autre part une proposition qui respecte les contraintes liées au codage RGB.

Q3. Très peu de candidat ont traité convenablement cette question. La principale erreur est liée à l'absence de prise en compte du type des objets manipulés, conduisant inévitablement à des erreurs dans les calculs demandés. Beaucoup de copies tentent d'écrire les opérations sous forme binaire.

Q4. Cette question est globalement bien traitée. Une attention particulière doit, là encore, être portée sur le type du résultat renvoyé. Beaucoup d'erreurs sont liées à un manque de rigueur dans le suivi des consignes. L'énoncé demandait *la meilleure approximation entière* (qui n'est ni la partie entière, ni le quotient euclidien d'une division) d'une moyenne, retournée sous le type `np.uint8`.

Q5. Cette question est comprise par l'ensemble des candidats mais les réponses sont souvent imprécises ou incomplètes. Deux points structuraient la réponse : un premier point détaillait le contenu de `source.shape`, un second point précisait la signification et le contenu de `source[0,0]`. La rédaction est souvent trop vague. Parler de *largeur* et de *longueur* d'une image est ambigu.

Q6. Cette question de codage est bien réussie par les candidats. Certaines réponses utilisant `a.shape` oublient parfois qu'en raison de la nature même de `a`, cette instruction renvoie un triplet. Le type des éléments du nouveau tableau renvoyé par la fonction est parfois oublié. Enfin, l'énoncé demandant de retourner une image en niveaux de gris qui est un tableau à deux dimensions, il est incorrect de modifier le tableau `a` passé en argument.

II Redimensionnement d'images

Q7. Dans cette question globalement comprise par les candidats, les explications sont parfois confuses même si les résultats sont corrects. Le jury est attentif à la qualité rédactionnelle et aux explications fournies. Même s'il fait preuve de bienveillance, les réponses succinctes, de type *steno*, sont pénalisées.

Q8. Cette question est globalement bien traitée. Une erreur récurrente est observée dans les arguments calculés qui permettent de sélectionner les éléments du tableau `A`. Certaines réponses ont tendance à utiliser `W` et `H` sans les définir dans la fonction. Des confusions sont également faites dans l'utilisation des dimensions `w` et `h`, parfois interverties.

Q9. Bien qu'à priori relativement simple, cette question a révélé la difficulté de nombreux candidats à argumenter leurs calculs de complexité. Une telle question attend une réponse détaillée : opérations prises en compte, décomptes du nombre de ces opérations, expressions du résultat final en respectant les notations strictes de l'énoncé. La seule réponse à la question ne suffit pas à obtenir tous les points.

Q10. Trop souvent l'explication se résume à une simple paraphrase des lignes du code, sans montrer une compréhension de celles-ci. Parler des valeurs « autour de `A[i,j]` » est bien trop vague. L'exercice de lecture et d'analyse d'un code attend bien évidemment plus qu'une lecture ligne à ligne. Comme le signale le début de ce rapport, le rôle du code analysé doit être précisé. S'agissant dans le cas présent d'une fonction, étant données des informations en entrée, le résultat renvoyé et son type doivent être indiqués. Ensuite, il convient d'identifier les blocs structurels importants et les étapes clés des calculs effectués dans le corps de la fonction. Ainsi, par son argumentation, le candidat montre sa capacité à prendre du recul par rapport à la question.

Q11. Comme pour la question 9, cette question a révélé les difficultés d'une présentation claire et rigoureuse des nombres d'opérations menant à l'établissement d'une complexité temporelle. Cette question est peu réussie.

Q12. Cette question a été soit très bien réussie, soit pas réussie du tout ou non traitée. Elle s'appuie sur des connaissances de cours simples.

Q13. Cette question plus délicate nécessitait une réflexion préalable à l'écriture de la fonction. Malheureusement, ce travail de préparation, trop souvent clairement absent au vu des productions, a abouti à l'écriture de fonctions incomplètes ou fausses. Quelques trop rares copies ont proposé de bonnes solutions. Une lecture attentive de l'énoncé aurait également pu éviter certaines erreurs comme par exemple le dimensionnement incorrect `H*W` d'un tableau alors qu'il était attendu `(H+1)*(W+1)`.

Q14. Cette question a fait l'objet d'un traitement variable. Les réponses présentent les mêmes défauts que ceux énoncés pour la question 10.

Q15. Cette question amène les mêmes commentaires que ceux des questions 9 et 11.

Q16. Cette question amène les mêmes commentaires que ceux de la question 10.

Q17. Quand elle traitée, cette question est peu réussie. Elle nécessitait de prendre du recul par rapport à l'ensemble des questions de la deuxième partie.

Q18. Cette question marquant la fin de la deuxième partie, quelques candidats ont fourni des réponses partielles en discutant la qualité des images obtenues.

III Sélection des images de la banque

Q19. Cette question est globalement très bien traitée par l'ensemble des candidats.

Q20. Cette question est bien traitée par l'ensemble des candidats. De nombreuses copies utilisent `USING` proposé en annexe. Signalons malgré tout quelques erreurs liées à une mauvaise écriture de la jointure. Attention également aux erreurs de syntaxe comme l'utilisation de `==` dans la clause `WHERE`.

Q21. Cette question nécessitait l'écriture de deux jointures. À ce sujet, les réponses sont inégales. Un nombre non négligeable de candidats ne maitrise pas l'écriture de jointures multiples. Par ailleurs, certaines clauses `WHERE` de fin de requête sont maladroitement écrites, avec un `OR` parfois hasardeux. Plusieurs candidats ont pensé que `PH_auteur` était le prénom de l'auteur alors que le type `integer` est clairement mentionné dans la table `Photo`. Néanmoins, plus de la moitié des candidats apporte une réponse tout à fait satisfaisante à cette question.

Q22. Cette question nécessitait de jointer convenablement trois tables avec deux conditions entre les tables `Photo` et `Present` par exemple. Une erreur fréquente réside dans l'oubli d'une de ces deux conditions. L'extraction de l'année, expliquée dans l'annexe, est souvent fausse.

Q23. Cette question a été peu traitée. Une réponse pouvait être formulée en un `INTERSECT` avec un `EXCEPT` ou un `INTERSECT` avec un `COUNT`.

Q24. Cette question laissait une place importante à la prise d'initiative. Néanmoins, la réponse proposée devait respecter le cahier des charges exprimé par les points a. et b. de l'énoncé, ce qui a conduit à des réponses parfois incomplètes.

Q25. Suite naturelle de la question 24, cette question est convenablement traitée par moins de la moitié des candidats. Peu de copies proposent une réponse satisfaisante et rigoureuse.

IV Placement des vignettes

Q26. Le plus simple était d'utiliser convenablement la fonction `procheVoisin` définie à la question 8. Les arguments passés à cette fonction sont parfois incorrects. Le traitement global de cette question est donc très variable.

Q27. Cette question est peu réussie en raison essentiellement de la non prise en compte du dépassement de capacité. Les entiers utilisés n'étant pas signés, `abs(a-b)` est généralement différent de `abs(b-a)`. Une solution consistait à redéfinir les images `a` et `b` en tableaux de type `np.int64` ou bien à calculer des différences de la forme `int(a[i,j])-int(b[i,j])`.

Q28. Cette question a globalement été réussie.

Q29. Cette question abordée par beaucoup de candidats nécessitait l'écriture d'un code organisé, appelant des fonctions définies aux questions précédentes. Les réponses sont variables en raison d'un manque d'organisation préliminaire des idées qui aurait permis une écriture plus aisée du code.

Q30. Peu de candidats ont abordé cette question avec succès.

Q31. et **Q32**. Les deux dernières questions, relativement ouvertes, laissaient la part belle aux propositions des candidats. Si la question 31 a permis de lire quelques propositions pertinentes de stratégie, son implantation dans la question 32 n'a fait l'objet que de très rares réponses correctes et complètes.

Conclusion

Les résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences certaines en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrice autorisée

2019 2019

Élasticité d'un brin d'ADN

La capacité des molécules d'ADN à participer à des mécanismes de réplication et de transcription ainsi qu'à s'organiser en chromosomes doit beaucoup à leur élasticité. Ainsi, l'étude de la réaction d'une molécule d'ADN aux contraintes mécaniques permet d'éclairer les processus biologiques mis en œuvre dans une cellule vivante.

À l'aide d'une expérience et de deux modèles mécaniques d'un brin d'ADN, ce sujet propose de caractériser l'élasticité de l'ADN. Pour cela, on suppose qu'on exerce une traction sur un brin d'ADN et on cherche à établir une relation entre la force utilisée et l'allongement de la molécule.

Le seul langage de programmation autorisé dans cette épreuve est Python. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet on suppose que les bibliothèques `math`, `numpy` et `random` ont été importées grâce aux instructions

```
import math
import numpy as np
import random
```

Si les candidats font appel à des fonctions d'autres bibliothèques, ils doivent préciser les instructions d'importation correspondantes.

Ce sujet utilise la syntaxe des annotations pour préciser le type des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, X:[float], c:str, u) -> (int, np.ndarray):
```

signifie que la fonction `maFonction` prend quatre arguments, le premier (`n`) est un entier, le deuxième (`X`) une liste de nombres à virgule flottante, le troisième (`c`) une chaine de caractères et le type du dernier (`u`) n'est pas précisé. Cette fonction renvoie un couple dont le premier élément est un entier et le deuxième un tableau numpy. Il n'est pas demandé aux candidats de recopier les entêtes avec annotations telles qu'elles sont fournies dans ce sujet, ils peuvent utiliser des entêtes classiques. Ils veilleront cependant à décrire précisément le rôle des fonctions qu'ils définiraient eux-mêmes.

Dans ce sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type `list`. Les termes « vecteur » et « tableau » désignent des objets numpy de type `np.ndarray`, respectivement à une dimension ou de dimension quelconque. Enfin le terme « séquence » représente une suite itérable et indiçable, indépendamment de son type Python, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers ».

Une attention particulière sera portée à la lisibilité, la simplicité et l'efficacité du code proposé. En particulier, l'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires et la description du principe de chaque programme seront appréciés.

Une liste de fonctions utiles est fournie à la fin du sujet.

I Fonctions utilitaires

Cette partie définit quelques fonctions qui pourront avantageusement être utilisées dans la suite du sujet.

Q 1. Écrire une fonction d'entête

```
def moyenne(X) -> float:
```

qui prend en paramètre une séquence de nombres et qui calcule la moyenne de ces nombres. Cette fonction ne doit pas modifier le paramètre `X`.

Par exemple : `moyenne([1, 2, 3, 4]) -> 2.5`

CC BY-NC-SA

Q 2. Écrire une fonction d'entête

```
def variance(X) -> float:
```

qui calcule la variance d'une séquence de nombres, sans la modifier. Pour rappel, la variance des n nombres x_1, ..., x_n est la moyenne des carrés des écarts à la moyenne, c'est-à-dire

$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2=\frac{1}{n}\sum_{i=1}^{n}x_i^2-\bar{x}^2 \qquad \text{avec} \quad \bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i.$$

Par exemple : `variance([1, 2, 3, 4]) -> 1.25`

Q 3. Écrire une fonction d'entête

```
def somme(M):
```

qui prend en paramètre une séquence imbriquée, de profondeur et de structure quelconques, dont tous les composants élémentaires sont des nombres, et calcule la somme de tous ces éléments.

Par exemple : `somme([[[1, 2], [3, 4, 5]], 6, [7, 8], 9]) -> 45`

Indication — L'expression booléenne `isinstance(x, numbers.Real)` permet de tester si `x` est un scalaire numérique. Par exemple

```
isinstance(1, numbers.Real) -> True
isinstance(2.3e4, numbers.Real) -> True
isinstance([1, 2, 3], numbers.Real) -> False
```

II Mesures expérimentales

Depuis quelques décennies, des équipes de recherche réussissent à isoler un brin d'ADN et à mesurer ses propriétés mécaniques. Cette partie s'appuie sur une série d'expériences réalisées dans les années 1990, en particulier au laboratoire de physique statistique de l'École Normale Supérieure.

Une molécule d'ADN est attachée à une de ses extrémités sur un support transparent, une microbille magnétique de diamètre 2,5 µm est greffée à son autre extrémité. À l'aide d'aimants, la molécule d'ADN est soumise à une force de traction notée $\vec{F}$. Afin de caractériser l'élasticité du brin d'ADN, on cherche à mesurer son allongement pour différentes intensités de la force de traction.

L'intensité de la force de traction n'est pas accessible directement, nous allons l'évaluer indirectement. Une fois le brin d'ADN mis en tension, son extrémité matérialisée par la bille ne reste pas immobile, elle est animée d'un mouvement aléatoire, dit mouvement brownien, dû à l'agitation des molécules du liquide qui l'entourent. En assimilant la molécule à un ressort, on montre que l'intensité de la force de traction est inversement proportionnelle aux fluctuations quadratiques moyennes de la position de la bille.

Une caméra CCD reliée à un ordinateur permet de photographier l'image de la bille (figure 1). Compte tenu de la taille de cette bille, on obtient une image de diffraction que nous allons analyser pour déterminer la position de la bille.

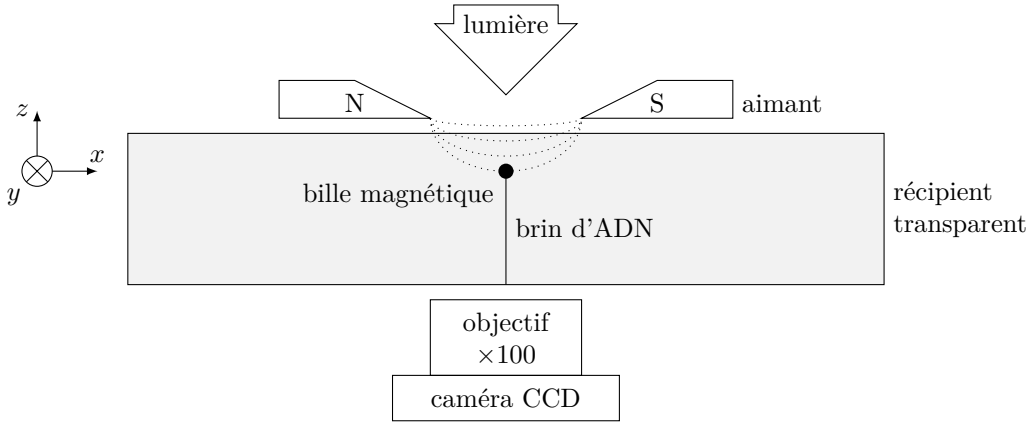

Figure 1 Schéma du dispositif expérimental (échelle non respectée)

II.A – Position de la bille

La figure 2 donne, à gauche, un exemple d'image obtenue par la caméra CCD. Cette caméra est pilotée par un programme Python qui récupère chaque image sous la forme d'un tableau d'entiers à deux dimensions. Les images obtenues sont en niveau de gris, chaque pixel est codé sur 8 bits, soit une valeur comprise entre 0 (noir) et 255 (blanc).

BY-NC-SA

Afin de repérer le centre de la figure de diffraction, l'image est convertie en noir et blanc inversé suivant une valeur seuil du niveau de gris : les pixels au-dessus du seuil deviennent noirs, ceux en dessous deviennent blancs. Une fois ce seuillage effectué, on calcule le barycentre des pixels blancs de l'image seuillée pour obtenir la position de la bille.

On rappelle que l'abscisse (respectivement ordonnée) du barycentre d'un ensemble de points de même poids est la moyenne des abscisses (respectivement ordonnées) des points considérés.

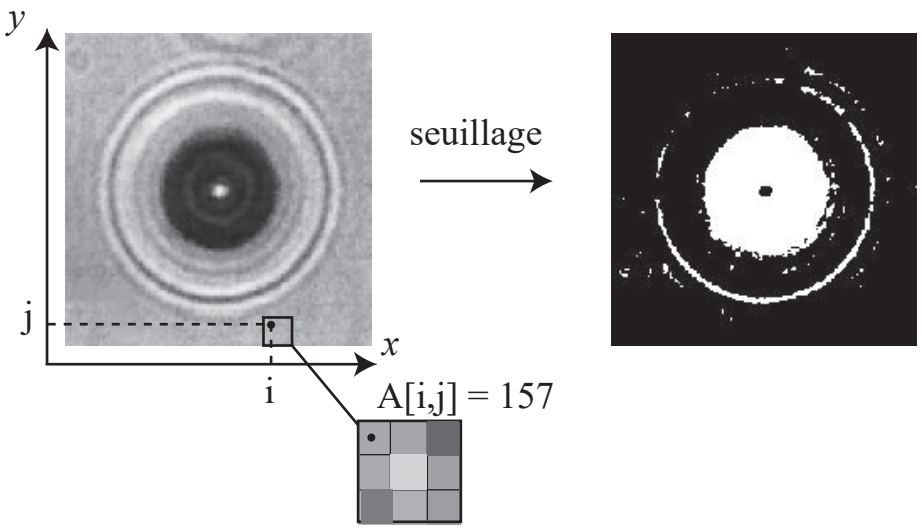

Figure 2 Figure de diffraction d'une bille et opération de seuillage

Q 4. Écrire une fonction d'entête

```
def seuillage(A:np.ndarray, seuil:int) -> np.ndarray:
```

qui prend en paramètre un tableau d'entiers à deux dimensions représentant un cliché de la caméra CCD et construit un tableau de même forme contenant la valeur 1 là où la valeur des pixels de l'image originale est strictement inférieure au seuil et la valeur 0 ailleurs (pixels supérieurs ou égaux au seuil).

Q 5. Écrire une fonction d'entête

```
def pixel_centre_bille(A:np.ndarray) -> (int, int):
```

qui prend en paramètre l'image seuillée telle que produite par la fonction `seuillage` et renvoie les indices (ligne et colonne) du pixel le plus proche du centre de la bille (barycentre des pixels à 1).

On dispose de la fonction d'entête

```
def prendre_photo() -> np.ndarray:
```

qui déclenche la prise d'un cliché par la caméra CCD et renvoie l'image prise sous la forme d'un tableau à deux dimensions tel que décrit plus haut.

Q 6. Écrire une fonction d'entête

```
def positions(n:int, seuil:int) -> [(int, int)]:
```

qui prend `n` photographies de la bille et renvoie la liste de ses positions dans chaque photographie en seuillant les images à la valeur `seuil`. Le résultat de cette fonction est donc une liste de `n` couples de deux entiers correspondants à l'indice de ligne et de colonne des positions successives du centre de la bille au cours de son mouvement brownien.

Le capteur CCD est positionné parallèlement au plan (xOy) et ses pixels sont carrés. La caméra a été calibrée dans les conditions de l'expérience : un pixel correspond à un carré du plan (xOy) de côté t.

Q 7. Définir une fonction d'entête

```
def fluctuations(P:[(int, int)], t:float) -> float:
```

qui prend en paramètre une liste de positions successives de la bille (telle que produite par la fonction `positions`) et la longueur correspondant à un pixel et calcule la valeur moyenne des déplacements quadratiques de la bille : moyenne des carrés des écarts entre chaque position mesurée et la position d'équilibre de la bille (correspondant au barycentre des différentes positions observées).

BY-NC-SA

II.B – Allongement du brin d'ADN

La position de la bille étant déterminée dans le plan (xOy), nous allons maintenant nous intéresser à sa cote, c'est-à-dire sa position dans la direction perpendiculaire à la caméra.

Pour déterminer la position de la bille suivant z, nous utilisons une méthode basée sur la répartition des cercles de la figure de diffraction. Pour cela, nous construisons un profil de cette figure en découpant l'image seuillée en anneaux concentriques centrés sur la position de la bille. Le décompte de la proportion de pixels blancs dans chaque anneau fournit un profil de la figure de diffraction qui permet de calculer la cote z de la bille en tenant compte des paramètres de calibration de la caméra.

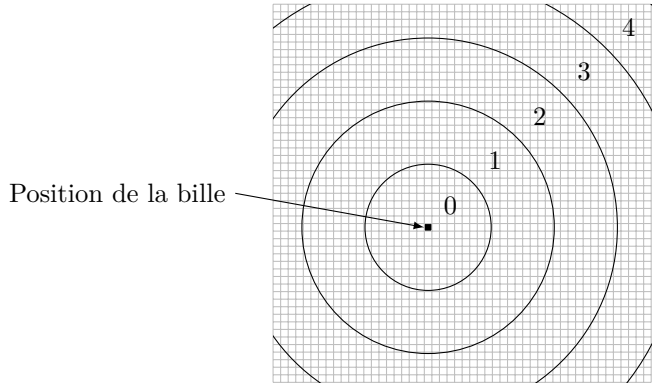

Figure 3 Exemple de découpage d'une image en cinq anneaux concentriques

Q 8. Écrire une fonction d'entête

```
def profil(A:np.ndarray, n:int):
```

qui construit le profil d'une figure de diffraction seuillée `A` en la découpant en `n` anneaux concentriques. Cette fonction renvoie, au choix du candidat, un vecteur ou une liste de `n` nombres, compris entre 0 et 1, qui donne la proportion de pixels blancs compris dans chaque anneau. Un pixel sera considéré comme contenu dans un anneau si son centre s'y trouve. L'élément d'indice 0 du résultat correspond à la bande la plus proche du centre de la figure (position de la bille).

Afin de clarifier l'écriture du code, il peut être pertinent de définir des fonctions intermédiaires pour programmer la fonction `profil`. Les candidats veilleront à expliquer précisément le rôle de chaque fonction intermédiaire qu'ils définiraient.

Q 9. Si on travaille sur une image carrée de dimension $p \times p$ pixels, quelle est la complexité de la fonction `profil` en fonction de p et de n (nombre d'anneaux) ?

II.C – Synthèse

Pour une configuration expérimentale donnée, il est ainsi possible en prenant une série de clichés de déterminer l'amplitude du mouvement brownien de l'extrémité du brin d'ADN ainsi que sa position en trois dimensions. Ces éléments permettent alors de déterminer l'intensité de la force de traction appliquée au brin d'ADN ainsi que son allongement.

En modifiant les aimants, on peut faire varier l'intensité du champ magnétique et donc la force appliquée au brin d'ADN. En renouvelant l'expérience, on obtient ainsi une série de points expérimentaux correspondant à diverses valeurs de force et d'allongement.

III Modèle du ver

Le « modèle du ver » est un modèle souvent utilisé pour décrire le comportement mécanique de certains polymères. Dans ce modèle, la molécule étudiée est représentée par une succession de segments semi-rigides orientés grossièrement dans la même direction. Il permet d'obtenir une expression simplifiée de F, l'intensité de la force de traction $\vec{F}$, en fonction de z, l'allongement de la molécule :

$$F(z) = \frac{k_B T}{L_p}\left(\frac{1}{4(1-z/L_0)^2} - \frac{1}{4} + \frac{z}{L_0}\right) \tag{III.1}$$

où k_B est la constante de Boltzman et T la température. Ce modèle est paramétré par deux longueurs :

— L_p, longueur de persistance représentant la longueur typique sur laquelle le polymère maintient sa forme malgré les déformations dues à l'agitation thermique ;

— L_0, extension maximale du polymère.

CC BY-NC-SA

III.A – Calcul des paramètres

Ces deux grandeurs ne sont pas accessibles directement pour une molécule d'ADN. L'objectif de cette partie est de déterminer les valeurs de L_p et L_0 correspondant au brin d'ADN objet des mesures développées dans la partie précédente.

Pour cela nous utilisons la fonction `curve_fit` du package `scipy.optimize` qui permet d'ajuster les paramètres d'une courbe afin qu'elle passe au plus proche d'un certain nombre de points. La fonction `curve_fit` utilise pour cela une méthode de moindres carrés non linéaire. Une adaptation de la documentation de cette fonction est fournie par la figure 4.

```
popt, pcov = scipy.optimize.curve_fit(f, xdata, ydata)
```

Paramètres

— `f` : callable
La fonction modèle, `f(x, ...)`. Elle doit prendre la variable indépendante comme premier argument et chaque paramètre à ajuster comme argument suivant.

— `xdata` : séquence de longueur `M`
La liste des valeurs de la variable indépendante correspondant aux différentes mesures.

— `ydata` : séquence de longueur `M`
Les mesures, typiquement `f(xdata, ...)`.

Résultat

— `popt` : tableau
Valeurs optimales des paramètres telles que la somme des carrés des écarts `f(xdata, *popt) - ydata` soit minimale.

— `pcov` : tableau à deux dimensions
Une estimation de la covariance de `popt`. Les termes diagonaux donnent la variance de l'estimateur du paramètre correspondant.
Pour estimer l'écart-type de l'erreur sur les paramètres, on peut utiliser `perr = np.sqrt(np.diag(pcov))`.

Figure 4 Extrait adapté de la documentatin de `curve_fit`

Q 10. Écrire une fonction d'entête

```
def force(z:np.ndarray, Lp:float, L0:float, T:float) -> np.ndarray:
```

qui calcule la force donnée par la formule (III.1) pour chaque élément du vecteur `z`. Cette fonction renvoie un vecteur de même taille que `z` contenant le résultat du calcul pour chaque composante de `z`. La variable globale `K_B` fournit la valeur de la constante de Boltzman.

On dispose d'une série de points expérimentaux issus d'essais réalisés suivant les modalités décrites dans le II.C. Ces valeurs expérimentales sont stockées dans un tableau à deux dimensions. Chaque ligne (première dimension) contient le résultat d'une mesure, la première colonne donne la valeur obtenue pour la force et la deuxième celle de l'allongement.

Q 11. Écrire une fonction d'entête

```
def ajusteWLC(Fz:np.ndarray, T:float) -> (float, float):
```

qui ajuste les paramètres de la formule (III.1) pour qu'ils correspondent au mieux aux valeurs expérimentales du tableau `Fz` obtenues par une série d'essais effectués à la température `T`. Cette fonction renvoie un couple de nombres donnant les valeurs optimales de L_p et de L_0.

III.B – Algorithme de minimum local

La fonction `curve_fit` permet d'utiliser différents algorithmes d'optimisation. Nous allons jeter les bases d'un algorithme permettant d'obtenir les valeurs optimales L_p et L_0.

III.B.1) Implantation d'un algorithme de minimisation 1D

Soit ϕ une fonction de classe C^2 sur $\mathbb{R}$ présentant un minimum local.

On rappelle que

$$\frac{\phi(x(1+h)) - \phi(x(1-h))}{2xh} \tag{III.2}$$

est une expression approchée d'ordre 2 de la dérivée de ϕ en x (notée $\phi'(x)$).

CC BY-NC-SA

On suppose que l'ordinateur utilisé représente les nombres flottants sur 64 bits avec un bit de signe, 11 bits d'exposant et 52 bits de mantisse.

Q 12. Calculer le nombre de chiffres significatifs décimaux donnés par ce codage.

Q 13. Justifier que les valeurs $h = 1$ et $h = 10^{-16}$ ne permettent pas obtenir une bonne approximation du nombre dérivé $\phi'(x)$. Proposer alors une valeur adaptée de h.

Q 14. Écrire une fonction d'entête

```
def derive(phi, x:float, h:float) -> float:
```

qui calcule une valeur approchée de la dérivée au point `x` de `phi`, fonction réelle d'une variable réelle, où `h` correspond au h de la formule (III.2).

Q 15. Écrire une fonction d'entête

```
def derive_seconde(phi, x:float, h:float) -> float:
```

permettant d'obtenir une approximation de la dérivée seconde de la fonction `phi` au point x.

Q 16. Écrire une fonction d'entête

```
def min_local(phi, x0:float, h:float) -> float:
```

basée sur la méthode de Newton permettant de trouver l'abscisse d'un minimum local de la fonction `phi`. La valeur approchée de cette abscisse vérifiera $|\phi'(x)| < 10^{-7}$.

III.B.2) Implantation d'un algorithme de minimisation 2D

L'écart quadratique entre les valeurs expérimentales de la force F_i correspondant à l'élongation z_i et les valeurs de la fonction `force` est défini par

$$E(L_p, L_0) = \sum_i \big(F_i - \texttt{force}(z_i, L_p, L_0, T)\big)^2.$$

Les valeurs optimales de L_p et L_0 correspondent au minimum de la fonction E, c'est-à-dire à un point où son gradient est nul. Pour déterminer le point (x_m, y_m) correspondant au minimum de la fonction E, nous allons adapter la méthode de Newton unidimensionnelle pour rechercher un zéro d'une fonction de deux variables puis appliquer cette méthode au gradient de E.

On considère G une fonction réelle de deux variables réelles x, y de classe C^2 sur $\mathbb{R}^2$, présentant un minimum local. On note $g_x = \dfrac{\partial G}{\partial x}$ et $g_y = \dfrac{\partial G}{\partial y}$ les composantes du gradient de la fonction G. On rappelle que

$$g_x(x,y) = g_x(x_0,y_0) + \frac{\partial g_x}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial g_x}{\partial y}(x_0,y_0)(y-y_0) + o(x-x_0, y-y_0)$$

$$g_y(x,y) = g_y(x_0,y_0) + \frac{\partial g_y}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial g_y}{\partial y}(x_0,y_0)(y-y_0) + o(x-x_0, y-y_0)$$

L'objectif est d'approcher les valeurs x_m, y_m qui annulent les fonctions g_x et g_y et correspondent donc à un extremum de la fonction G, en partant d'un point arbitraire (x_0, y_0).

Q 17. Montrer que les coordonnées $(x_1, y_1, 0)$ du point situé à l'intersection des plans

— tangent à la surface $z = g_x(x,y)$ au point $(x_0, y_0, g_x(x_0,y_0))$,

— tangent à la surface $z = g_y(x,y)$ au point $(x_0, y_0, g_y(x_0,y_0))$,

— d'équation $z = 0$,

vérifient la relation suivante où on explicitera l'expression de $J(x_0, y_0)$:

$$\begin{pmatrix} -g_x(x_0,y_0) \\ -g_y(x_0,y_0) \end{pmatrix} = J(x_0,y_0) \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix}.$$

Si la matrice J est inversible, en s'inspirant de la méthode de Newton, on construit une relation de récurrence sous la forme :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - J^{-1}(x_n,y_n) \begin{pmatrix} g_x(x_n,y_n) \\ g_y(x_n,y_n) \end{pmatrix}$$

Nous allons utiliser cette relation pour recherche le minimum local d'une fonction réelle de deux variables réelles implantée en Python sous la forme d'une fonction prenant en paramètre une séquence de deux nombres. Par exemple

```
def fct_dont_je_veux_le_minimum(X:np.ndarray) -> float:
```


CC BY-NC-SA

Q 18. Écrire une fonction d'entête

```
def grad(G, X:np.ndarray, h:float) -> np.ndarray:
```

qui fournit une approximation de la valeur du gradient de la fonction réelle de deux variables réelles `G` au point `X` ($= (x, y)$) en utilisant `h` comme paramètre pour le calcul approché des dérivées, voir formule (III.2).

Q 19. Écrire une fonction d'entête

```
def min_local_2D(G, X0:np.ndarray, h:float) -> np.ndarray:
```

permettant d'obtenir une approximation numérique des valeurs de x_m et y_m correspondant à un minimum local de la fonction `G` en partant du point `X0` ($= (x_0, y_0)$) et en utilisant `h` comme paramètre pour le calcul approché des dérivées. La valeur approchée du minimum local vérifiera $|g_x(x, y)| < 10^{-7}$ et $|g_y(x, y)| < 10^{-7}$.

IV Modèle de la chaine librement jointe

La molécule d'ADN peut également être représentée par le modèle de la « chaine librement jointe » dans laquelle des segments rigides (appelés *monomères*) sont liés à leurs extrémités et librement orientables aux points de jointure. On appelle *conformation* de la molécule sa configuration géométrique. En l'absence d'action extérieure, l'orientation de chaque segment par rapport à ses voisins est aléatoire et toutes les conformations sont équiprobables.

Ce modèle supposant une part d'aléa, il n'est plus possible, comme dans le modèle du ver, d'obtenir une formule liant directement la force et l'allongement. Nous allons donc utiliser un programme informatique pour simuler le comportement de ce modèle de molécule et obtenir l'allongement en fonction de la force utilisée.

La simulation proposée est basée sur la méthode dite de « Monte Carlo » faisant participer nombres aléatoires, statistiques et probabilités. Dans le sens plus spécifique des simulations moléculaires, un modèle de la molécule est développé pour calculer une énergie, puis des changements aléatoires sont effectués pour converger vers l'état naturel de la molécule. Une fois la convergence atteinte, des calculs statistiques permettent d'approximer les paramètres cherchés.

Par souci de simplicité, nous travaillons dans un espace à deux dimensions.

IV.A – Modélisation plane

La molécule comporte n monomères de longueur l. On note $\theta_i \in [-\pi, \pi[$ ($i \in [\![0, n-1]\!]$) l'angle formé par le segment i avec la direction de la force $\vec{F}$. Les angles θ_i définissent la conformation de la molécule (figure 5).

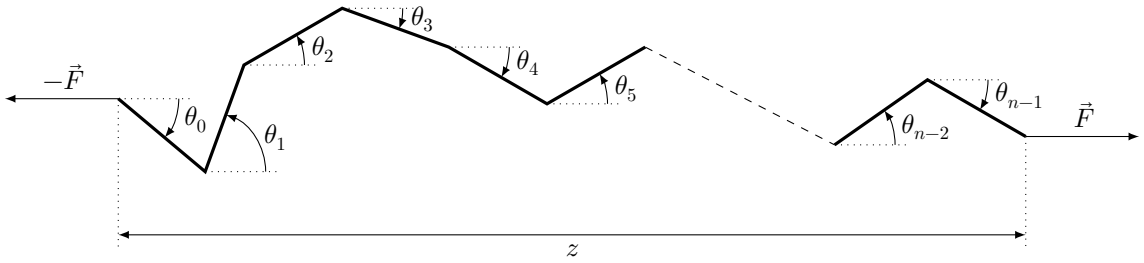

Figure 5 Représentation d'un brin d'ADN sous forme de chaine librement jointe

Pour un brin d'ADN soumis à une force $\vec{F}$ imposée, l'énergie mécanique E développée pour étendre la molécule s'écrit

$$E = -zF \tag{IV.1}$$

où F est l'intensité de la force et z l'allongement de la molécule suivant la direction de la force (figure 5).

La simulation démarre à partir d'une conformation aléatoire du brin d'ADN.

Q 20. Écrire une fonction d'entête

```
def conformation(n:int):
```

qui génère une conformation aléatoire d'un brin d'ADN composé de `n` segments. Cette fonction renvoie une liste ou un vecteur de longueur `n` correspondant à l'orientation (angle θ_i) de chaque segment.

Q 21. Écrire une fonction d'entête

```
def allongement(theta, l:float) -> float:
```

qui calcule l'allongement z de la chaine dans la conformation `theta` pour une longueur de segment `l`.

BY-NC-SA

La modification de la conformation se fait en modifiant de façon aléatoire k angles successifs, k étant un paramètre de simulation ajustable.

Q 22. Écrire une fonction d'entête

```
def nouvelle_conformation(theta, k:int):
```

qui crée une nouvelle conformation, à partir de la conformation `theta` en modifiant `k` valeurs successives à partir d'un indice aléatoire.

IV.B – Critère de Metropolis Monte Carlo (MMC)

La méthode spécifique utilisée dans la plupart des études génétiques a été développée par Metropolis *et al.* en 1953. Une probabilité P simule l'agitation thermique de la molécule. Cette agitation tend à désordonner la molécule (maximum d'entropie), alors que la force extérieure tend à aligner les brins (diminution de l'entropie). Les deux phénomènes convergent vers une situation d'équilibre statistique, où force et allongement moyen sont liés. L'algorithme vise à déterminer cet équilibre statistique.

À partir d'une conformation de départ, d'énergie calculée E_1, une nouvelle conformation est crée et son énergie E_2 est calculée. Si cette nouvelle conformation possède une énergie inférieure à celle de son précurseur ($E_2 < E_1$), elle est conservée. Si $E_2 \geqslant E_1$, la nouvelle conformation est conservée, avec la probabilité

$$P = \exp\left(\frac{E_1 - E_2}{k_B T}\right) \tag{IV.2}$$

où k_B est la constante de Boltzman et T la température. Si la nouvelle conformation est rejetée, c'est la conformation de départ, d'énergie E_1, qui est conservée pour la suite de la simulation.

Q 23. Écrire une fonction d'entête

```
def selection_conformation(thetaA, thetaB, F:float, l:float, T:float):
```

qui prend en paramètre deux conformations successives `thetaA` et `thetaB` (`thetaA` étant le précurseur de `thetaB`) et renvoie la conformation conservée connaissant `F`, l'intensité de la force de traction, `l` la longueur d'un monomère et `T` la température.

IV.C – Implantation de la simulation

L'algorithme est supposé avoir convergé lorsque la variance de l'allongement du brin d'ADN sur les 500 dernières itérations est inférieure à une valeur ε, paramètre de la simulation.

Q 24. Écrire une fonction d'entête

```
def monte_carlo(F:float, n:integer, l:float, T:float, k:integer, epsilon:float) -> float:
```

qui simule l'application d'une force de traction d'intensité `F` sur un brin d'ADN de `n` monomères de longueur `l`, à la température `T`. Les arguments `k` et `epsilon` correspondent aux paramètres de la simulation présentés plus haut. Le résultat de la fonction est l'allongement moyen des 500 dernières conformations, une fois la convergence atteinte.

Les candidats ont la liberté de concevoir et d'utiliser les structures de données qui leur semblent les mieux adaptées à la programmation de la fonction `monte_carlo`. Ils veilleront à préciser le rôle et l'organisation des données manipulées qui ne seraient pas déjà décrites dans le sujet.

Indication — Compte tenu du nombre d'itérations envisagées, il est prudent de ne pas enregistrer toutes les étapes intermédiaires de la simulation. On pourra considérer l'utilisation d'une file pour stocker les données utiles à la simulation. À contrario d'une pile, une file est une structure de données où les premiers éléments ajoutés à la file sont les premiers à en être retirés (« First In First Out »). En Python, une liste peut être utilisée pour représenter une file grâce aux opérations `append` et `pop(0)`.

Une fois la fonction `monte_carlo` développée, il est trivial de l'utiliser pour simuler différentes intensités de la force et obtenir l'allongement correspondant du brin d'ADN simulé.

CC BY-NC-SA

Opérations et fonctions Python disponibles

Fonctions

— `range(n)` renvoie la séquence des `n` premiers entiers ($0 \to n-1$)
`list(range(5))` → `[0, 1, 2, 3, 4]`
— `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers)
— `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme
— `random.shuffle(u)` permute aléatoirement les éléments de la liste `u` (modifie `u`)
— `random.sample(u, n)` renvoie une liste de `n` éléments distincts de la liste `u` choisis aléatoirement, si `n > len(u)`, déclenche l'exception `ValueError`
— `math.sqrt(x)` calcule la racine carrée du nombre `x`
— `round(n)` arrondit le nombre `n` à l'entier le plus proche
— `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à `x`
— `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`

Opérations sur les listes

— `len(u)` donne le nombre d'éléments de la liste `u` :
`len([1, 2, 3])` → 3 ; `len([[1,2], [3,4]])` → 2
— `u + v` construit une liste constituée de la concaténation des listes `u` et `v` :
`[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`
— `n * u` construit une liste constituée de la liste `u` concaténée `n` fois avec elle-même :
`3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`
— `e in u` et `e not in u` déterminent si l'objet `e` figure dans la liste `u`, cette opération a une complexité temporelle en $O($`len`$(u))$
`2 in [1, 2, 3]` → `True` ; `2 not in [1, 2, 3]` → `False`
— `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (similaire à `u = u + [e]`)
— `u.pop(i)` : renvoie l'élément à l'indice `i` de la liste `u` et le supprime
— `del u[i]` supprime de la liste `u` son élément d'indice `i`
— `del u[i:j]` supprime de la liste `u` tous ses éléments dont les indices sont compris dans l'intervalle $[$`i`$,$ `j`$[$
— `u.remove(e)` supprime de la liste `u` le premier élément qui a pour valeur `e`, déclenche l'exception `ValueError` si `e` ne figure pas dans `u`, cette opération a une complexité temporelle en $O($`len`$(u))$
— `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste
— `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`

Opérations sur les tableaux (`np.ndarray`)

— `np.array(u)` crée un nouveau tableau contenant les éléments de la séquence `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`
— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique numpy (`np.int16`, `np.float32`, ...). Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float` par défaut
— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens
— `a.ndim` nombre de dimensions du tableau `a`
— `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions
— `len(a)` taille du tableau `a` dans sa première dimension, équivalent à `a.shape[0]`
— `a.size` nombre total d'éléments du tableau `a`
— `a.flat` itérateur sur tous les éléments du tableau `a`
— `np.ndenumerate(a)` itérateur sur tous les couples (index, élément) du tableau `a` où « index » est un tuple de `a.ndim` entiers donnant les indices de l'élément
— `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O($`a.size`$)$
— `b in a` détermine si `b` est un élément du tableau `a` ; si `b` est un scalaire, vérifie si `b` est un élément de `a` ; si `b` est un vecteur ou une liste et `a` un tableau à deux dimensions, détermine si `b` est une ligne de `a`

BY-NC-SA

— `np.concatenate((a1, a2))` construit un nouveau tableau en concaténant deux tableaux selon leur première dimension ; `a1` et `a2` doivent avoir le même nombre de dimensions et la même taille à l'exception de leur taille dans la première dimension (deux tableaux à deux dimensions doivent avoir le même nombre de colonnes pour pouvoir être concaténés)

— `np.transpose(a)` renvoie le transposé du tableau `a`

— `np.dot(a, b)` calcule le produit matriciel des tableaux `a` et `b`

— `np.linalg.inv(a)` renvoie l'inverse du tableau `a`, lève l'exception `ValueError` si `a` n'est pas un tableau carré à deux dimensions et `LinAlgError` si `a` n'est pas inversible

---

• • • FIN • • •

BY-NC-SA

Concours Centrale-Supélec 2019 filière PC

Informatique

Présentation du sujet

Le sujet porte sur la mesure de la raideur d'un brin d'ADN ainsi que la simulation numérique de son comportement.

Après une première partie préliminaire sur des algorithmes classiques itératifs et récursifs, la seconde partie vise à réaliser un traitement d'image permettant de mesurer les fluctuations de positions et l'allongement du brin en fonction de la tension exercée. Cette partie mobilisait des compétences en algorithmique.

La troisième partie permet d'identifier les paramètres du modèle du ver à partir des données expérimentales par minimisation des écarts. Cette partie s'appuyait sur les compétences en simulation numérique au programme.

La quatrième partie propose une modélisation simple du brin sous forme d'une succession de segments, dont le comportement statistique permet de retrouver par simulation un comportement proche de celui du brin d'ADN. Bien qu'il s'agisse d'une simulation numérique, les compétences évaluées relevaient pour l'essentiel de l'algorithmique.

Analyse globale des résultats

Le sujet est de longueur raisonnable pour le temps imparti. Près de la moitié des candidats ont abordé plus de 80 % des questions.

À nouveau cette année, le jury se réjouit du niveau satisfaisant des copies. Le langage est bien maitrisé et permet de traduire les solutions aux questions sans difficulté. Quelques rares candidats ont visiblement négligé la formation en informatique et se contentent de répondre aux questions ne relevant pas immédiatement d'informatique. Ces copies conduisent à des notes très faibles.

Les petites erreurs syntaxiques n'ont pas été sanctionnées par le jury, dans la mesure où elles ne cachent pas des erreurs de fond. Les réponses pertinentes d'un point de vue algorithmique sont valorisées. Néanmoins, une accumulation significative de ces erreurs dans certaines copies a pu conduire à une dépréciation.

Certaines copies proposent des programmes particulièrement élégants et concis et reflètent un vrai recul sur les différentes stratégies de programmation. Ces copies ont été valorisées.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Au regard des copies évaluées, le jury propose aux futurs candidats de prêter attention aux remarques suivantes.

— L'indentation en Python délimite les blocs d'instructions et doit apparaitre clairement dans la rédaction. Toute rédaction claire est bienvenue ; bien souvent, un trait vertical marquant l'alignement du bloc d'instruction est suffisant.

— L'initialisation d'une variable dans une boucle ou hors de la boucle n'a pas les mêmes conséquences pour l'algorithme.

— Les espaces de noms en Python sont organisés en poupées russes : il peut parfois être utile de définir une fonction à l'intérieur d'une fonction.

- Le nombre d'itérations d'une boucle doit être bien réfléchi pour s'assurer que les indices des éléments d'une liste appelés dans la boucle sont bien définis. L'expression `range(n)` fournit les entiers de l'intervalle $[0, n-1]$ et permet donc d'effectuer n itérations indicées de 0 à $n-1$. Les opérateurs `and` et `or` utilisent une évaluation paresseuse, l'ordre des clauses peut donc avoir de l'importance, par exemple dans la condition d'une instruction `while`.
- Lorsque le sujet précise explicitement les arguments des fonctions et les valeurs retournées, ainsi que leur type, il convient de veiller à les respecter.
- Les listes ou les tableaux fournis en argument d'une fonction peuvent subir des modifications par effet de bord s'ils ne sont pas copiés, ce qui n'est pas toujours souhaitable. L'instruction `A = B` ne réalise pas de copie de l'objet désigné par `B`, s'il est mutable sa modification affectera les deux variables.
- Beaucoup de questions sont indépendantes et généralement le prototype d'une fonction, donné dans une question, permet de l'utiliser dans les questions suivantes.
- Les opérateurs classiques (`+`, `*`, etc.) n'ont pas toujours le même sens selon les types des opérandes, en particulier pour les listes et les tableaux.
- La concision et l'élégance des programmes sont appréciées dans l'évaluation. Les candidats qui réinvestissent les fonctions déjà codées sont valorisés par rapport à ceux qui recopient les lignes de code équivalentes.
- Des listes de conditions en cascade nuisent à la lisibilité de l'algorithme. Une condition booléenne bien choisie distingue les candidats dont la pensée est claire.
- Des noms de variables explicites aident à la compréhension du code. De trop nombreux candidats utilisent des noms de variables quelconques (`a`, `b`, `c`...) ce qui nuit à la compréhension du programme. La clarté du programme (en particulier le choix des noms de variables) ainsi que la présence de commentaires opportuns sont prises en compte dans l'évaluation.
- Un long paragraphe expliquant le principe d'un algorithme est souvent moins clair qu'un code bien formulé, utilisant des noms de variables évoquant leur contenu.
- Lors d'un calcul de complexité, une justification minimale est attendue.
- L'ordre des questions importe. Prendre soin de rédiger les réponses aux questions en respectant leur ordre dans le sujet.
- La qualité d'expression (l'orthographe notamment) et la qualité visuelle de présentation relèvent des compétences de communication indispensables à un candidat à une école d'ingénieur. Le correcteur n'attribue les points qu'aux éléments de réponse qu'il parvient à lire et à comprendre. Les copies obscures et difficiles à comprendre sont pénalisées.
- Les variables utilisées dans une fonction doivent être définies dans cette fonction ou être explicitement définies comme variables globales (soit par le sujet, soit par le candidat).
- Les candidats sont invités à bien lire l'annexe contenant certaines fonctions utiles pour traiter le sujet.

I Fonctions utilitaires

La première partie comporte trois questions préliminaires, participant à la progressivité du sujet. Les deux premières questions sur la moyenne et la variance sont réussies par la majorité des candidats. Certains candidats utilisent la fonction moyenne dans les itérations du calcul de la variance, ce qui conduit à une complexité en $O(n^2)$ au lieu de $O(n)$. La question 3 se traitait naturellement par un

appel récursif de la fonction `somme` et a été réussie dans 36 % des copies. De très rares candidats ont pu traiter cette question en aplatissant la liste, sans appel récursif.

II Mesures expérimentales

Les questions **Q4** à **Q6** sont très bien réussies. Il convient de ne pas stocker les photographies dans une liste pour éviter une occupation mémoire inutile. Lorsque la question demande de renvoyer un nouveau tableau, il convient de ne pas modifier le tableau fourni en argument.

La question **Q7** est moins bien abordée ; elle réutilise naturellement les fonctions antérieures.

Les questions **Q8** et **Q9**, plus difficiles, furent discriminantes. Les candidats avaient toute latitude pour proposer plusieurs fonctions permettant de déterminer un profil radial de l'image de diffraction. La discrétisation des anneaux, la répartition des pixels dans chaque anneau et le calcul du profil lui-même nécessitaient une approche structurée pour réussir. Bien que les candidats parvenus à une solution fonctionnelle soit rares, toutes les propositions répondant à certains points clés de l'algorithme ont été valorisées. Cinquante-huit pour cent des candidats ont abordé ces questions en obtenant, en moyenne, 41 % des points du barème. Lorsqu'il est abordé, le calcul de complexité est bien traité.

III Modèle du ver

La question **Q11**, visant à mettre en œuvre la fonction d'une bibliothèque, a posé des difficultés à la plupart des candidats. Il s'agissait de préparer les données en dissociant les abscisses et les ordonnées, préparer la fonction en créant une nouvelle fonction sans l'argument `T` et renvoyer la partie utile des résultats.

Les questions **Q12** et **Q13** ont montré un recul très relatif de certains candidats sur les nombres flottants utilisés en calcul numérique. Combien d'entre eux proposent 52 chiffres significatifs décimaux ou encore 10^{15} chiffres significatifs ! Beaucoup indiquent aussi que 10^{-16} est trop petit pour être représenté en mémoire. La représentation des données en mémoire est pourtant bien au programme.

Les questions **Q14** et **Q15** sont relativement bien réussies. La fonction dérivée est souvent mal utilisée pour le calcul de la dérivée seconde. La questions **Q16**, proposant une minimisation par recherche du zéro de la dérivée par la méthode de Newton, est souvent bien abordée dans le principe mais trop souvent les algorithmes des candidats recherchent le zéro de la fonction `phi` et non de sa dérivée.

La question **Q17**, ne faisant pas appel à la programmation, est souvent traitée correctement. La mise en œuvre au travers des programmes des questions **Q18** et **Q19** est cependant rarement réussie.

IV Modèle de la chaine librement jointe

Les questions **Q20** et **Q21** sont souvent bien abordées, mais obtenir un nombre aléatoire entre $-\pi$ et π semble poser des difficultés à certains candidats. La fonction `random.randrange` n'était pas adaptée pour cette question. La question **Q22** visait à créer une nouvelle conformation (et non pas modifier la conformation donnée) en changeant k angles successifs. Le tirage aléatoire du début de la section modifiée en s'assurant de ne pas dépasser la taille du tableau a conduit à de nombreuses erreurs.

La question **Q23** nécessite de sélectionner une des conformations selon une probabilité donnée, ce que tous les candidats n'ont pas su faire.

La question **Q24** propose de mettre en œuvre une simulation de type Monte Carlo. Elle donnait l'occasion d'utiliser une file, dont les propriétés étaient rappelées, très similaires à celles des piles au programme. Cette question est abordée par seulement 23 % des candidats qui, lorsqu'ils l'abordent, la réussissent plutôt bien.

Conclusion

Le sujet aborde une large partie du programme d'informatique commune, dont le programme de deuxième année. Le choix d'un sujet s'appuyant sur la caractérisation d'un phénomène physique par une approche numérique, impliquant une part d'algorithmique, assure une cohérence avec la formation d'ingénieur. Cette approche sera reconduite sur des problématiques de simulation ou d'algorithmique en informatique, à partir du programme des trois semestres d'informatique.

Les bons résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences affirmées en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrices autorisées

2018 *2018*

Simulation de la cinétique d'un gaz parfait

La théorie cinétique des gaz vise à expliquer le comportement macroscopique d'un gaz à partir des mouvements des particules qui le composent. Depuis la naissance de l'informatique, de nombreuses simulations numériques ont permis de retrouver les lois de comportement de différents modèles de gaz comme celui du gaz parfait.

Ce sujet s'intéresse à un gaz parfait monoatomique. Nous considérerons que le gaz étudié est constitué de N particules sphériques, toutes identiques, de masse m et de rayon R, confinées dans un récipient rigide. Les simulations seront réalisées dans un espace à une, deux ou trois dimensions ; le récipient contenant le gaz sera, suivant le cas, un segment de longueur L, un carré de côté L ou un cube d'arête L.

Dans le modèle du gaz parfait, les particules ne subissent aucune force (leur poids est négligé) ni aucune autre action à distance. Elle n'interagissent que par l'intermédiaire de chocs, avec une autre particule ou avec la paroi du récipient. Ces chocs sont toujours élastiques, c'est-à-dire que l'énergie cinétique totale est conservée.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet on suppose que les bibliothèques `math`, `numpy` et `random` ont été importées grâce aux instructions

```
import math
import numpy as np
import random
```

Si les candidats font appel à des fonctions d'autres bibliothèques ils doivent préciser les instructions d'importation correspondantes.

Ce sujet utilise la syntaxe des annotations pour préciser le types des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, x:float, l:[str]) -> (int, np.ndarray):
```

signifie que la fonction `maFonction` prend trois arguments, le premier est un entier, le deuxième un nombre à virgule flottante et le troisième une liste de chaines de caractères et qu'elle renvoie un couple dont le premier élément est un entier et le deuxième un tableau numpy. Il n'est pas demandé aux candidats de recopier les entêtes avec annotations telles qu'elles sont fournies dans ce sujet, ils peuvent utiliser des entêtes classiques. Ils veilleront cependant à décrire précisément le rôle des fonctions qu'ils définiraient eux-mêmes.

Une liste de fonctions utiles est donnée à la fin du sujet.

Représentation en Python

Chaque particule est représentée par une liste de deux éléments, le premier correspond à la position de son centre, la deuxième à sa vitesse. Chacun de ces éléments (position et vitesse) est représenté par un vecteur (`np.ndarray`) dont le nombre de composantes correspond à la dimension de l'espace de simulation.

Les positions et vitesses sont exprimées sous forme de coordonnées cartésiennes dans un repère orthonormé dont l'origine est placée dans un coin du récipient contenant le gaz et dont les axes sont parallèles aux côtés du récipient issus de ce coin de façon à ce que tout point situé à l'intérieur du récipient ait ses coordonnées comprises entre 0 et L.

Les positions sont exprimées en mètres et les vitesses en $\mathrm{m{\cdot}s^{-1}}$. La figure 1 propose des exemples de particules dans des espaces de diverses dimensions.

```
p1 = [np.array([5.3]), np.array([412.3])] # 1D
p2 = [np.array([3.1, 4.8]), np.array([241, -91.4])] # 2D
p3 = [np.array([5.2, 3.2, 2.3]), np.array([-130.1, 320, 260.2])] # 3D
```

Figure 1 Exemples de particules

BY-NC-SA

I Initialisation

Pour pouvoir réaliser une simulation, il convient de disposer d'une situation initiale, c'est-à-dire d'un ensemble de particules réparties dans le récipient et dotées d'une vitesse initiale connue. Cette partie s'intéresse au positionnement aléatoire d'un ensemble de particules. L'attribution de vitesses initiales à ces particules ne sera pas abordé ici.

I.A – Placement en dimension 1

Nous cherchons d'abord comment placer N particules (sphères de rayon R) le long d'un segment de longueur L sans qu'elles se chevauchent ni qu'elles sortent du segment. La figure 2 montre quelques exemples de placements possibles avec $N = 5$, $R = 0{,}5$ et $L = 10$.

Figure 2 Exemples de placement de 5 particules de rayon 0,5 sur un segment de longueur 10

La fonction `placement1D` construit aléatoirement, à partir des paramètres géométriques du problème (nombre et rayon des particules, taille du récipient), une liste de coordonnées correspondant à la position initiale du centre de chaque particule.

```
1   def placement1D(N:int, R:float, L:float) -> [np.ndarray]:

2       def possible(c:np.ndarray) -> bool:
3           if c[0] < R or c[0] > L - R: return False
4           for p in res:
5               if abs(c[0] - p[0]) < 2*R: return False
6           return True

7       res = []
8       while len(res) < N:
9           p = L * np.random.rand(1)
10          if possible(p): res.append(p)
11      return res
```

Q 1. Détailler l'action de la ligne 9.

Q 2. Quelle est la signification du paramètre `c` de la fonction `possible` (ligne 2) ?

Q 3. Expliquer le rôle de la ligne 3.

Q 4. Expliquer le rôle des lignes 4 et 5.

Q 5. Donner en une phrase le rôle de la fonction `possible`.

Q 6. Proposer une nouvelle version de la ligne 9 permettant d'éviter certains rejets de la part de la fonction `possible`.

Q 7. On considère l'appel `placement1D(4, 0.5, 5)` et on suppose que les trois premières particules ont été placées aux points d'abscisses 1, 2,5 et 4 (figure 3). Quelle sera la suite du déroulement de la fonction `placement1D` ?

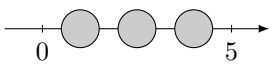

Figure 3

Q 8. Quelle est la complexité temporelle de la fonction `placement1D` dans le cas où $N \ll N_{\max}$, nombre maximal de particules de rayon R pouvant être placées sur un segment de longueur L ?

Q 9. Pour remédier de manière simple (mais non optimale) à la situation de la question 7, on décide de recommencer à zéro le placement des particules dès qu'une particule est rejetée par la fonction `possible`. Réécrire les lignes 7 à 11 de la fonction `placement1D` pour mettre en œuvre cette décision.

BY-NC-SA

I.B – Optimisation du placement en dimension 1

Pour placer aléatoirement N particules le long d'un segment, nous envisageons une approche plus efficace que celle étudiée dans la sous-partie I.A.

L'idée est de calculer l'espace laissé libre sur le segment cible par N particules puis de répartir aléatoirement cet espace libre entre les particules. Afin de conserver une répartition uniforme des particules dans tout le segment, nous utilisons l'algorithme suivant :

1. déterminer ℓ, espace laissé libre par les N particules dans le segment $[0, L[$;
2. placer aléatoirement dans le segment $[0, \ell[$, N particules virtuelles ponctuelles ($R = 0$) ; à cette étape, deux particules peuvent parfaitement occuper la même abscisse : il n'y a pas de conflit ;
3. remplacer chaque particule virtuelle par une particule réelle de rayon R en décalant toutes les particules (réelles et virtuelles) situées plus à droite de façon à dégager l'espace nécessaire.

Q 10. Écrire la fonction d'entête

```
def placement1Drapide(N:int, R:float, L:float) -> [np.ndarray]:
```

qui implante cet algorithme et renvoie la liste des coordonnées des centres de `N` particules de rayon `R` réparties aléatoirement le long d'un segment situé entre les abscisses 0 et `L`. On précise que l'ordre de la liste résultat n'est pas important.

Q 11. Quelle est la complexité de la fonction `placement1Drapide` ? Commenter.

I.C – Analyse statistique

Afin de vérifier que la fonction `placement1Drapide` produit une répartition de particules uniformément répartie sur le segment cible, on l'appelle un grand nombre de fois et on comptabilise pour chaque résultat obtenu la position initiale de chaque particule. Le résultat final est présenté sous forme d'un histogramme dont l'axe horizontal correspond à l'abscisse du centre de la particule dans l'intervalle $[0, L]$ et l'axe vertical au nombre total de particules placées à cette abscisse au cours des différentes exécutions de la fonction.

Q 12. Tracer et justifier l'allure des histogrammes pour $N = 1$, $N = 2$ et $N = 5$ dans le cas où $R = 1$ et $L = 10$.

I.D – Dimension quelconque

L'algorithme optimisé pour un segment, n'est pas utilisable pour des espaces de dimensions supérieures. Nous allons donc généraliser la fonction `placement1D` pour la transformer en une fonction utilisable dans un espace de dimension 1, 2 ou 3.

Q 13. En s'inspirant de la fonction `placement1D`, écrire la fonction d'entête

```
def placement(D:int, N:int, R:float, L:float) -> [np.ndarray]:
```

qui renvoie la liste des coordonnées des centres de `N` particules sphériques de rayon `R` placées aléatoirement dans un récipient de côté `L` dans un espace à `D` dimensions. Les modifications prévues aux questions 6 et 9 seront prises en compte dans cette fonction.

II Mouvement des particules

On suppose que l'on dispose désormais d'une fonction d'entête

```
def situationInitiale(D:int, N:int, R:float, L:float) -> [[np.ndarray, np.ndarray]]:
```

qui renvoie une liste de `N` particules de rayon `R`, représentées chacune par une liste à deux éléments (position et vitesse, cf. figure 1), placées aléatoirement à l'intérieur d'un récipient de taille `L` dans un espace à `D` dimensions. À partir de cette situation initiale, les positions et vitesses des particules vont évoluer au gré du déplacement des particules, des différents chocs entre elles et des rebonds sur les parois. On appelle *évènement* chaque choc ou rebond.

II.A – Analyse physique

Q 14. Comment évolue une particule entre deux évènements ?

Plaçons-nous dans un **espace à une dimension** et considérons deux particules de masses m_1 et m_2 qui entrent en collision avec les vitesses initiales $\vec{v}_1$ et $\vec{v}_2$. Les vitesses $\vec{v}_1{}'$ et $\vec{v}_2{}'$ des deux particules après le choc sont données par

$$\begin{cases} \vec{v}_1{}' = \dfrac{m_1 - m_2}{m_1 + m_2}\vec{v}_1 + \dfrac{2m_2}{m_1 + m_2}\vec{v}_2 \\ \vec{v}_2{}' = \dfrac{2m_1}{m_1 + m_2}\vec{v}_1 + \dfrac{m_2 - m_1}{m_1 + m_2}\vec{v}_2 \end{cases}$$

Q 15. Que deviennent ces formules lorsque $m_1 = m_2$? Commenter.

Q 16. Que deviennent ces formules lorsque $m_1 \ll m_2$? À quelle situation ce cas correspond-t-il dans le problème qui nous occupe ?

BY-NC-SA

II.B – Évolution des particules

Q 17. Écrire la fonction d'entête

```
def vol(p:[np.ndarray, np.ndarray], t:float) -> None:
```

qui met à jour l'état de la particule `p` (position et vitesse dans un espace de dimension quelconque) au bout d'un vol de `t` secondes sans choc ni rebond.

Q 18. Écrire la fonction d'entête

```
def rebond(p:[np.ndarray, np.ndarray], d:int) -> None:
```

qui met à jour la vitesse de la particule `p` suite à un rebond sur une paroi perpendiculaire à la dimension d'indice `d`, c'est-à-dire l'axe des abscisses si `d` vaut 0, l'axe des ordonnées si `d` vaut 1 et l'axe des cotes si `d` vaut 2.

Par généralisation du résultat obtenu dans un espace à une dimension, on supposera que le rebond d'une particule sur une paroi ne modifie pas la composante de la vitesse parallèle à la paroi et change le signe de sa composante normale à la paroi (rebond parfait). La fonction `rebond` n'est pas chargée de vérifier que la particule se trouve au contact d'une paroi.

Q 19. On revient dans un **espace à une dimension**. Écrire la fonction d'entête

```
def choc(p1:[np.ndarray, np.ndarray], p2:[np.ndarray, np.ndarray]) -> None:
```

qui modifie les vitesses des deux particules, `p1` et `p2`, suite au choc de l'une contre l'autre. La fonction `choc` n'est pas chargée de vérifier que les deux particules sont en contact.

On supposera dans la toute la suite que l'on dispose d'une version de la fonction `choc` également opérationnelle dans un espace à deux et trois dimensions.

III Inventaire des évènements

Chaque évènement sera représenté par une liste de cinq éléments avec la signification suivante :

0. un booléen indiquant si l'évènement est valide ou pas ;
1. un flottant donnant le nombre de secondes, à partir de l'instant courant, au bout duquel l'évènement aura lieu ;
2. un entier compris entre 0 et $N-1$ donnant l'indice dans la liste des N particules de la première (ou seule) particule concernée par l'évènement ;
3. un entier compris entre 0 et $N-1$ donnant l'indice de la deuxième particule concernée par l'évènement ou `None` s'il n'y a pas de deuxième particule concernée (l'évènement est un rebond sur une paroi) ;
4. un entier compris entre 0 et $D-1$ donnant l'indice de la dimension perpendiculaire à la paroi concernée par l'évènement ou `None` s'il n'y a pas de paroi concernée (l'évènement est un choc entre deux particules).

On supposera, sans avoir besoin de le vérifier, qu'on a toujours une et une seule valeur `None` parmi les deux derniers éléments de tout évènement.

Ainsi `[True, 0.4, 34, 57, None]` désigne le choc entre les particules d'indice 34 et 57 qui aura lieu dans 0,4 s. Et `[True, 1.7, 34, None, 1]` désigne le rebond de la particule d'indice 34 sur une paroi perpendiculaire à la dimension d'indice 1 (axe des ordonnées) qui aura lieu dans 1,7 s.

III.A – Prochains évènements dans un espace à une dimension

Q 20. Écrire, pour un **espace à une dimension**, la fonction d'entête

```
def tr(p:[np.ndarray, np.ndarray], R:float, L:float) -> None or (float, int):
```

qui détermine dans combien de temps la particule `p`, de rayon R, rencontrera une paroi du récipient de taille L, en faisant abstraction de toute autre particule qui pourrait se trouver sur son chemin. Cette fonction renvoie `None` si la particule ne rencontre jamais de paroi, sinon elle renvoie un couple dont le premier élément est la durée (en secondes) avant le rebond et le deuxième la direction de la paroi désignée par l'indice de sa dimension perpendiculaire.

Q 21. Toujours dans un **espace à une dimension**, écrire la fonction d'entête

```
def tc(p1:[np.ndarray, np.ndarray], p2:[np.ndarray, np.ndarray], R:float) -> None or float:
```

qui détermine si les deux particules `p1` et `p2`, de rayon R, vont se rencontrer, en faisant abstraction de la présence des autres particules et des parois, autrement dit en considérant que ces deux particules sont seules dans un espace infini. Cette fonction renvoie `None` si les deux particules ne se rencontrent jamais, sinon elle renvoie le temps (en secondes) au bout duquel les particules entrent en collision.

On supposera dans la toute la suite que l'on dispose d'une version des fonction `tr` et `tc` également opérationnelles dans un espace à deux et trois dimensions.

CC BY-NC-SA

III.B – Catalogue d'évènements

Afin d'alimenter l'algorithme de la partie suivante, on souhaite construire un catalogue des évènements qui pourraient se produire prochainement. Ce catalogue sera représenté par une liste dans laquelle les évènements, représentés par la liste de cinq éléments décrite au début de cette partie, sont ordonnés par date décroissante : le plus lointain en début de liste, le plus proche en fin de liste.

Q 22. Écrire la fonction d'entête

```
def ajoutEv(catalogue:[[bool, float, int, int or None, int or None]],
            e:[bool, float, int, int or None, int or None]) -> None:
```

qui ajoute au bon endroit dans la liste `catalogue` l'évènement `e`. La liste `catalogue` contient des évènements ordonnés par temps décroissant.

Q 23. Écrire la fonction d'entête

```
def ajout1p(catalogue:[[bool, float, int, int or None, int or None]], i:int,
            R:float, L:float, particules:[[np.ndarray, np.ndarray]]) -> None:
```

qui ajoute, dans la liste ordonnée d'évènements `catalogue`, les prochains évènements potentiels concernant la particule d'indice `i` de la liste `particules` qui contient toutes les particules présentes dans le récipient. Le paramètre `R` donne le rayon d'une particule et `L` la taille du récipient. Les évènements à prendre en compte sont le prochain rebond contre une paroi et le prochain choc avec chacune des autres particules (cf III.A). Les prochains évènements seront supposés valides et la fonction veillera à maintenir ordonnée la liste `catalogue`.

Q 24. Écrire la fonction d'entête

```
def initCat(particules:[[np.ndarray, np.ndarray]], R:float,
            L:float) -> [[bool, float, int, int or None, int or None]]:
```

qui utilise la fonction `ajout1p` et qui renvoie la liste, ordonnée par temps décroissant, des prochains évènements potentiels concernant une liste de particules `particules` de rayon `R` dans un récipient de taille `L`.

Q 25. Expliquer pourquoi la liste renvoyée par la fonction `initCat` contient certains éléments qui correspondent en fait au même évènement.

Q 26. Déterminer la complexité temporelle de la fonction `initCat` pour un espace à une dimension.

Q 27. Quelle est la fonction à optimiser en priorité afin d'améliorer la complexité de la fonction `initCat` ? Quel algorithme classique peut être utilisé pour optimiser cette fonction ?

IV Simulation

Nous disposons désormais des éléments de base pour simuler l'évolution d'un ensemble de particules identiques enfermées dans un récipient. En partant d'une situation initiale, nous pouvons déterminer les prochains évènements possibles, le plus proche de ces évènements va forcément avoir lieu. Nous pouvons alors établir un nouvel état de l'ensemble des particules juste après cet évènement, puis déterminer une nouvelle liste des prochains évènements possibles à partir de cette nouvelle situation. En répétant ce traitement, il est théoriquement possible de déterminer la position et la vitesse de chacune des particules à un instant quelconque dans le futur.

Q 28. Montrer que la liste des prochains évènements possibles ne peut jamais être vide, sauf si toutes les particules sont initialement à l'arrêt.

Dans toute la suite, nous considèrerons qu'au moins une particule est en mouvement.

Q 29. Écrire la fonction d'entête

```
def etape(particules:[[np.ndarray, np.ndarray]],
          e:[bool, float, int, int or None, int or None]) -> None:
```

qui, partant d'une liste de particules `particules` représentant la situation à l'instant courant, modifie l'état de chaque particule pour refléter la situation des particules juste après l'évènement `e` (supposé valide), en supposant qu'aucun autre évènement n'arrive avant celui-ci.

Disposant de la fonction `etape`, il suffirait de la combiner avec la fonction `initCat` pour implanter l'algorithme de simulation décrit plus haut. Cependant, étant donné la complexité de `initCat`, il semble intéressant d'optimiser cette phase de l'algorithme. Pour cela, remarquons que les évènements qui ne concernent par les particules impliquées dans l'évènement traité par la fonction `etape` restent valides, à un décalage temporel près. Les seuls nouveaux prochains évènements possibles concernent les particules impliquées dans l'évènement traité.

Q 30. Écrire la fonction d'entête

```
def majCat(catalogue:[[bool, float, int, int or None, int or None]],
           particules:[[np.ndarray, np.ndarray]],
           e:[bool, float, int, int or None, int or None], R:float, L:float) -> None:
```

qui met à jour son paramètre `catalogue`, liste ordonnée des prochains évènements potentiels, en supposant que `particules` représente la situation juste après l'évènement `e`, supposé valide et déjà retiré de `catalogue`. Les paramètres `R` et `L` désignent respectivement le rayon d'une particule et la taille du récipient. Afin de limiter les

CC BY-NC-SA

manipulations de listes, les évènements qui n'ont plus cours seront conservés dans le catalogue et simplement marqués non valides.

On dispose de la fonction d'entête

```
def enregistrer(bdd, t:float, e:[bool, float, int, int or None, int or None],
                particules:[[np.ndarray, np.ndarray]]) -> None:
```

qui enregistre dans la base de données `bdd` des informations à propos de l'évènement `e` survenu au temps `t` de la simulation. Le temps de la simulation est exprimé en secondes, le début de la simulation étant pris comme origine. Le paramètre `particules` donne la situation (position, vitesse) des particules au temps `t` considéré, juste après la survenue de l'évènement `e`.

Q 31. Écrire la fonction d'entête

```
def simulation(bdd, d:int, N:int, R:float, L:float, T:float) -> int:
```

qui simule l'évolution de `N` particules identiques de rayon `R` dans un récipient de côté `L` dans un espace à `d` dimensions pendant la durée `T` (exprimée en secondes). Cette fonction utilise une situation initiale générée aléatoirement par l'intermédiaire de la fonction `situationInitiale` (partie II) et renvoie le nombre d'évènements ayant eu lieu pendant toute la simulation. D'autre part, elle enregistre chaque évènement dans la base de données `bdd`.

Q 32. Comment sont gérés les doublons repérés à la question 25 ?

Q 33. Dans la représentation choisie pour les évènements, le temps auquel cet évènement peut survenir est donné par rapport à un instant courant (qui correspond à l'instant de l'évènement précédent dans l'implantation choisie) ce qui oblige à recaler chaque évènement au fur et à mesure que le temps de la simulation s'écoule. Une autre possibilité aurait été d'indiquer le temps de chaque évènement par rapport à une référence fixe (le début de la simulation). Discuter des avantages et des inconvénients de chaque représentation en terme de précision du résultat et de complexité de l'algorithme. La représentation retenue ici est-elle la mieux adaptée des deux pour traiter le problème posé ?

V Exploitation des résultats

On dispose d'une version plus générale de la fonction `simulation` pour laquelle toutes les particules ne sont plus nécessairement identiques. Cette fonction enregistre ses résultats dans une base de données dont la structure est donnée figure 4.

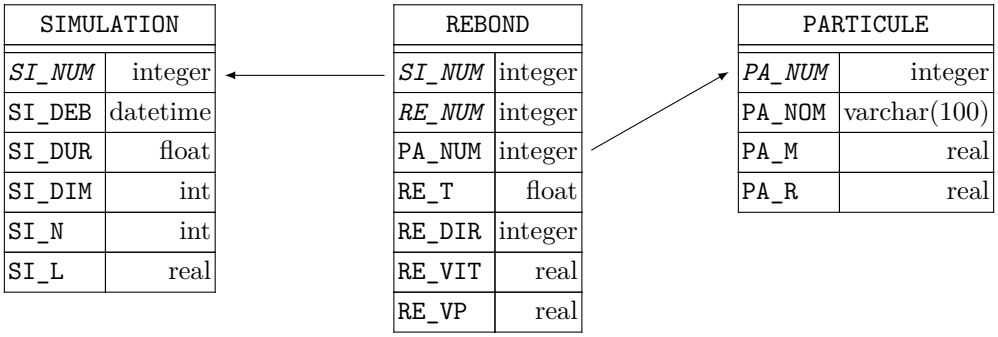

Figure 4 Structure physique de la base de données des résultats de simulation

Cette base comporte les trois tables suivantes :

— la table `SIMULATION`, de clef primaire `SI_NUM`, donne les caractéristiques de chaque simulation effectuée. Elle contient les colonnes
- `SI_NUM` numéro d'ordre de la simulation (clef primaire)
- `SI_DEB` date et heure du lancement du programme de simulation
- `SI_DUR` durée (en secondes) de la simulation (il ne s'agit pas du temps d'exécution du programme, mais du temps simulé)
- `SI_DIM` nombre de dimensions de l'espace de simulation
- `SI_N` nombre de particules pour cette simulation
- `SI_L` (en mètres) taille du récipient utilisé pour la simulation

— la table `PARTICULE`, de clef primaire `PA_NUM`, des types de particules considérées. Elle contient les colonnes
- `PA_NUM` numéro (entier) identifiant le type de particule (clef primaire)
- `PA_NOM` nom de ce type de particule
- `PA_M` masse de la particule (en grammes)
- `PA_R` rayon (en mètres) de la particule

BY-NC-SA

— la table `REBOND`, de clef primaire `(SI_NUM, RE_NUM)`, liste les chocs des particules avec les parois du récipient. Elle contient les colonnes

- `SI_NUM` numéro d'ordre de la simulation ayant généré ce rebond
- `RE_NUM` numéro d'ordre du rebond au sein de cette simulation
- `PA_NUM` numéro du type de particule concernée par ce rebond
- `RE_T` temps de simulation (en secondes) auquel ce rebond est arrivé
- `RE_DIR` paroi concernée : entier non nul de l'intervalle `[-SI_DIM, SI_DIM]` donnant la direction de la normale à la paroi. Ainsi -2 désigne la paroi située en $y = 0$ alors que 1 désigne la paroi située en $x = L$
- `RE_VIT` norme de la vitesse de la particule qui rebondit (en $\mathrm{m{\cdot}s^{-1}}$)
- `RE_VP` valeur absolue de la composante de la vitesse normale à la paroi (en $\mathrm{m{\cdot}s^{-1}}$)

Q 34. Écrire une requête SQL qui donne le nombre de simulations effectuées pour chaque nombre de dimensions de l'espace de simulation.

Q 35. Écrire une requête SQL qui donne, pour chaque simulation, le nombre de rebonds enregistrés et la vitesse moyenne des particules qui frappent une paroi.

Q 36. Écrire une requête SQL qui, pour une simulation n donnée, calcule, pour chaque paroi, la variation de quantité de mouvement due aux chocs des particules sur cette paroi tout au long de la simulation. On se rappellera que lors du rebond d'une particule sur une paroi la composante de sa vitesse normale à la paroi est inversée, ce qui correspond à une variation de quantité de mouvement de $2m|v_\perp|$ où m désigne la masse de la particule et $v_\perp$ la composante de sa vitesse normale à la paroi.

Opérations et fonctions Python disponibles

Fonctions

— `range(n)` renvoie la séquence des `n` premiers entiers ($0 \to n-1$)

— `list(range(n))` renvoie une liste contenant les `n` premiers entiers dans l'ordre croissant :
`list(range(5))` → `[0, 1, 2, 3, 4]`

— `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers)

— `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme

— `random.shuffle(u)` permute aléatoirement les éléments de la liste `u` (modifie `u`)

— `random.sample(u, n)` renvoie une liste de `n` éléments distincts de la liste `u` choisis aléatoirement, si `n > len(u)`, déclenche l'exception `ValueError`

— `math.sqrt(x)` calcule la racine carrée du nombre x

— `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à `x`

— `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`

— `sorted(u)` renvoie une nouvelle liste contenant les éléments de la liste `u` triés dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les listes

— `len(u)` donne le nombre d'éléments de la liste `u` :
`len([1, 2, 3])` → 3 ; `len([[1,2], [3,4]])` → 2

— `u + v` construit une liste constituée de la concaténation des listes `u` et `v` :
`[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`

— `n * u` construit une liste constitué de la liste `u` concaténée `n` fois avec elle-même :
`3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`

— `e in u` et `e not in u` déterminent si l'objet `e` figure dans la liste `u`
`2 in [1, 2, 3]` → `True` ; `2 not in [1, 2, 3]` → `False`

— `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (similaire à `u = u + [e]`)

— `u.pop()` renvoie le dernier élément de la liste `u` (`u[-1]`) et le supprime (`del u[-1]`)

— `del u[i]` supprime de la liste `u` son élément d'indice `i`

— `del u[i:j]` supprime de la liste `u` tous ses éléments dont les indices sont compris dans l'intervalle `[i, j[`

— `u.remove(e)` supprime de la liste `u` le premier élément qui a pour valeur `e`, déclenche l'exception `ValueError` si `e` ne figure pas dans `u`

— `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste

CC BY-NC-SA

— `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`

— `u.sort()` trie la liste `u` en place, dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les tableaux (`np.ndarray`)

— `np.array(u)` crée un nouveau tableau contenant les éléments de la liste `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`

— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou une matrice à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique numpy (`np.int16`, `np.float32`, ...). Si le paramètres `dtype` n'est pas précisé, les éléments seront de type `float`

— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens

— `np.random.rand(n)`, `np.random.rand(n, m)` crée un tableau de la forme indiquée (`n` lignes, `m` colonnes) en initialisant chaque élément avec une valeur aléatoire issue d'une distribution uniforme sur $[0, 1[$

— `a.ndim` nombre de dimensions du tableau `a` (1 pour un vecteur, 2 pour une matrice, etc.)

— `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions

— `len(a)` taille du tableau `a` dans sa première dimension (nombre d'éléments d'un vecteur, nombre de lignes d'une matrice, etc.) équivalent à `a.shape[0]`

— `a.size` nombre total d'éléments du tableau `a`

— `a.flat` itérateur sur tous les éléments du tableau `a`

— `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O($`a.size`$)$

— `b in a` détermine si `b` est un élément du tableau `a` ; si `b` est un scalaire, vérifie si `b` est un élément de `a` ; si `b` est un vecteur ou une liste et `a` une matrice, détermine si `b` est une ligne de `a`

— `np.concatenate((a1, a2))` construit un nouveau tableau en concaténant deux tableaux ; `a1` et `a2` doivent avoir le même nombre de dimensions et la même taille à l'exception de leur taille dans la première dimension (deux matrices doivent avoir le même nombre de colonnes pour pouvoir être concaténées)

— `a.sort(d)` trie le tableau `a` en place suivant sa dimension d'indice `d` (par défaut, la dernière du tableau) : `a.sort(0)` trie les éléments du vecteur `a` ou les lignes de la matrice `a` ; `a.sort(1)` trie les colonnes de la matrice `a`

— `np.sort(a, d)` renvoie une copie triée du tableau `a` suivant sa dimension d'indice `d` (voir `a.sort(d)` pour la signification exacte du paramètre `d`)

• • • FIN • • •

CC BY-NC-SA

Concours Centrale-Supélec 2018 filière PC

Informatique

Présentation du sujet

Le sujet porte sur la simulation de la cinétique d'un gaz parfait à l'échelle microscopique. Les molécules sont assimilées à des sphères et les interactions entre molécules et avec les parois sont modélisées par des chocs élastiques, à l'image du modèle de Boltzmann. L'ensemble des étapes de simulation est abordé, de l'initialisation à l'extraction des résultats, sur des cas à une ou plusieurs dimensions.

La première partie aborde la problématique d'initialisation aléatoire des positions et vitesses des molécules, en évitant toute interférence entre molécules et avec le récipient. Divers algorithmes sont envisagés et évalués, sur des modèles à une ou plusieurs dimensions.

La seconde partie porte sur la simulation du mouvement et des interactions, conduisant à l'écriture de plusieurs fonctions réemployées par la suite.

La troisième partie porte sur la manipulation d'une structure de données contenant les évènements (les chocs) possibles recensés et ordonnés.

La quatrième partie développe le cœur de l'algorithme de simulation en structurant les successions d'évènements et le déroulement temporel, tout en réemployant les fonctions précédemment écrites.

La cinquième partie clôt l'étude en proposant d'élaborer des requêtes SQL d'extraction de grandeurs physiques dans la base de données ayant stocké les résultats de simulation.

Analyse globale des résultats

Le sujet est de longueur raisonnable pour le temps imparti : 20 % des candidats ont abordé plus de 80 % des questions tandis que la moitié des candidats a abordé au moins 2/3 des questions.

À nouveau cette année, le jury se réjouit du niveau satisfaisant des copies. Le langage est bien maitrisé et permet de traduire les solutions aux questions sans difficulté. Quelques rares candidats ont visiblement négligé la formation en informatique et se contentent de répondre aux questions ne relevant pas immédiatement d'informatique. Ces copies conduisent à des notes très faibles.

Les petites erreurs syntaxiques n'ont pas été retenues par le jury comme un élément discriminatoire, dans la mesure où elles ne cachent pas des erreurs de fond. Les réponses pertinentes d'un point de vue algorithmique sont valorisées.

Certaines copies proposent des programmes particulièrement élégants et concis, et reflètent un vrai recul sur les différentes stratégies de programmation. Ces copies ont été valorisées.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Au regard des copies évaluées, le jury conseille aux futurs candidats de prêter attention aux remarques suivantes.

L'indentation en python délimite les blocs d'instructions et doit apparaitre clairement dans la rédaction. Toute rédaction claire est bienvenue ; bien souvent, un trait vertical marquant l'alignement du bloc d'instruction est suffisant.

L'initialisation d'une variable dans une boucle ou hors de la boucle n'a pas les mêmes conséquences pour l'algorithme.

Le nombre d'itérations d'une boucle doit être bien réfléchi pour s'assurer que les indices des éléments d'une liste appelée dans la boucle sont bien définis. L'instruction `range(n)` produit `n` entiers compris entre `0` et `n-1` et permet dont d'effectuer `n` itérations indicées de `0` à `n-1`.

Les opérateurs booléens `and` et `or` fonctionnent séquentiellement en arrêtant l'évaluation dès que la valeur logique est établie, ils ne sont donc pas commutatifs. Ainsi, l'ordre des conditions dans une instruction `while` a souvent de l'importance.

Lorsque le sujet précise explicitement les paramètres des fonctions et les valeurs renvoyées, ainsi que leur type, il convient de veiller à les respecter. Certaines variables en argument peuvent être modifiées par une fonction sans nécessairement être renvoyées.

Beaucoup de questions sont indépendantes et généralement le prototype d'une fonction, donné dans une question, permet de l'utiliser dans les questions suivantes.

Les opérateurs classiques (+, *, etc.) n'ont pas toujours le même sens selon les types des opérandes (en particulier pour les listes et les tableaux numpy).

La concision et l'élégance des programmes sont appréciées dans l'évaluation. Les candidats qui réinvestissent les fonctions déjà codées sont valorisés par rapport à ceux qui recopient les lignes de code équivalentes.

Des listes de conditions en cascade nuisent à la lisibilité de l'algorithme. Une condition booléenne bien choisie distingue les candidats dont la pensée est claire.

Des noms de variables explicites aident à la compréhension du code. De trop nombreux candidats utilisent des noms de variables non significatifs (a, b, c ...) ce qui nuit à la compréhension du programme. La clarté du programme (en particulier le choix des noms de variables) ainsi que la présence de commentaires opportuns sont prises en compte dans l'évaluation.

Lors d'un calcul de complexité, une justification minimale est attendue.

L'ordre des questions importe. Prendre soin de rédiger les réponses aux questions en respectant leur ordre dans le sujet.

La qualité d'expression (l'orthographe notamment) et la qualité visuelle de présentation relèvent des compétences de communication indispensables à un candidat à une école d'ingénieurs. Le correcteur n'attribue les points qu'aux éléments de réponse qu'il parvient à lire et à comprendre. Les copies obscures et difficiles à comprendre sont pénalisées.

Les variables utilisées dans une fonction doivent être définies dans cette fonction ou être explicitement définies comme variables globales (soit par le sujet, soit par le candidat). Les candidats sont invités à bien lire l'annexe contenant certaines fonctions utiles pour traiter le sujet.

I Initialisation

La première partie vise à évaluer plusieurs stratégies d'initialisation de la simulation, d'abord en une dimension, puis en trois dimensions.

Les questions 1 à 5 sont très bien réussies. Quelques rares copies ont confondu l'opérateur * pour les tableaux numpy et ce même opérateur pour les listes (question 1). Quelques candidats détaillent en français les opérations élémentaires réalisées (multiplication, tirage aléatoire, etc.) alors qu'il était attendu une explication du sens de ces lignes pour le problème posé. Un schéma était souvent plus

clair qu'un long discours pour les questions 3 et 4. Étrangement, les questions 6 (tirer un nombre aléatoire entre deux bornes) et 7 (comportement d'un algorithme) sont moins bien abordées.

La question 10 demande d'écrire un algorithme décrit dans l'énoncé. Le jury a accepté tout algorithme qui permettait de répartir sans interférence les boules, même lorsque la répartition n'est pas équiprobable. Beaucoup de propositions partent du principe que la liste est triée après le tirage aléatoire. Un certain nombre d'algorithmes proposés s'éloigne notablement des consignes du sujet. La question 12 est souvent correctement traitée pour N=1, beaucoup plus rarement pour N=5 presque jamais pour N=2.

La question 13 permettait d'envisager une reformulation du code donné pour une simulation en trois dimensions. De façon inattendue, la norme euclidienne a posé beaucoup de difficultés.

II Mouvement des particules

La deuxième partie s'attache à décrire le mouvement des particules, d'abord par les lois physiques (données), puis en développant les fonctions associées à chaque phénomène (vol libre, rebond sur la paroi ou choc entre particules).

L'analyse des lois physiques a posé peu de difficultés aux candidats, bien que le jury constate que la notion de mouvement rectiligne uniforme ne soit pas toujours claire. Les fonctions à écrire étaient très simples, mais ont permis de distinguer les candidats qui proposent une formulation élégante en une ligne et d'autres dont le code est laborieux. Le sujet demandait une fonction qui mette à jour l'état de la particule par effet de bord et renvoie `None` ; certains candidats renvoient la particule.

III Inventaire des évènements

La troisième partie permet d'élaborer une structure de données mémorisant le catalogue des évènements anticipés (chocs et rebonds) et de définir les fonctions de manipulation de cette structure (initialisation, ajout d'évènements).

Beaucoup de candidats ont eu des difficultés à traduire informatiquement les conditions booléennes de collision entre particules et paroi pour les fonctions `tr` et `tc`. Certains n'utilisent qu'une seule structure conditionnelle `if` judicieusement choisie tandis que d'autres en utilisent 6 à 8, le code devenant très difficile à lire.

L'ajout d'un évènement nécessitait d'insérer un élément dans une liste triée. Certains oublient d'envisager le cas où l'élément à insérer se place au début ou à la fin de la liste.

L'ajout de tous les évènements relatifs à une particule est relativement bien abordé, mais beaucoup de candidats oublient de vérifier que les fonctions `tr` et `tc` ne renvoient pas None avant d'ajouter le résultat au catalogue. Certains ont cherché à sélectionner l'évènement le plus proche, contrairement à ce que demande le sujet.

L'initialisation du catalogue n'a pas posé de difficulté, mais le calcul de complexité nécessitait de remarquer que la taille du catalogue n'est pas en $O(N)$, ce que très peu de candidats ont vu. Un calcul de complexité ne se limite pas toujours à compter les boucles `for` imbriquées.

IV Simulation

Seuls les meilleurs candidats ont abordé correctement cette partie, qui vise à mettre en œuvre les étapes de simulation.

La fonction `etape` permet de tenir compte du déplacement rectiligne uniforme des particules entre deux évènements (vol), et de traiter l'évènement (le rebond ou le choc). Certains candidats oublient la phase de vol, soit pour toutes les particules, soit pour les particules subissant l'évènement.

La mise à jour du catalogue est souvent incomplète, en particulier pour la gestion des dates. De même, la gestion correcte des évènements invalides et du temps dans la simulation est rarement juste.

V Exploitation des résultats

La cinquième partie demande d'élaborer trois requêtes SQL d'extraction de résultats. Bien qu'elles soient en fin de sujet, ces questions sont la plupart du temps abordées, avec des résultats plutôt corrects. L'instruction GROUP BY, relative aux fonctions d'agrégation, n'est pas toujours connue. Certaines requêtes utilisent des jointures inutiles.

Conclusion

Le sujet aborde une large partie du programme d'informatique commune. Le choix d'un sujet s'appuyant la simulation d'un phénomène physique par une approche numérique, impliquant une part d'algorithmique, assure une cohérence avec la formation d'ingénieurs. Cette approche sera reconduite sur des problématiques de simulation ou d'algorithmique en informatique, à partir du programme des trois semestres d'informatique.

Les bons résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences affirmées en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

Informatique

MP, PC, PSI, TSI

CONCOURS CENTRALE•SUPÉLEC

3 heures Calculatrices autorisées

2017

Mars Exploration Rovers
Mission d'exploration martienne

Mars Exploration Rovers (MER) est une mission de la NASA qui cherche à étudier le rôle joué par l'eau dans l'histoire de la planète Mars. Deux robots géologues, Spirit et Opportunity (figure 1), se sont posés sur cette planète, sur deux sites opposés, en janvier 2004. Leur mission est de rechercher et d'analyser différents types de roches et de sols qui peuvent contenir des indices sur la présence d'eau. Ils sont équipés de six roues et d'une suspension spécialement conçue pour leur permettre de se déplacer quelle que soit la nature du terrain rencontré. Leur cahier des charges prévoyait une durée de vie de 90 jours martiens (le jour martien est environ 40 minutes plus long que le jour terrestre). Spirit a cessé d'émettre le 22 mars 2010, soit 2210 jours martiens après son arrivée sur la planète. Début 2017, Opportunity est toujours en activité et il a parcouru plus de 44 km sur Mars.

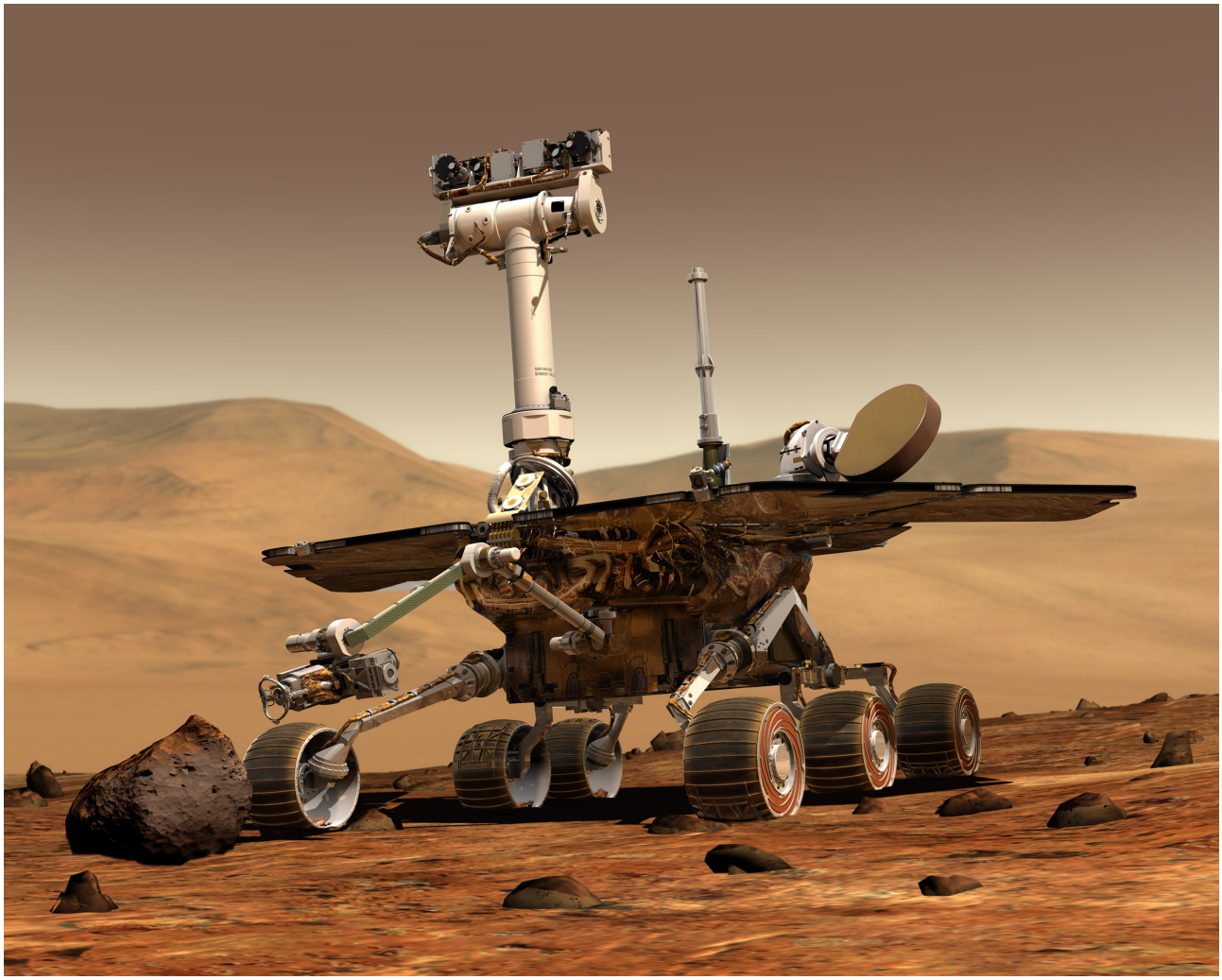

Figure 1 Vue d'artiste d'un robot géologue de la mission *Mars Exploration Rovers* (NASA/JPL – Caltech/Cornell)

Chaque robot est équipé de plusieurs instruments d'analyse (caméra, microscope, spectromètres) et d'un bras qui permet d'amener les instruments au plus près des roches et sols dignes d'intérêt. À partir de photographies de la surface de la planète, prises à plusieurs longueurs d'ondes par différents satellites et par le robot lui-même, les scientifiques de la NASA définissent une liste d'emplacements (*points d'intérêt* ou PI) où effectuer des analyses. Cette liste est transmise au robot qui doit se rendre à chaque emplacement indiqué et y effectuer les analyses prévues. Chaque robot est capable d'effectuer un certain nombre de types d'analyses géologiques correspondant aux différents instruments dont il dispose. Une fois tous les points d'intérêts visités et les résultats des analyses

CC BY-NC-SA

transmis à la Terre, le robot reçoit une nouvelle liste de points d'intérêts et démarre une nouvelle *exploration*. Compte-tenu des contraintes de transmission entre la Terre et les robots (latence, périodes d'ombre, faible débit, etc.) il est prévu que les robots travaillent en autonomie pour planifier le parcours de chaque exploration. Ainsi, une fois la liste des points d'intérêt reçue, le robot analyse le terrain afin de détecter d'éventuels obstacles et détermine le meilleur chemin lui permettant de visiter l'ensemble de ces points en dépensant le moins d'énergie possible.

Après s'être intéressé à l'enregistrement des explorations, des points d'intérêts correspondants et des analyses à y mener, ce sujet aborde trois algorithmes qui peuvent être utilisés par le robot pour déterminer le meilleur parcours lui permettant de visiter chaque point d'intérêt une et une seule fois. Pour cela nous faisons quelques hypothèses simplificatrices.

— La zone d'exploration est dépourvue d'obstacle : le robot peut rejoindre directement en ligne droite n'importe quel point d'intérêt.

— Le sol est horizontal et de nature constante : l'énergie utilisé pour se déplacer entre deux points ne dépend que de leur distance, autrement dit le meilleur chemin est le plus court.

— La courbure de la planète est négligée compte tenu de la dimension réduite de la zone d'exploration : nous travaillerons en géométrie euclidienne et les points d'intérêts seront repérés par leurs coordonnées cartésiennes à l'intérieur de la zone d'exploration.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Toutes les questions sont indépendantes. Néanmoins, il est possible de faire appel à des fonctions ou procédures créées dans d'autres questions. Dans tout le sujet on suppose que les bibliothèques `math`, `numpy` et `random` ont été importées grâce aux instructions

```
import math
import numpy as np
import random
```

Si les candidats font appel à des fonctions d'autres bibliothèques ils doivent préciser les instructions d'importation correspondantes.

Ce sujet utilise la syntaxe des annotations pour préciser le types des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, x:float, d:str) -> np.ndarray:
```

signifie que la fonction `maFonction` prend trois arguments, le premier est un entier, le deuxième un nombre à virgule flottante et le troisième une chaine de caractères et qu'elle renvoie un tableau numpy.

Une liste de fonctions utiles est donnée à la fin du sujet.

I Création d'une exploration et gestion des points d'intérêt

Une exploration est un ensemble de points d'intérêts à l'intérieur d'une zone géographique limitée, une série d'analyses étant associée à chaque point d'intérêt. Chaque type d'analyse que le robot peut effectuer est codifié et référencé par un nombre entier. Un point d'intérêt est repéré par deux entiers, positifs ou nuls, correspondant à ses coordonnées cartésiennes en millimètres à l'intérieur de la zone d'exploration. L'ensemble des points d'intérêts d'une exploration qui en contient n est représenté par un objet de type `numpy.ndarray`, à éléments entiers, à 2 colonnes et n lignes, l'élément d'indice `i,0` correspondant à l'abscisse du point d'intérêt i et l'élément d'indice `i,1` à son ordonnée.

	x	y
0	345	635
1	1076	415
2	38	859
3	121	582

Figure 2 Exemple d'exploration avec quatre points d'intérêt

I.A – Génération d'une exploration d'essai

I.A.1) Choix de points au hasard

a) Afin de disposer de données pour tester les différents algorithmes de calcul de chemin qui seront développés plus tard, écrire une fonction qui construit une exploration au hasard. Cette fonction d'entête

```
def générer_PI(n:int, cmax:int) -> np.ndarray:
```


CC BY-NC-SA

prend en paramètres le nombre de points d'intérêts à générer et la largeur de la zone d'exploration (supposée carrée) et renvoie un objet de type `numpy.ndarray` contenant les coordonnées de n points **deux à deux distincts** choisis au hasard dans la zone d'exploration (figure 2).

b) Quelles contraintes doivent vérifier les arguments de la fonction `générer_PI` ?

I.A.2) Calcul des distances

On dispose de la fonction d'entête

```
def position_robot() -> tuple:
```

qui renvoie un couple donnant les coordonnées actuelles du robot dans le système de coordonnées de l'exploration à planifier. Ainsi l'instruction `x, y = position_robot()` permet de récupérer les coordonnées courantes du robot.

Afin de faciliter l'application des différents algorithmes de recherche de chemin, on souhaite construire un tableau des distances entre les différents points d'intérêt d'une exploration et entre ceux-ci et la position courante du robot au moment du calcul. Écrire une fonction d'entête

```
def calculer_distances(PI:np.ndarray) -> np.ndarray:
```

qui prend en paramètre un tableau de n points d'intérêt tel que décrit précédemment et renvoie un tableau de nombres flottants, de dimension $(n+1) \times (n+1)$, tel que l'élément d'indice `i,j` fournit la distance entre les points d'intérêt i et j, l'indice `n` désignant le point de départ du robot.

I.B – Traitement d'image

On dispose de photographies d'une zone d'exploration effectuées à différentes longueur d'onde. Chaque photographie a été mises à l'échelle de la zone d'exploration puis stockée dans un tableau numpy (`np.ndarray`) à deux dimensions. Les dimensions correspondent aux coordonnées géographiques du point photographié, chaque élément est un entier, compris entre 0 et 255, donnant l'intensité du point considéré sur l'image. Ainsi l'élément d'indice `x,y` contient un entier, compris entre 0 et 255, correspondant à l'intensité sur la photographie considérée du point de coordonnées (x, y). À partir de ces photographies, les géologues déterminent les endroits à analyser en filtrant ceux qui ont un profil d'émission caractéristique de certaines roches intéressantes.

I.B.1) Analyse d'une image

La fonction `F1` ci-dessous prend en paramètre une photographie représentée comme décrit plus haut. Expliquer ce que fait cette fonction et décrire son résultat.

```
1   def F1(photo:np.ndarray) -> np.ndarray:
2       n = photo.min()
3       b = photo.max()
4       h = np.zeros(b - n + 1, np.int64)
5       for p in photo.flat:
6           h[p - n] += 1
7       return h
```

I.B.2) Sélection de points d'intérêts

Écrire une fonction d'entête

```
def sélectionner_PI(photo:np.ndarray, imin:int, imax:int) -> np.ndarray:
```

où `photo` est un tableau représentant une photographie. Le résultat de la fonction `sélectionner_PI` est un tableau à deux dimensions, de structure similaire à celui décrit figure 2, contenant les coordonnées des points dont l'intensité sur la photographie est comprise entre `imin` et `imax`.

I.C – Base de données

Afin d'assurer son autonomie opérationnelle, le robot dispose localement des informations nécessaires à son fonctionnement quotidien. Ainsi, il enregistre la durée d'utilisation de ses différents instruments embarqués. Il connait également les différents types d'analyses qu'il peut effectuer et, pour chacun de ces types, les instruments à utiliser. Il enregistre la prochaine exploration, c'est-à-dire les différents points d'intérêts qu'il doit visiter et pour chacun la ou les analyses qu'il doit effectuer. D'autre part, il conserve les résultats d'analyses effectuées lors de ses explorations passées. Ces résultats ne sont effacés qu'après confirmation de leur bonne transmission sur Terre.

Ces différentes informations sont stockées dans une base de données relationnelle dont le modèle physique est schématisé figure 3.

BY-NC-SA

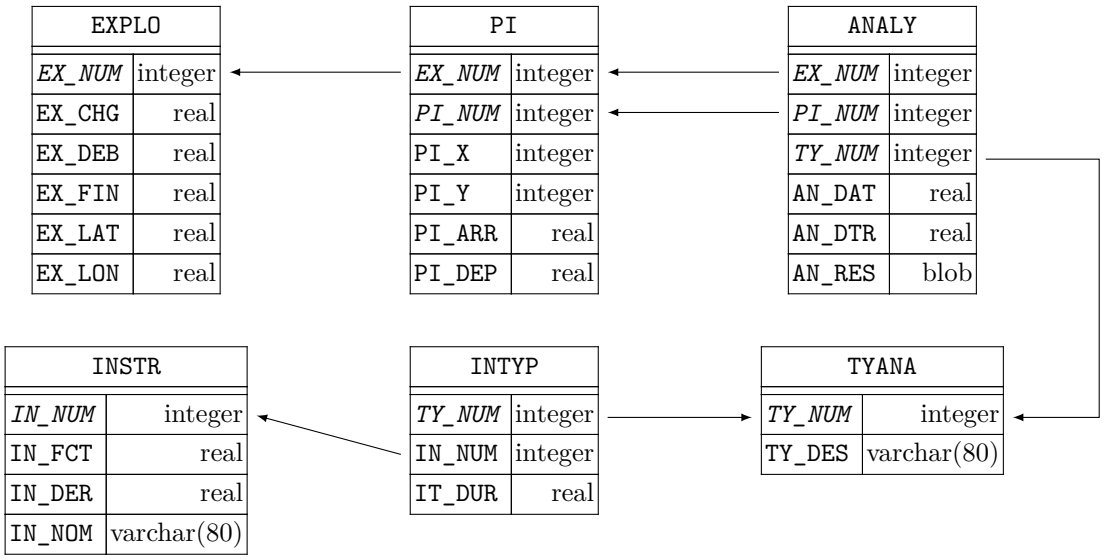

Figure 3 Structure physique de la base de données d'un robot

Cette base comporte les six tables suivantes :

— la table `EXPLO` des explorations, avec les colonnes
  - `EX_NUM` numéro (entier) de l'exploration (clef primaire)
  - `EX_CHG` date de transmission des points d'intérêts de cette exploration
  - `EX_DEB` date de début de l'exploration (`NULL` si l'exploration n'est pas encore commencée)
  - `EX_FIN` date de fin de l'exploration (`NULL` si l'exploration n'est pas encore terminée)
  - `EX_LAT` latitude (en degrés décimaux) du point de coordonnées $(0, 0)$ de la zone d'exploration
  - `EX_LON` longitude (en degrés décimaux) du point de coordonnées $(0, 0)$ de la zone d'exploration

— la table `PI` des points d'intérêts, de clef primaire `(EX_NUM, PI_NUM)`, avec les colonnes
  - `EX_NUM` numéro de l'exploration à laquelle appartient le point d'intérêt
  - `PI_NUM` numéro du point d'intérêt dans l'exploration (au sein d'une exploration les PI sont numérotés en séquence en commençant à 0, ce numéro n'a pas de rapport avec l'ordre dans lequel les PI sont explorés par le robot)
  - `PI_X` l'abscisse du point d'intérêt dans la zone d'exploration (entier positif en millimètres)
  - `PI_Y` l'ordonnée du point d'intérêt dans la zone d'exploration (entier positif en millimètres)
  - `PI_ARR` date d'arrivée du robot au point d'intérêt (`NULL` si ce point n'a pas encore été visité)
  - `PI_DEP` date à laquelle le robot a quitté le point d'intérêt (`NULL` si ce point n'a pas encore été exploré ou si la visite est en cours)

— la table `INSTR` des instruments embarqués, avec les colonnes
  - `IN_NUM` le numéro (entier) de l'instrument (clef primaire)
  - `IN_FCT` la durée pendant lequel l'instrument a déjà été utilisé depuis l'arrivée sur la planète (nombre décimal : fraction de jour martien)
  - `IN_DER` la date de la dernière utilisation de l'instrument
  - `IN_NOM` nom de l'instrument

— la table `TYANA` des types d'analyses à effectuer, avec les colonnes
  - `TY_NUM` le numéro de référence (entier) du type d'analyse (clef primaire)
  - `TY_DES` le nom du type d'analyse

— la table `INTYP` des instruments utilisés pour un type d'analyse, de clef primaire `(TY_NUM, IN_NUM)`, avec les colonnes
  - `TY_NUM` le numéro de référence (entier) du type d'analyse
  - `IN_NUM` le numéro (entier) de l'instrument
  - `IT_DUR` la durée standard d'utilisation de l'instrument dans ce type d'analyse (nombre décimal : fraction de jour martien)

— la table `ANALY` indiquant pour chaque point d'intérêt les types d'analyses à effectuer ou effectuées, de clef primaire `(EX_NUM, PI_NUM, TY_NUM)` et avec les colonnes
  - `EX_NUM` numéro de l'exploration à laquelle appartient le point d'intérêt
  - `PI_NUM` numéro du point d'intérêt dans l'exploration
  - `TY_NUM` type de l'analyse

BY-NC-SA

- `AN_DAT` date de l'analyse (`NULL` si l'analyse n'a pas été effectuée)
- `AN_DTR` date de transmission sur Terre des résultats de l'analyse (`NULL` si l'analyse n'a pas été transmise)
- `AN_RES` résultat de l'analyse (donnée opaque dont la signification dépend du type d'analyse)

Toutes les dates sont stockées sous forme d'un nombre décimal correspondant au nombre de jours martiens depuis l'arrivée du robot sur la planète.

I.C.1) Écrire une requête SQL qui donne le numéro de l'exploration en cours, s'il y en a une.

I.C.2) Écrire une requête SQL qui donne, pour une exploration dont on connait le numéro, la liste des points d'intérêts de cette exploration avec leurs coordonnées.

I.C.3) Écrire une requête SQL qui donne la surface, en mètres carrés, de chaque zone déjà explorée par le robot. La zone d'exploration est définie comme le plus petit rectangle qui englobe l'ensemble des points d'intérêts de l'exploration et dont les bords sont parallèles aux axes de référence (axes des abscisse et des ordonnées).

I.C.4) Quelle est la surface maximale d'une zone d'exploration que peut stocker cette base de données ?

I.C.5) Écrire une requête SQL qui donne, pour l'exploration en cours, le nombre de fois où chaque instrument doit être utilisé et sa durée d'utilisation théorique (en jours martiens) pour la totalité de l'exploration.

II Planification d'une exploration : première approche

Avant de démarrer une nouvelle exploration, le robot doit déterminer un chemin qui lui permet de passer par tous les points d'intérêts une et une seule fois. L'enjeu de l'opération est de trouver le chemin le plus court possible afin de limiter la dépense d'énergie et de limiter l'usure du robot.

Chaque point d'intérêt sera repéré par un entier positif ou nul correspondant à son indice dans le tableau des points d'intérêt. Un chemin d'exploration sera représenté par un objet de type `list` donnant les indices des points d'intérêt dans l'ordre de leur parcours.

II.A – Quelques fonctions utilitaires

II.A.1) Longueur d'un chemin

Écrire une fonction d'entête

```
def longueur_chemin(chemin:list, d:np.ndarray) -> float:
```

qui prend en paramètre un chemin à parcourir et la matrice des distances entre points d'intérêt (telle que renvoyée par la fonction `calculer_distances`) et renvoie la distance que doit effectuer le robot pour suivre ce chemin en partant de sa position courante (correspondant à la dernière ligne/colonne du tableau `d`) et en visitant tous les points d'intérêt dans l'ordre indiqué.

II.A.2) Normalisation d'un chemin

Écrire une fonction d'entête

```
def normaliser_chemin(chemin:list, n:int) -> list:
```

qui prend en paramètre une liste d'entiers et renvoie une liste correspondant à un chemin valide, c'est-à-dire contenant une seule fois tous les entiers entre 0 et `n` (exclu). Pour cela cette fonction commence par supprimer les éventuels doublons (en ne conservant que la première occurrence) et les valeurs supérieures ou égales à `n`, sans modifier l'ordre relatif des éléments conservés, puis ajoute à la fin les éventuels éléments manquants en ordre croissant de numéros.

II.B – Force brute

Pour rechercher le plus court chemin, on peut imaginer de considérer tous les chemins possibles et de calculer leur longueur. On obtiendra ainsi à coup sûr le chemin le plus court.

II.B.1) Déterminer en fonction de n, nombre de points à visiter, le nombre de chemins possibles passant exactement une fois par chacun des points.

II.B.2) Cet algorithme est-il utilisable pour une zone d'exploration contenant 20 points d'intérêts ? Justifier.

II.C – Algorithme du plus proche voisin

Une idée simple pour obtenir un algorithme utilisable est de construire un chemin en choisissant systématiquement le point, non encore visité, le plus proche de la position courante.

II.C.1) Écrire une fonction d'entête

```
def plus_proche_voisin(d:np.ndarray) -> list:
```

qui prend en paramètre le tableau des distances résultat de la fonction `calculer_distances` (question I.A.2) et fournit un chemin d'exploration en appliquant l'algorithme du plus proche voisin.

BY-NC-SA

II.C.2) Quelle est la complexité temporelle de l'algorithme du plus proche voisin en considérant que cet algorithme est constitué des deux fonctions `calculer_distances` et `plus_proche_voisin` ?

II.C.3) En considérant les trois points de coordonnées $(0,0)$, $(0,3000)$, $(0,7000)$ et en choisissant un point de départ adéquat pour le robot, montrer que l'algorithme du plus proche voisin ne fournit pas nécessairement le plus court chemin.

Dans la pratique, on constate que, dès que le nombre de points d'intérêt devient important, l'algorithme du plus proche voisin fournit un chemin qui peut être 50% plus long que le plus court chemin.

III Deuxième approche : algorithme génétique

Les algorithmes génétiques s'inspirent de la théorie de l'évolution en simulant l'évolution d'une population. Ils font intervenir cinq traitements.

1. Initialisation
   Il s'agit de créer une population d'origine composée de m individus (ici des chemins pour l'exploration à planifier). Généralement la population de départ est produite aléatoirement.
2. Évaluation
   Cette étape consiste à attribuer à chaque individu de la population courante une note correspondant à sa capacité à répondre au problème posé. Ici la note sera simplement la longueur du chemin.
3. Sélection
   Une fois tous les individus évalués, l'algorithme ne conserve que les « meilleurs » individus. Plusieurs méthodes de sélection sont possibles : choix aléatoire, ceux qui ont obtenu la meilleure note, élimination par tournoi, etc.
4. Croisement
   Les individus sélectionnés sont croisés deux à deux pour produire de nouveaux individus et donc une nouvelle population. La fonction de croisement (ou reproduction) dépend de la nature des individus.
5. Mutation
   Une proportion d'individus est choisie (généralement aléatoirement) pour subir une mutation, c'est-à-dire une transformation aléatoire. Cette étape permet d'éviter à l'algorithme de rester bloqué sur un optimum local.

En répétant les étapes de sélection, croisement et mutation, l'algorithme fait ainsi évoluer la population, jusqu'à trouver un individu qui réponde au problème initial. Cependant dans les cas pratiques d'utilisation des algorithmes génétiques, il n'est pas possible de savoir simplement si le problème est résolu (le plus court chemin figure-t-il dans ma population ?). On utilise donc des conditions d'arrêt heuristiques basées sur un critère arbitraire.

Le but de cette partie est de construire un algorithme génétique pour rechercher un meilleur chemin d'exploration que celui obtenu par l'algorithme du plus proche voisin.

III.A – Initialisation et évaluation

Une population est représentée par une liste d'individus, chaque individu étant représenté par un couple *(longueur, chemin)* dans lequel

- *chemin* désigne un chemin représenté comme précédemment par une liste d'entiers correspondant aux indices des points d'intérêt dans le tableau des distances produit par la fonction `calculer_distances` ;
- *longueur* est un entier correspondant à la longueur du chemin, en tenant compte de la position de départ du robot.

Écrire une fonction d'entête

```
def créer_population(m:int, d:np.ndarray) -> list:
```

qui crée une population de m individus aléatoires. Cette fonction prend en paramètre le nombre d'individus à engendrer et le tableau des distances entre points d'intérêt (et la position courante du robot) tel que produit par la fonction `calculer_distances`. Elle renvoie une liste d'individus, c'est-à-dire de couples *(longueur, chemin)*.

III.B – Sélection

Écrire une fonction d'entête

```
def réduire(p:list) -> None:
```

qui réduit une population de moitié en ne conservant que les individus correspondant aux chemins les plus courts. On rappelle que `p` est une liste de couples *(longueur, chemin)*. La fonction `réduire` ne renvoie pas de résultat mais modifie la liste passée en paramètre.

BY-NC-SA

III.C – Mutation

III.C.1) Écrire une fonction d'entête

```
def muter_chemin(c:list) -> None:
```

qui prend en paramètre un chemin et le transforme en inversant aléatoirement deux de ses éléments.

III.C.2) Écrire une fonction d'entête

```
def muter_population(p:list, proba:float, d:np.ndarray) -> None:
```

qui prend en paramètre une population dont elle fait muter un certain nombre d'individus. Le paramètre `proba` (compris entre 0 et 1) désigne la probabilité de mutation d'un individu. Le paramètre `d` est la matrice des distances entre points d'intérêt.

III.D – Croisement

III.D.1) Écrire une fonction d'entête

```
def croiser(c1:list, c2:list) -> list:
```

qui crée un nouveau chemin à partir de deux chemins passés en paramètre. Ce nouveau chemin sera produit en prenant la première moitié du premier chemin suivi de la deuxième moitié du deuxième puis en « normalisant » le chemin ainsi obtenu.

III.D.2) Écrire une fonction d'entête

```
def nouvelle_génération(p:list, d:np.ndarray) -> None:
```

qui fait grossir une population en croisant ses membres pour en doubler l'effectif. Pour cela, la fonction fait se reproduire tous les couples d'individus qui se suivent dans la population `(p[i], p[i+1])` et `(p[m-1], p[0])` de façon à produire m nouveaux individus qui s'ajoutent aux m individus de la population de départ.

III.E – Algorithme complet

III.E.1) Écrire une fonction d'entête

```
def algo_génétique(PI:np.ndarray, m:int, proba:float, g:int) -> float, list:
```

qui prend en paramètre un tableau de points d'intérêts (figure 2), la taille `m` de la population, la probabilité de mutation `proba` et le nombre de générations `g`. Cette fonction implante un algorithme génétique à l'aide des différentes fonctions écrites jusqu'à présent et renvoie la longueur du plus court chemin d'exploration et le chemin lui-même obtenus au bout de `g` générations.

III.E.2) Est-il possible avec l'implantation réalisée, qu'une itération de l'algorithme dégrade le résultat : le meilleur chemin obtenu à la génération $n+1$ est plus long que celui de la génération n ?

Dans l'affirmative, comment modifier le programme pour que cette situation ne puisse plus arriver ?

III.E.3) Quelles autres conditions d'arrêt peut-on imaginer ? Établir un comparatif présentant les avantages et inconvénients de chaque condition d'arrêt envisagée.

Opérations et fonctions Python disponibles

Fonctions

- `range(n)` renvoie la séquence des `n` premiers entiers ($0 \to n-1$)
- `list(range(n))` renvoie une liste contenant les `n` premiers entiers dans l'ordre croissant : `list(range(5))` $\to$ `[0, 1, 2, 3, 4]`
- `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers)
- `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme
- `random.shuffle(u)` permute aléatoirement les éléments de la liste `u` (modifie `u`)
- `random.sample(u, n)` renvoie une liste de `n` éléments distincts de la liste `u` choisis aléatoirement, si `n > len(u)`, déclenche l'exception `ValueError`
- `math.sqrt(x)` calcule la racine carrée du nombre x
- `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à `x`

BY-NC-SA

- `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`
- `sorted(u)` renvoie une nouvelle liste contenant les éléments de la liste `u` triés dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les listes

- `len(u)` donne le nombre d'éléments de la liste `u` :
  `len([1, 2, 3])` → 3 ; `len([[1,2], [3,4]])` → 2
- `u + v` construit une liste constituée de la concaténation des listes `u` et `v` :
  `[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`
- `n * u` construit une liste constitué de la liste `u` concaténée `n` fois avec elle-même :
  `3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`
- `e in u` et `e not in u` déterminent si l'objet `e` figure dans la liste `u`, cette opération a une complexité temporelle en $O($`len(u)`$)$
  `2 in [1, 2, 3]` → `True` ; `2 not in [1, 2, 3]` → `False`
- `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (similaire à `u = u + [e]`)
- `del u[i]` supprime de la liste `u` son élément d'indice `i`
- `del u[i:j]` supprime de la liste `u` tous ses éléments dont les indices sont compris dans l'intervalle $[$`i`$,$ `j`$[$
- `u.remove(e)` supprime de la liste `u` le premier élément qui a pour valeur `e`, déclenche l'exception `ValueError` si `e` ne figure pas dans `u`, cette opération a une complexité temporelle en $O($`len(u)`$)$
- `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste
- `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`
- `u.sort()` trie la liste `u` en place, dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les tableaux (`np.ndarray`)

- `np.array(u)` crée un nouveau tableau contenant les éléments de la liste `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`
- `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou une matrice à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique numpy (`np.int16`, `np.float32`, ...). Si le paramètres `dtype` n'est pas précisé, il prend la valeur `float` par défaut
- `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens
- `a.ndim` nombre de dimensions du tableau `a` (1 pour un vecteur, 2 pour une matrice, etc.)
- `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions
- `len(a)` taille du tableau `a` dans sa première dimension (nombre d'éléments d'un vecteur, nombre de lignes d'une matrice, etc.) équivalent à `a.shape[0]`
- `a.size` nombre total d'éléments du tableau `a`
- `a.flat` itérateur sur tous les éléments du tableau `a`
- `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O($`a.size`$)$
- `b in a` détermine si `b` est un élément du tableau `a` ; si `b` est un scalaire, vérifie si `b` est un élément de `a` ; si `b` est un vecteur ou une liste et `a` une matrice, détermine si `b` est une ligne de `a`
- `np.concatenate((a1, a2))` construit un nouveau tableau en concaténant deux tableaux ; `a1` et `a2` doivent avoir le même nombre de dimensions et la même taille à l'exception de leur taille dans la première dimension (deux matrices doivent avoir le même nombre de colonnes pour pouvoir être concaténées)

• • • FIN • • •

BY-NC-SA

Concours Centrale-Supélec 2017 filière PC

Informatique

Présentation du sujet

Le sujet porte sur le thème de la recherche de plus court chemin, appliqué au cas d'un système embarqué sur le robot Spirit, où une optimisation des déplacements entre points de mesure permet d'économiser de l'énergie.

La première partie s'intéresse à l'extraction de points d'intérêt où le robot doit réaliser des mesures, soit de façon aléatoire pour des besoins de tests de l'algorithme, soit par analyse d'image, ou encore par extraction des informations dans une base de données.

La deuxième partie propose une première approche du calcul du chemin entre les points d'intérêt par l'algorithme du plus proche voisin. Une comparaison de complexité est faite entre le traitement « par force brute », c'est-à-dire en déterminant la longueur de la totalité des chemins possibles et l'algorithme du plus proche voisin, bien plus rapide. La dernière question permet néanmoins de montrer que l'algorithme ne détermine pas nécessairement le chemin le plus court.

La troisième partie aborde une seconde approche, par un algorithme génétique, mieux à même de trouver un chemin plus court, mais pour lequel il faut trouver un critère d'arrêt adapté.

Analyse globale des résultats

Le sujet est de longueur raisonnable pour le temps imparti. De nombreux candidats abordent la totalité du sujet.

À nouveau cette année, le jury se réjouit du niveau satisfaisant des copies. Le langage est bien maitrisé et permet de traduire les solutions aux questions sans difficultés. Seule une petite proportion des candidats (moins de 5%) rend une copie vide ou contenant des programmes sans aucune cohérence, ne sachant pas écrire une boucle, réaliser une simple affectation correctement ou discerner le nombre de dimensions d'un tableau. Ces copies conduisent à quelques notes très faibles et demeurent une énigme pour le jury, après trois semestres de cours et de travaux dirigés en informatique. Il est possible que ces copies soient celles de candidats n'ayant pas suivi d'enseignement d'informatique de classe préparatoire.

Les petites erreurs syntaxiques n'ont pas été retenues par le jury comme un élément discriminatoire, dans la mesure où elles ne cachent pas des erreurs de fond. Les réponses pertinentes d'un point de vue algorithmique sont valorisées.

Certaines copies proposent des programmes particulièrement élégants et concis et reflètent un vrai recul sur les différentes stratégies de programmation. Ces copies ont été valorisées.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Au regard des copies évaluées, le jury propose aux futurs candidats de prêter attention aux remarques suivantes.

L'indentation en Python délimite les blocs d'instructions et doit apparaitre clairement dans la rédaction. Toute présentation claire est bienvenue ; bien souvent, un trait vertical marquant l'alignement du bloc d'instruction est suffisant.

L'initialisation d'une variable dans une boucle ou hors de la boucle n'a pas les mêmes conséquences pour l'algorithme.

Le nombre d'itérations d'une boucle doit être bien réfléchi pour s'assurer que les indices des éléments d'une liste appelée dans la boucle sont bien définis. L'instruction `range(n)` parcourt `n` itérations indicées de `0` à `n-1`.

La concision et l'élégance des programmes sont appréciées dans l'évaluation. Les candidats qui réinvestissent les fonctions déjà codées sont valorisés par rapport à ceux qui recopient les lignes de code équivalentes. Bien souvent, une condition booléenne bien choisie permet d'éviter de longues listes de conditions aux instructions identiques.

Des noms de variables explicites aident à la compréhension du code. De trop nombreux candidats utilisent des noms de variables non significatifs (`a`, `b`...) ce qui nuit à la compréhension du programme. La clarté du programme (en particulier le choix des noms de variables) ainsi que la présence de commentaires opportuns sont prises en compte dans l'évaluation.

Lors d'un commentaire sur une complexité, une justification chiffrée minimale est attendue : un programme de complexité exponentielle pourrait être utilisable pour peu que le nombre d'opérations soit faible au regard de la puissance de calcul d'un micro-processeur actuel.

L'ordre des questions importe. Prendre soin de rédiger les réponses aux questions en respectant leur ordre dans le sujet.

La qualité d'expression (l'orthographe notamment) et la qualité visuelle de présentation relèvent des compétences de communication indispensables à un candidat à une école d'ingénieur. Le correcteur n'attribue les points qu'aux éléments de réponse qu'il parvient à lire et à comprendre. Les copies obscures et difficiles à comprendre sont pénalisées.

Les variables utilisées dans une fonction doivent être définies dans cette fonction ou être explicitement définies comme variables globales (soit par le sujet, soit par le candidat). Beaucoup de candidats ont utilisé la variable `n` sans la définir, ce qui a soulevé une ambiguïté, `n` pouvant être le nombre de points d'intérêt incluant la position initiale du robot ou pas.

Les candidats sont invités à bien lire l'annexe contenant certaines fonctions utiles pour traiter le sujet.

Première partie

Les points d'intérêt choisis au hasard devaient être distincts. Certains candidats ne vérifient pas cette contrainte dans leur programme. Le calcul des distances devait prendre en compte (dans la dernière colonne) les distances à la position courante du robot. De nombreuses erreurs sur les indices ont été relevées, que ce soit sur le nombre d'itérations ou les indices des éléments du tableau. Certains candidats confondent encore l'affectation (`=`) et le test d'égalité (`==`).

Les explications sur la fonction `F1` se limitent parfois à paraphraser les lignes de code. Il était attendu une description de la tache globale réalisée et une interprétation du résultat renvoyé.

Les deux premières questions de bases de données sont très souvent abordées et bien réussies. C'est moins le cas pour les trois questions suivantes. Les questions 3 et 5 faisaient intervenir des jointures. La question 4 nécessitait de prendre des initiatives en définissant le nombre de bit de codage des entiers pour en déduire la surface maximale d'exploration enregistrable.

Deuxième partie

Cette partie débute avec deux fonctions relativement simples et utiles pour la suite. La longueur du chemin est souvent bien calculée, lorsque la position courante du robot est prise en compte. En revanche, la normalisation fait souvent l'objet de programmes décousus. L'erreur la plus fréquente étant d'engager une boucle sur la longueur du chemin tout en supprimant des éléments de ce même chemin, ce qui ne manque pas de conduire à un débordement et à des décalages d'indices.

La plupart des candidats trouvent le nombre de chemins possibles mais un minimum de justification était attendu. Pour conclure, il fallait non seulement réaliser une application numérique pour déterminer l'ordre de grandeur du nombre d'itérations à prévoir, mais aussi pour comparer ce nombre aux puissances de calcul des processeurs actuels.

L'algorithme du plus proche voisin était probablement le plus complexe à élaborer, sachant qu'il convenait de s'assurer de ne pas repasser sur un point déjà visité. Les réponses ont souvent conduit à des programmes sans queue ni tête, difficilement évaluables par le correcteur en l'absence de commentaires. Le calcul de complexité devait prendre en compte la complexité d'une instruction telle que `if k not in L`.

Beaucoup de candidats ont trouvé un exemple où l'algorithme ne fournit pas le plus court chemin. Un schéma était souvent plus clair qu'un long discours à cette question.

Troisième partie

Les fonctions `créer_population` et `réduire` distinguent les candidats qui savent utiliser judicieusement les fonctions mises à disposition et ceux qui tentent de se débrouiller sans. Les questions suivantes n'ont pas posé beaucoup de difficultés aux candidats attentifs aux indices. Une grande partie des candidats est à l'aise avec la manipulation des listes pour effectuer des concaténations et du slicing.

La construction de la fonction principale `algo_génétique`, réutilisant bon nombre des fonctions précédentes, est bien réussie. L'argumentation sur le critère d'arrêt est souvent pertinente lorsque la question est abordée.

Conclusion

Le sujet aborde une large partie du programme d'informatique commune. Le choix d'un sujet s'appuyant sur un thème courant en informatique assure une cohérence avec la formation d'ingénieur. Cette approche sera reconduite sur des problématiques de simulation ou d'algorithmique en informatique, à partir du programme des trois semestres d'informatique.

Les bons résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences affirmées en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrices autorisées

2016

Prévention des collisions aériennes

Ce problème s'intéresse à différents aspects relatifs à la sécurité aérienne et plus précisément au risque de collision entre deux appareils. Dans la première partie nous abordons l'enregistrement des plans de vol des différentes compagnies aériennes. La deuxième partie explique la manière d'attribuer à chaque vol un couloir aérien répondant au mieux aux souhaits des compagnies aériennes. Enfin, la troisième partie s'intéresse à la procédure de détection d'un risque de collision entre deux avions se croisant à des altitudes proches.

Une liste d'opérations et de fonctions qui peuvent être utilisées dans les fonctions Python figure en fin d'énoncé. Les candidats peuvent coder toute fonction complémentaire qui leur semble utile. Ils devront dans ce cas préciser le rôle de cette fonction, la signification de ses paramètres et la nature de la valeur renvoyée.

I Plan de vol

Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut 30,48 cm) au-dessus de la surface isobare à 1013,25 hPa. L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres « FL » (*flight level*) suivies de l'altitude en centaines de pieds : « FL310 » désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence.

EUROCONTROL est l'organisation européenne chargée de la navigation aérienne, elle gère plusieurs dizaines de milliers de vol par jour. Toute compagnie qui souhaite faire traverser le ciel européen à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, EUROCONTROL peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc.

Nous modélisons (de manière très simplifiée) les plans de vol gérés par EUROCONTROL sous la forme d'une base de données comportant deux tables :

— la table `vol` qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes
- `id_vol` : numéro du vol (chaine de caractères) ;
- `depart` : code de l'aéroport de départ (chaine de caractères) ;
- `arrivee` : code de l'aéroport d'arrivée (chaine de caractères) ;
- `jour` : jour du vol (de type `date`, affiché au format `aaaa-mm-jj`) ;
- `heure` : heure de décollage souhaitée (de type `time`, affiché au format `hh:mi`) ;
- `niveau` : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
AF1204	CDG	FCO	2016-05-02	07:35	300
AF1205	FCO	CDG	2016-05-02	10:25	300
AF1504	CDG	FCO	2016-05-02	10:05	310
AF1505	FCO	CDG	2016-05-02	13:00	310

Figure 1 Extrait de la table `vol` : vols de la compagnie Air France entre les aéroports Charles-de-Gaule (Paris) et Léonard-de-Vinci à Fiumicino (Rome)

— la table `aeroport` qui répertorie les aéroports européens ; elle contient les colonnes
- `id_aero` : code de l'aéroport (chaine de caractères) ;
- `ville` : principale ville desservie (chaine de caractères) ;
- `pays` : pays dans lequel se situe l'aéroport (chaine de caractères).

id_aero	ville	pays
CDG	Paris	France
ORY	Paris	France
MRS	Marseille	France
FCO	Rome	Italie

Figure 2 Extrait de la table `aeroport`

BY-NC-SA

Les types SQL `date` et `time` permettent de mémoriser respectivement un jour du calendrier grégorien et une heure du jour. Deux valeurs de type `date` ou de type `time` peuvent être comparées avec les opérateurs habituels (`=`, `<`, `<=`, etc.). La comparaison s'effectue suivant l'ordre chronologique. Ces valeurs peuvent également être comparées à une chaine de caractères correspondant à leur représentation externe (`'aaaa-mm-jj'` ou `'hh:mi'`).

I.A – Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.

I.B – Écrire une requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport desservant Paris le 2 mai 2016.

I.C – Que fait la requête suivante ?

```
SELECT id_vol
FROM vol
     JOIN aeroport AS d ON d.id_aero = depart
     JOIN aeroport AS a ON a.id_aero = arrivee
WHERE
     d.pays = 'France' AND
     a.pays = 'France' AND
     jour = '2016-05-02'
```

I.D – Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples $(\mathrm{Id}_1, \mathrm{Id}_2)$ des identifiants des vols dans cette situation.

II Allocation des niveaux de vol

Lors du dépôt d'un plan de vol, la compagnie aérienne doit préciser à quel niveau de vol elle souhaite faire évoluer son avion lors de la phase de croisière. Ce niveau de vol souhaité, le RFL pour *requested flight level*, correspond le plus souvent à l'altitude à laquelle la consommation de carburant sera minimale. Cette altitude dépend du type d'avion, de sa charge, de la distance à parcourir, des conditions météorologiques, etc.

Cependant, du fait des similitudes entre les différents avions qui équipent les compagnies aériennes, certains niveaux de vols sont très demandés ce qui engendre des conflits potentiels, deux avions risquant de se croiser à des altitudes proches. Les contrôleurs aériens de la région concernée par un conflit doivent alors gérer le croisement de ces deux avions.

Pour alléger le travail des contrôleurs et diminuer les risques, le système de régulation s'autorise à faire voler un avion à un niveau différent de son RFL. Cependant, cela engendre généralement une augmentation de la consommation de carburant. C'est pourquoi on limite le choix aux niveaux immédiatement supérieur et inférieur au RFL.

Ce problème de régulation est modélisé par un graphe dans lequel chaque vol est représenté par trois sommets. Le sommet 0 correspond à l'attribution du RFL, le sommet + au niveau supérieur et le sommet − au niveau inférieur. Chaque conflit potentiel entre deux vols sera représenté par une arête reliant les deux sommets concernés. Le cout d'un conflit potentiel (plus ou moins important en fonction de sa durée, de la distance minimale entre les avions, etc.) sera représenté par une valuation sur l'arête correspondante.

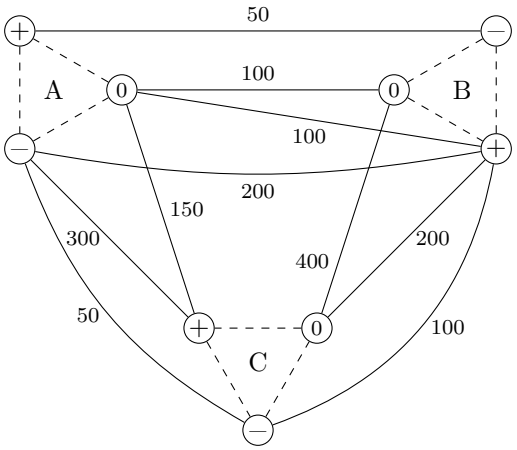

Figure 3 Exemple de conflits potentiels entre trois vols

Dans l'exemple de la figure 3, faire voler les trois avions à leur RFL engendre un cout de régulation entre A et B de 100 et un cout de régulation entre B et C de 400, soit un cout total de la régulation de 500 (il n'y a pas de

BY-NC-SA

conflit entre A et C). Faire voler l'avion A à son RFL et les avions B et C au-dessus de leur RFL engendre un conflit potentiel de cout 100 entre A et B et 150 entre A et C, soit un cout total de 250 (il n'y a plus de conflit entre B et C).

On peut observer que cet exemple possède des solutions de cout nul, par exemple faire voler A et C à leur RFL et B au-dessous de son RFL. Mais en général le nombre d'avions en vol est tel que des conflits potentiels sont inévitables. Le but de la régulation est d'imposer des plans de vol qui réduisent le plus possible le cout total de la résolution des conflits.

II.A – Implantation du problème

Chaque vol étant représenté par trois sommets, le graphe des conflits associé à n vols $v_0, v_1, ..., v_{n-1}$ possède $3n$ sommets que nous numéroterons de 0 à $3n-1$. Nous conviendrons que pour $0 \leqslant k < n$:
- le sommet $3k$ représente le vol v_k à son RFL ;
- le sommet $3k+1$ représente le vol v_k au-dessus de son RFL ;
- le sommet $3k+2$ représente le vol v_k au-dessous de son RFL ;

Le cout de chaque conflit potentiel est stocké dans une liste de $3n$ listes de $3n$ entiers (tableau $3n \times 3n$) accessible grâce à la variable globale `conflit` : si i et j désignent deux sommets du graphe, alors `conflit[i][j]` est égal au cout du conflit potentiel (s'il existe) entre les plans de vol représentés par les sommets i et j. S'il n'y a pas de conflit entre ces deux sommets, `conflit[i][j]` vaut 0. On convient que `conflit[i][j]` vaut 0 si les sommets i et j correspondent au même vol (figure 4).

On notera que pour tout couple de sommets (i, j), `conflit[i][j]` et `conflit[j][i]`, représentent un seul et même conflit et donc `conflit[i][j] == conflit[j][i]`.

```
conflit = [ [   0,   0,   0, 100, 100,   0,   0, 150,   0],
            [   0,   0,   0,   0,   0,  50,   0,   0,   0],
            [   0,   0,   0,   0, 200,   0,   0, 300,  50],
            [ 100,   0,   0,   0,   0,   0, 400,   0,   0],
            [ 100,   0, 200,   0,   0,   0, 200,   0, 100],
            [   0,  50,   0,   0,   0,   0,   0,   0,   0],
            [   0,   0,   0, 400, 200,   0,   0,   0,   0],
            [ 150,   0, 300,   0,   0,   0,   0,   0,   0],
            [   0,   0,  50,   0, 100,   0,   0,   0,   0] ]
```

Figure 4 Tableau des couts des conflits associé au graphe représenté figure 3

II.A.1) Écrire en Python une fonction `nb_conflits()` sans paramètre qui renvoie le nombre de conflits potentiels, c'est-à-dire le nombre d'arêtes de valuation non nulle du graphe.

II.A.2) Exprimer en fonction de n la complexité de cette fonction.

II.B – Régulation

Pour un vol v_k on appelle *niveau relatif* l'entier r_k valant 0, 1 ou 2 tel que :
- $r_k = 0$ représente le vol v_k à son RFL ;
- $r_k = 1$ représente le vol v_k au-dessus de son RFL ;
- $r_k = 2$ représente le vol v_k au-dessous son RFL.

On appelle *régulation* la liste $(r_0, r_1, ..., r_{n-1})$. Par exemple, la régulation $(0, 0, ..., 0)$ représente la situation dans laquelle chaque avion se voit attribuer son RFL. Une régulation sera implantée en Python par une liste d'entiers.

Il pourra être utile d'observer que les sommets du graphe des conflits choisis par la régulation r portent les numéros $3k + r_k$ pour $0 \leqslant k < n$. On remarque également qu'au sommet s du graphe correspond le niveau relatif $r_k = s \bmod 3$ et le vol v_k tel que $k = \lfloor s/3 \rfloor$.

II.B.1) Écrire en Python une fonction `nb_vol_par_niveau_relatif(regulation)` qui prend en paramètre une régulation (liste de n entiers) et qui renvoie une liste de 3 entiers `[a, b, c]` dans laquelle a est le nombre de vols à leurs niveaux RFL, b le nombre de vols au-dessus de leurs niveaux RFL et c le nombre de vols au-dessous de leurs niveaux RFL.

II.B.2) Cout d'une régulation

On appelle *cout d'une régulation* la somme des couts des conflits potentiels que cette régulation engendre.

a) Écrire en Python une fonction `cout_regulation(regulation)` qui prend en paramètre une liste représentant une régulation et qui renvoie le cout de celle-ci.

b) Évaluer en fonction de n, la complexité de cette fonction.

c) Déduire de la question a) une fonction `cout_RFL()` qui renvoie le cout de la régulation pour laquelle chaque avion vole à son RFL.

II.B.3) Combien existe-t-il de régulations possibles pour n vols ?

Est-il envisageable de calculer les couts de toutes les régulations possibles pour trouver celle de cout minimal ?

CC BY-NC-SA

II.C – L'algorithme Minimal

On définit le *cout d'un sommet* comme la somme des couts des conflits potentiels dans lesquels ce sommet intervient. Par exemple, le cout du sommet correspondant au niveau RFL de l'avion A dans le graphe de la figure 3 est égal à $100 + 100 + 150 = 350$.

L'algorithme *Minimal* consiste à sélectionner le sommet du graphe de cout minimal ; une fois ce dernier trouvé, les deux autres niveaux possibles de ce vol sont supprimés du graphe et on recommence avec ce nouveau graphe jusqu'à avoir attribué un niveau à chaque vol.

Dans la pratique, plutôt que de supprimer effectivement des sommets du graphe, on utilise une liste `etat_sommet` de $3n$ entiers tels que :
- `etat_sommet[s]` vaut 0 lorsque le sommet s a été supprimé du graphe ;
- `etat_sommet[s]` vaut 1 lorsque le sommet s a été choisi dans la régulation ;
- `etat_sommet[s]` vaut 2 dans les autres cas.

II.C.1)

a) Écrire en Python une fonction `cout_du_sommet(s, etat_sommet)` qui prend en paramètres un numéro de sommet s (n'ayant pas été supprimé) ainsi que la liste `etat_sommet` et qui renvoie le cout du sommet s dans le graphe défini par la variable globale `conflit` et le paramètre `etat_sommet`.

b) Exprimer en fonction de n la complexité de la fonction `cout_du_sommet`.

II.C.2)

a) Écrire en Python une fonction `sommet_de_cout_min(etat_sommet)` qui, parmi les sommets qui n'ont pas encore été choisis ou supprimés, renvoie le numéro du sommet de cout minimal.

b) Exprimer en fonction de n la complexité de la fonction `sommet_de_cout_min`.

II.C.3)

a) En déduire une fonction `minimal()` qui renvoie la régulation résultant de l'application de l'algorithme Minimal.

b) Quelle est sa complexité ? Commenter.

II.D – Recuit simulé

L'algorithme de *recuit simulé* part d'une régulation initiale quelconque (par exemple la régulation pour laquelle chacun des avions vole à son RFL) et d'une valeur positive T choisie empiriquement. Il réalise un nombre fini d'étapes se déroulant ainsi :
- un vol v_k est tiré au hasard ;
- on modifie r_k en tirant au hasard parmi les deux autres valeurs possibles ;
  - si cette modification diminue le cout de la régulation, cette modification est conservée ;
  - sinon, cette modification n'est conservée qu'avec une probabilité $p = \exp(-\Delta c/T)$, où Δc est l'augmentation de cout liée à la modification de la régulation ;
- le paramètre T est diminué d'une certaine quantité.

Écrire en Python une fonction `recuit(regulation)` qui modifie la liste `regulation` passée en paramètre en appliquant l'algorithme du recuit simulé. On fera débuter l'algorithme avec la valeur $T = 1000$ et à chaque étape la valeur de T sera diminuée de 1%. L'algorithme se terminera lorsque $T < 1$.

Remarque. Dans la pratique, l'algorithme de recuit simulé est appliqué plusieurs fois de suite en partant à chaque fois de la régulation obtenue à l'étape précédente, jusqu'à ne plus trouver d'amélioration notable.

III Système d'alerte de trafic et d'évitement de collision

L'optimisation globale des niveaux de vol d'un système d'avions étudiée précédemment est complétée par une surveillance locale dans l'objectif de maintenir à tout instant une séparation suffisante avec tout autre avion. La règlementation actuelle impose aux avions de ligne d'être équipé d'un système embarqué d'évitement de collision en vol, ou TCAS pour *traffic collision avoidance system*.

Nous nous intéressons au fonctionnement du TCAS vu d'un avion particulier que nous appelons *avion propre*. Les avions qui volent à proximité de l'avion propre sont qualifiés d'*intrus*.

Le système TCAS surveille l'environnement autour de l'avion propre (typiquement dans un rayon d'un soixantaine de kilomètres) pour identifier les intrus et déterminer s'ils présentent un risque de collision. Pour cela, le TCAS évalue, pour chaque intrus, le point où sa distance avec l'avion propre sera minimale. Ce point est appelé CPA, pour *closest point of approach*. Le fonctionnement global (simplifié) du système TCAS est décrit par la fonction `TCAS` donnée figure 5.

Cette fonction maintient une liste des CPA des avions situés dans son périmètre de surveillance (variable `CPAs`). Cette liste est limitée à un nombre restreint d'intrus (`intrus_max`) dont l'instant du CPA est proche (dans moins de `suivi_max`) de façon à garantir la détection et le traitement d'un éventuel danger dans un temps suffisamment court.

BY-NC-SA

```
def TCAS():
    """Fonction principale du système TCAS."""
    intrus_max = 30 # nombre maximum d'avions suivis
    suivi_max = 100 # délai maximum pour le CPA (en secondes)
    CPAs = []  # liste des CPA des avions suivis
    while (TCAS_actif()):
        intrus = acquerir_intrus()
        enregistrer_CPA(intrus, CPAs, intrus_max, suivi_max)
        traiter_CPAs(CPAs)
```

Figure 5

La fonction `TCAS_actif` renvoie la position de l'interrupteur principal du système.

La fonction `acquerir_intrus` détermine la position et la vitesse d'un intrus particulier. À chaque appel, elle s'intéresse à un avion différent dans le périmètre de surveillance du TCAS. Lorsque tous les intrus ont été examinés, l'appel suivant revient au premier intrus qui est toujours dans le périmètre.

La fonction `enregistrer_CPA` intègre dans la liste CPA de nouvelles informations telles que fournies par la fonction `acquerir_intrus`.

Enfin la fonction `traiter_CPAs` examine les CPA des intrus les plus dangereux pour décider si l'un d'eux présente un risque de collision. Si c'est le cas, cette fonction détermine la manœuvre à effectuer (monter ou descendre) en coordination avec le système TCAS de l'intrus concerné et génère une alarme et une consigne à destination du pilote.

III.A – Acquisition et stockage des données

Chaque avion est équipé d'un émetteur radio spécialisé, appelé *transpondeur*, qui fournit automatiquement, en réponse à l'interrogation d'une station au sol ou d'un autre avion, des informations sur l'avion dans lequel il est installé. La fonction `acquerir_intrus` utilise les données du système de navigation de l'avion propre, les données fournies par le transpondeur de l'intrus, le relèvement de son émission et les informations fournies par le système de contrôle aérien au sol.

III.A.1) Les transpondeurs utilisent tous la même fréquence radio. Afin d'éviter la saturation de cette fréquence, en particulier dans les zones à fort trafic, chaque émission ne doit pas durer plus de 128 µs. Le débit binaire utilisé est de 10^6 bits par seconde ; chaque message commence par une marque de début de 6 bits et se termine par 4 bits de contrôle et une marque de fin de 6 bits.

$$\underbrace{d\,d\,d\,d\,d\,d}_{\text{début}}\ \underbrace{x\,x\,x\ldots x\,x\,x}_{\text{données}}\ \underbrace{e\,e\,e\,e}_{\text{contrôle}}\ \underbrace{f\,f\,f\,f\,f\,f}_{\text{fin}}$$

Déterminer le nombre maximum de bits de données dans une émission de transpondeur.

III.A.2) Le système TCAS souhaite récupérer l'altitude et la vitesse ascensionnelle de chaque intrus en interrogeant son transpondeur. La réponse du transpondeur contient systématiquement un numéro d'identification de l'avion sur 24 bits. Les autres informations sont des entiers codés en binaire qui peuvent varier dans les intervalles suivants :

– altitude de 2000 à 66 000 pieds ;
– vitesse ascensionnelle de −5000 à 5000 pieds par minute.

La taille d'un message de transpondeur est-elle suffisante pour obtenir ces informations en une seule fois ?

III.A.3) Après avoir récupéré les informations nécessaires, la fonction `acquerir_intrus` calcule la position et la vitesse de l'intrus par rapport à l'avion propre et renvoie une liste de huit nombres :

$$[\texttt{id},\ \texttt{x},\ \texttt{y},\ \texttt{z},\ \texttt{vx},\ \texttt{vy},\ \texttt{vz},\ \texttt{t0}]$$

où

– `id` est le numéro d'identification de l'intrus ;
– `x, y, z` les coordonnées (en mètres) de l'intrus dans un repère orthonormé $\mathcal{R}_0$ lié à l'avion propre ;
– `vx, vy, vz` la vitesse (en mètres par seconde) de l'intrus dans ce même repère ;
– `t0` le moment de la mesure (en secondes depuis un instant de référence).

À des fins d'analyse une fois l'avion revenu au sol, la fonction `acquerir_intrus` conserve chaque résultat obtenu. Chaque nombre est stocké sur 4 octets. En supposant que cette fonction est appelé au maximum 100 fois par seconde, quel est le volume de mémoire nécessaire pour conserver les données de 100 heures de fonctionnement du TCAS ?

Ce volume de stockage représente-t-il une contrainte technique forte ?

BY-NC-SA

III.B – Estimation du CPA

Le but de cette sous-partie III.B est d'estimer le CPA d'un avion intrus à partir des informations fournies par la fonction `acquerir_intrus`. Pour cela on suppose que l'avion propre et l'avion intrus poursuivent leur route sans changer de direction ni de vitesse. Vu les distances entre les deux avions, on néglige la courbure de la Terre, on considère donc qu'ils suivent un mouvement rectiligne uniforme.

III.B.1) On se place dans le repère orthonormé $\mathcal{R}_0$ utilisé par la fonction `acquerir_intrus`. On note O l'origine de ce repère (qui correspond à la position de l'avion propre), $G(t)$ la position de l'intrus à l'instant t et $\overrightarrow{V}$ sa vitesse dans $\mathcal{R}_0$ (supposée constante). La fonction `acquerir_intrus` fournit ainsi les coordonnées dans $\mathcal{R}_0$ des vecteurs $\overrightarrow{OG}(t_0)$ et $\overrightarrow{V}$ pour l'intrus `id`.

Exprimer le vecteur $\overrightarrow{OG}(t)$ en fonction des informations fournies par la fonction `acquerir_intrus`.

III.B.2) Déterminer l'expression du temps t_c qui correspond à l'instant où les deux avions sont le plus proches (instant du CPA).

III.B.3) Montrer qu'il n'y a pas de risque de collision si le produit scalaire $\overrightarrow{OG}(t_0) \cdot \overrightarrow{V}$ est positif.

III.B.4) Écrire en Python une fonction `calculer_CPA(intrus)` qui prend en paramètre une liste de la forme de celle produite par la fonction `acquerir_intrus` et qui renvoie :
- `None` s'il n'y pas de risque de collision ;
- ou une liste de 3 nombre `[tCPA, dCPA, zCPA]` où
  - `tCPA` est l'instant prévu pour le CPA (t_c) ;
  - `dCPA` la distance en mètres entre les deux avions au moment du CPA ;
  - `zCPA` la différence d'altitude en pieds entre les deux avions au moment du CPA (négative si l'intrus est plus bas que l'avion propre).

Pour répondre à cette question, on considère que l'avion propre vole horizontalement (vol de croisière) et que l'axe $\vec{z}$ du repère $\mathcal{R}_0$ est orienté verticalement vers le haut.

III.C – Mise à jour de la liste des CPA

Chaque élément de la liste `CPAs` (figure 5) est une liste de 4 nombres `[id, tCPA, dCPA, zCPA]` où `id` est l'identifiant de l'intrus et les autres éléments ont la signification précisée à la question III.B.4. Un exemple de liste `CPAs` est donné figure 6. Ainsi, `CPAs[2][3]` indique la séparation verticale (800 pieds) au CPA pour le troisième avion suivi, dont l'identifiant est 32675398 et dont le CPA aura lieu à l'instant 1462190455 (nombre de secondes depuis l'instant de référence).

```
CPAs = [ [ 11305708,  1462190400,  2450, -1000],
         [ 12416823,  1462190412,  2500,   500],
         [ 32675398,  1462190455,  2000,   800],
         [ 18743283,  1462190463,  2100,  -200] ]
```

Figure 6 Exemple de liste de CPA

La fonction `traiter_CPA` attend une liste triée par ordre chronologique. La fonction `enregistrer_CPA` s'assurera donc que la liste `CPAs` qu'elle produit est triée par ordre croissant des `tCPA` (`CPAs[i-1][1] <= CPAs[i][1]` pour tout `i` compris entre 1 et `len(CPAs)-1`). La fonction `traiter_CPAs` ne modifie pas l'ordre de la liste `CPAs`, `enregistrer_CPA` peut donc supposer que la liste qu'elle reçoit en paramètre est déjà triée.

III.C.1) Écrire en Python une fonction `mettre_a_jour_CPAs(CPAs, id, nv_CPA, intrus_max, suivi_max)` qui prend en paramètre
- `CPAs` : la liste actuelle des CPA au format décrit ci-dessus ;
- `id` : l'identifiant d'un intrus ;
- `nv_CPA` : les caractéristiques du CPA de l'intrus telles que renvoyées par la fonction `calcul_CPA` (peut éventuellement valoir `None`) ;
- `intrus_max` : un entier indiquant le nombre maximum d'intrus à suivre ;
- `suivi_max` : un entier donnant le délai maximum (en secondes) avant un CPA pour s'intéresser à l'intrus correspondant.

Cette fonction modifie la liste `CPAs` en appliquant les règles suivantes :
- si l'avion `id` est déjà suivi et ne présente plus de risque de collision, ou si son CPA est prévu dans plus de `suivi_max` secondes, il est supprimé de la liste ;
- si l'avion `id` est déjà suivi et que son CPA est prévu dans moins de `suivi_max` secondes, la ligne correspondante est mise à jour avec les nouvelles informations sur le CPA ;
- si l'avion `id` n'est pas suivi et que son CPA est prévu dans moins de `suivi_max` secondes alors,
  - s'il reste de la place dans la liste il est ajouté à la fin,
  - si la liste est pleine (elle contient déjà `intrus_max` intrus) et si le `tCPA` du nouvel intrus est inférieur au `tCPA` du dernier de la liste alors le nouvel intrus remplace le dernier intrus de la liste.

BY-NC-SA

La fonction `mettre_a_jour_CPAs` renvoie :
- `None` si l'avion `id` a été supprimé ou n'a pas été ajouté ;
- ou un entier indiquant l'indice de la ligne de `CPAs` qui a été modifiée ou ajoutée.

III.C.2) Suite à un appel à la fonction `mettre_a_jour_CPAs`, il se peut que la liste `CPAs` ne soit plus triée par ordre croissant des `tCPA`.

Écrire en Python une fonction `replacer(ligne, CPAs)` qui modifie la liste `CPAs` de façon à ce qu'elle soit ordonnée par `tCPA` croissant, en sachant que la seule ligne qui n'est éventuellement pas à sa place est celle d'indice `ligne`.

Autrement dit, la fonction `replacer(ligne, CPAs)` doit remettre la ligne `CPAs[ligne]` à sa place dans l'ordre des `tCPA` croissants.

III.C.3) Écrire en Python la fonction `enregistrer_CPA` utilisée par la fonction `TCAS` (figure 5).

III.D – Évaluation des paramètres généraux du système TCAS

III.D.1) Le système TCAS est capable de détecter un intrus à une distance d'environ 60 km. Un avion de ligne vole typiquement à 900 km·h^{-1}. Quel temps va-t-il s'écouler entre la première détection d'un intrus et son CPA si celui-ci se situe à proximité de l'avion propre ?

Comparer ce résultat à la valeur standard du paramètre `suivi_max` (100 s) et commenter.

III.D.2) Afin de garantir le confort des passagers, les pilotes limitent généralement la vitesse ascensionnelle de leur avion à 1500 pieds par minute.

Si deux avions volent l'un vers l'autre à la même altitude, combien de temps avant leur CPA doivent-ils commencer à manœuvrer afin de se croiser avec une différence d'altitude d'au moins 500 pieds ?

III.D.3) Le système TCAS émet une alarme dans le cockpit demandant au pilote de monter ou descendre s'il détecte un intrus avec un `zCPA` inférieur à 500 pieds et un `tCPA` dans moins de 25 secondes. Commenter cette valeur.

La consigne de monter ou descendre est élaborée après concertation entre les systèmes TCAS des deux avions concernés afin d'éviter de conseiller la même manœuvre aux deux pilotes.

III.D.4) Les spécifications du système TCAS prévoient que la position de chaque intrus doit être vérifiée au moins une fois par seconde. Elles limitent d'autre part le nombre d'intrus suivis à 30. En déduire le temps maximum dont dispose la fonction `TCAS` pour exécuter une fois sa boucle.

III.D.5) Quel est le facteur limitant la vitesse d'exécution de la fonction `TCAS` ? En déduire un ordre de grandeur du temps minimum d'exécution d'une boucle. Est-il compatible avec les spécifications du système TCAS ?

Opérations et fonctions Python disponibles

Fonctions
- `exp(x)` calcule l'exponentielle du nombre x
- `randint(n)` (`n` entier) renvoie un entier aléatoire compris entre 0 et `n-1` inclus
- `random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme
- `sqrt(x)` calcule la racine carrée du nombre x
- `time()` renvoie l'heure sous la forme d'un nombre de secondes depuis un instant de référence

Opérations sur les listes
- `u + v` construit une liste constituée de la concaténation des listes `u` et `v` :
  `[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`
- `n * u` construit une liste constitué de la liste `u` concaténée `n` fois avec elle-même :
  `3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`
- `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (équivalent à `u = u + [e]`)
- `del(u[i])` supprime de la liste `u` son élément d'indice `i`
- `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste
- `len(u)` donne le nombre d'éléments de la liste `u` :
  `len([1, 2, 3])` → 3, `len([[1,2], [3,4]])` → 2
- `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`

• • • FIN • • •

BY-NC-SA

Concours Centrale-Supélec 2016 filière PC

Informatique

Présentation du sujet

Le sujet porte sur le thème du trafic aérien, et en particulier la gestion optimisée de l'attribution des niveaux de vol par Eurocontrol et le système anti-collision TCAS.

La première partie traite des requêtes d'extraction des données de vol utiles, à partir des bases de données stockant les caractéristiques des vols programmés et des aéroports.

La deuxième partie s'intéresse à l'attribution des niveaux de vol aux compagnies aériennes. Plusieurs algorithmes sont envisagés, permettant de détecter les conflits et de minimiser les couts de changement de niveaux de vol. Le nombre important de vols à gérer à l'échelle européenne nécessite une attention particulière au regard de la complexité des solutions envisagées.

La troisième partie porte sur le système anti-collision TCAS, embarqué dans les avions, qui traite les données de vol reçues de la part des autres avions environnant afin d'alerter le pilote en cas de danger de collision.

Analyse globale des résultats

Le sujet est de longueur raisonnable pour le temps imparti. De nombreux candidats abordent la totalité du sujet.

À nouveau cette année, le jury se réjouit du niveau satisfaisant des copies. Le langage est bien maitrisé et permet de traduire les solutions aux questions sans difficultés. Seule une petite proportion des candidats (de l'ordre de 5%) a de réelles difficultés à construire un programme, en respectant les bases de syntaxe.

Les petites erreurs syntaxiques n'ont pas été retenues par le jury comme un élément discriminatoire, dans la mesure où elles ne cachent pas des erreurs de fond. Les réponses pertinentes d'un point de vue algorithmique sont valorisées.

Les questions relatives aux bases de données, placées en début de sujet, sont relativement bien réussies cette année.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Au regard des copies évaluées, le jury propose aux futurs candidats de prêter attention aux remarques suivantes.

L'indentation en python délimite les blocs d'instructions et doit apparaitre clairement dans la rédaction.

L'initialisation d'une variable dans une boucle ou hors de la boucle n'a pas les mêmes conséquences pour l'algorithme.

Le nombre d'itérations d'une boucle doit être bien réfléchi, en notant que l'instruction `range(n)` parcourt `n` itérations indicées de `0` à `n-1`.

La somme de deux listes `L1+L2` conduit à la concaténation des listes, la somme de deux tableaux (`numpy.ndarray`) `A1+A2` conduit à la somme des éléments du tableau.

La concision et l'élégance des programmes sont appréciées dans l'évaluation. Les candidats qui réinvestissent les fonctions déjà codées sont valorisés par rapport à ceux qui recopient les lignes de code équivalentes. Bien souvent, une condition booléenne bien choisie permet d'éviter de longues listes de conditions aux instructions identiques.

Des noms de variables explicites aident à la compréhension du code. De trop nombreux candidats utilisent des noms de variables non significatifs (`a`, `b`, `c`, ...) ce qui nuit à la compréhension du programme. La clarté du programme (en particulier le choix des noms de variables) ainsi que la présence de commentaires sont prises en compte dans l'évaluation.

Dans une démonstration ou dans l'écriture d'un code, une justification minimale est attendue.

L'ordre des questions importe. Prendre soin de rédiger les réponses aux questions en respectant leur ordre dans le sujet.

La présentation d'une copie fait également partie des compétences attendues d'un candidat à une école d'ingénieur. Le correcteur n'attribue les points qu'aux éléments de réponse qu'il parvient à lire et à comprendre.

Les variables utilisées dans une fonction doivent être définies dans cette fonction ou être explicitement définies comme variables globales (soit par le sujet, soit par le candidat). Beaucoup de candidat ont utilisé la variable `n` sans la définir, ce qui a soulevé une ambiguïté, `n` pouvant être le nombre de vol comme le nombre de sommets.

Les candidats sont invités à lire attentivement l'annexe contenant certaines fonctions utiles pour traiter le sujet.

Enfin, le jury invite les candidats à ne pas se décourager en milieu d'épreuve lorsqu'une question difficile n'est pas réussie. Les sujets comportent de nombreuses parties indépendantes permettant de valoriser les compétences d'un candidat.

Première partie

Les questions sur les bases de données sont abordées par pratiquement tous les candidats, et souvent de façon satisfaisante.

La fonction `count` dans la première question est souvent oubliéee, ou remplacé par autre chose (`len` par exemple) faute de savoir quoi mettre.

La question **I.D** conduit souvent à des jointures inappropriées, en particulier avec la table `aeroport`.

Deuxième partie

Cette partie débute avec des fonctions nécessitant des parcours de listes et des conditions. Les sous-parties ±QII.A et **II.B** sont bien abordées par la plupart des candidats. Le paramètre `n` n'étant pas défini comme une variable globale, il convenait de le définir dans les fonctions à partir des dimensions des tableaux. Certains candidats confondent l'affectation (`=`) et le test d'égalité (`==`). De même, le test de différence a parfois été noté `=!` au lieu de `!=`. Le jury a été indulgent cette année vis-à-vis de ces erreurs, mais souhaite attirer l'attention des candidats sur ces nuances.

Un bon nombre de candidat ne semblent pas à l'aise avec la notation « grand O » et préfèrent parler de complexité en $9n^2$ plutôt qu'en $O(n^2)$.

Les sous-parties **II.C** et **II.D** sont très souvent abordées mais sans recueillir la totalité des points. Certains candidats oublient dans la fonction `cout_du_sommet` de ne pas compter les sommets supprimés, ou au contraire ne comptent pas les sommets non encore attribués.

L'algorithme de recuit simulé (sous-partie **II.D**) montre une forte disparité entre les meilleurs candidats qui proposent un algorithme juste et concis, d'autres qui proposent un algorithme très long suite à des répétitions, et d'autres encore qui n'abordent pas la question. À noter que la syntaxe `liste1 = liste2` en python ne permet pas d'obtenir une copie de `liste2` mais fournit simplement deux noms pour accéder au même objet.

Troisième partie

La sous-partie **III.A** est généralement bien abordée, les candidats sachant plutôt bien comment les nombres sont représentés en mémoire, même si les réponses comportent souvent de petites erreurs de calcul.

Les sous-parties **III.B** et **III.C** s'intéressent au calcul et à la mise à jour de la liste d'intrus, ordonnée en fonction de la dangerosité. Ces questions sont moins abordées et les réponses rarement correctes. Beaucoup de candidats n'ont pas lu l'annexe et n'utilisent pas la fonction `time`.

La question **III.C.2** aborde le cas d'un tri dans une liste déjà triée, où seule une ligne est à replacer. Cette question nécessitait quelques explications sur la stratégie de tri adoptée.

La question **III.C.3** n'a pas été comprise par la plupart des candidats qui l'ont abordée.

La sous-partie **III.D** permet de conclure l'étude sur les performances du TCAS. Ces questions, lorsqu'elles sont abordées, sont souvent correctement traitées.

Conclusion

Le sujet aborde la majeure partie du programme d'informatique commune. Le choix d'un sujet s'appuyant sur une application concrète de l'informatique assure une cohérence avec la formation d'ingénieur. Cette approche sera reconduite sur des problématiques de simulation ou d'algorithmique courantes en informatique, à partir du programme des 3 semestres d'informatique.

Les bons résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont su montrer des compétences affirmées en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

CONCOURS CENTRALE•SUPÉLEC

Informatique

MP,PC,PSI,TSI

3 heures — Calculatrices autorisées

2015

Autour de la dynamique gravitationnelle

Modéliser les interactions physiques entre un grand nombre de constituants mène à l'écriture de systèmes différentiels pour lesquels, en dehors de quelques situations particulières, il n'existe aucune solution analytique. Les problèmes de dynamique gravitationnelle et de dynamique moléculaire en sont deux exemples. Afin d'analyser le comportement temporel de tels systèmes, l'informatique peut apporter une aide substantielle en permettant leur simulation numérique. L'objet de ce sujet, composé de quatre parties, est l'étude de solutions algorithmiques en vue de simuler une dynamique gravitationnelle afin, par exemple, de prédire une éclipse ou le passage d'une comète.

Les programmes doivent être écrits en langage python et les requêtes de base de données en langage SQL. Les candidats sont libres de définir et de programmer toute fonction auxiliaire dont ils estiment avoir besoin pour répondre aux questions posées. Ils veilleront dans ce cas à définir précisément, le rôle de chaque fonction introduite, ses paramètres et son résultat. Ils peuvent également utiliser librement les fonctions de la bibliothèque standard Python, en particulier celles du module `math`.

Lorsque le sujet demande l'écriture d'une fonction python, la réponse doit commencer par l'entête de la fonction (instruction `def`*). D'autre part, si le sujet précise que la fonction prend un paramètre d'un certain type ou qui répond à une certaine condition, la fonction n'a pas à vérifier la conformité de l'argument reçu.*

La lisibilité des codes produits, tant en python qu'en SQL, est un élément important d'appréciation.

I Quelques fonctions utilitaires

I.A – Donner la valeur des expressions python suivantes :

I.A.1) `[1, 2, 3] + [4, 5, 6]`

I.A.2) `2 * [1, 2, 3]`

I.B – Écrire une fonction python `smul` à deux paramètres, un nombre et une liste de nombres, qui multiple chaque élément de la liste par le nombre et renvoie une nouvelle liste : `smul(2, [1, 2, 3])` → `[2, 4, 6]`.

I.C – ***Arithmétique de listes***

I.C.1) Écrire une fonction python `vsom` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes :
`vsom([1, 2, 3], [4, 5, 6])` → `[5, 7, 9]`.

I.C.2) Écrire une fonction python `vdif` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la différence terme à terme de ces deux listes (la première moins la deuxième) : `vdif([1, 2, 3], [4, 5, 6])` → `[-3, -3, -3]`.

II Étude de schémas numériques

Soient y une fonction de classe C^2 sur $\mathbb{R}$ et $t_{\min}$ et $t_{\max}$ deux réels tels que $t_{\min} < t_{\max}$. On note I l'intervalle $[t_{\min}, t_{\max}]$. On s'intéresse à une équation différentielle du second ordre de la forme :

$$\forall t \in I \qquad y''(t) = f(y(t)) \tag{II.1}$$

où f est une fonction donnée, continue sur $\mathbb{R}$. De nombreux systèmes physiques peuvent être décrits par une équation de ce type.

On suppose connues les valeurs $y_0 = y(t_{\min})$ et $z_0 = y'(t_{\min})$. On suppose également que le système physique étudié est conservatif. Ce qui entraine l'existence d'une quantité indépendante du temps (énergie, quantité de mouvement, ...), notée E, qui vérifie l'équation (II.2) où $g' = -f$.

$$\forall t \in I \qquad \frac{1}{2}y'(t)^2 + g(y(t)) = E \tag{II.2}$$

II.A – ***Mise en forme du problème***

Pour résoudre numériquement l'équation différentielle (II.1), on introduit la fonction $z : I \to \mathbb{R}$ définie par $\forall t \in I, z(t) = y'(t)$.

BY-NC-SA

II.A.1) Montrer que l'équation (II.1) peut se mettre sous la forme d'un système différentiel du premier ordre en $z(t)$ et $y(t)$, noté (S).

II.A.2) Soit n un entier strictement supérieur à 1 et $J_n = [\![0, n-1]\!]$. On pose $h = \dfrac{t_{\max} - t_{\min}}{n-1}$ et $\forall i \in J_n$, $t_i = t_{\min} + ih$. Montrer que, pour tout entier $i \in [\![0, n-2]\!]$,

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\,\mathrm{d}t \qquad \text{et} \qquad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\,\mathrm{d}t \tag{II.3}$$

La suite du problème exploite les notations introduites dans cette partie et présente deux méthodes numériques dans lesquelles les intégrales précédentes sont remplacées par une valeur approchée.

II.B – Schéma d'Euler explicite

Dans le schéma d'Euler explicite, chaque terme sous le signe intégrale est remplacé par sa valeur prise en la borne inférieure.

II.B.1) Dans ce schéma, montrer que les équations (II.3) permettent de définir deux suites $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$, où y_i et z_i sont des valeurs approchées de $y(t_i)$ et $z(t_i)$. Donner les relations de récurrence permettant de déterminer les valeurs de y_i et z_i connaissant y_0 et z_0.

II.B.2) Écrire une fonction `euler` qui reçoit en argument les paramètres qui vous semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i\in J_n}$ et $(z_j)_{i\in J_n}$. Vous justifierez le choix des paramètres transmis à la fonction.

II.B.3) Pour illustrer cette méthode, on considère l'équation différentielle $\forall t \in I, y''(t) = -\omega^2 y(t)$ dans laquelle ω est un nombre réel.

a) Montrer qu'on peut définir une quantité E, indépendante du temps, vérifiant une équation de la forme (II.2).

b) On note E_i la valeur approchée de E à l'instant t_i, $i \in J_n$, calculée en utilisant les valeurs approchées de $y(t_i)$ et $z(t_i)$ obtenues à la question II.B.1. Montrer que $E_{i+1} - E_i = h^2\omega^2 E_i$.

c) Qu'aurait donné un schéma numérique qui satisfait à la conservation de E ?

d) En portant les valeurs de y_i et z_i sur l'axe des abscisses et l'axe des ordonnées respectivement, quelle serait l'allure du graphe qui respecte la conservation de E ?

e) La mise en œuvre de la méthode d'Euler explicite génère le résultat graphique donné figure 1 à gauche. Dans un système d'unités adapté, les calculs ont été menés en prenant $y_0 = 3$, $z_0 = 0$, $t_{\texttt{min}} = 0$, $t_{\texttt{max}} = 3$, $\omega = 2\pi$ et $n = 100$.

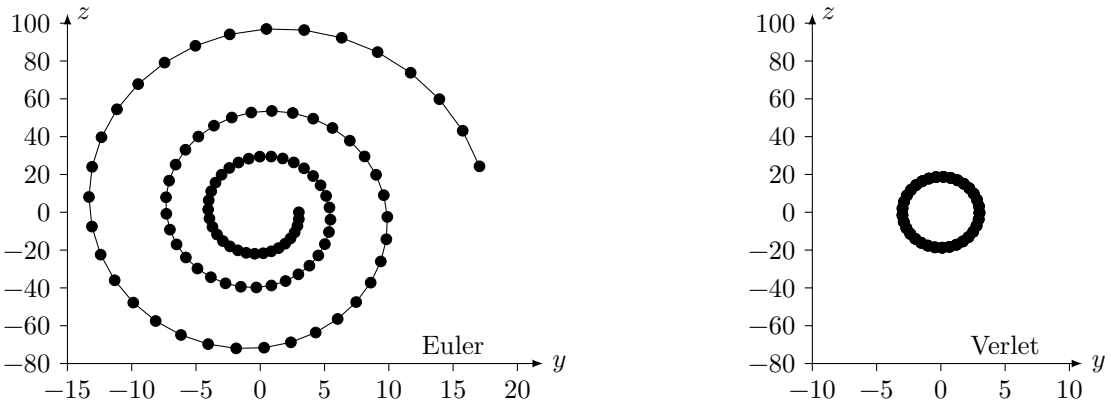

Figure 1

En quoi ce graphe confirme-t-il que le schéma numérique ne conserve pas E ? Pouvez-vous justifier son allure ?

II.C – Schéma de Verlet

Le physicien français Loup Verlet a proposé en 1967 un schéma numérique d'intégration d'une équation de la forme (II.1) dans lequel, en notant $f_i = f(y_i)$ et $f_{i+1} = f(y_{i+1})$, les relations de récurrence s'écrivent

$$y_{i+1} = y_i + hz_i + \frac{h^2}{2} f_i \qquad \text{et} \qquad z_{i+1} = z_i + \frac{h}{2}\left(f_i + f_{i+1}\right)$$

II.C.1) Écrire une fonction `verlet` qui reçoit en argument les paramètres qui vous semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i\in J_n}$ et $(z_j)_{i\in J_n}$.

II.C.2) On reprend l'exemple de l'oscillateur harmonique (question II.B.3) et on compare les résultats obtenus à l'aide des schémas d'Euler et de Verlet.

a) Montrer que dans le schéma de Verlet, on a $E_{i+1} - E_i = O(h^3)$.

BY-NC-SA

b) La mise en œuvre du schéma de Verlet avec les mêmes paramètres que ceux utilisés au II.B.3e donne le résultat de la figure 1 à droite. Interpréter l'allure de ce graphe.

c) Que peut-on conclure sur le schéma de Verlet ?

III Problème à N corps

On s'intéresse à présent à la dynamique d'un système de N corps massifs en interaction gravitationnelle. Dans la suite, les corps considérés sont assimilés à des points matériels P_j de masses m_j où $j \in [\![0, N-1]\!]$, $N \geqslant 2$ étant un entier positif donné. Le mouvement de ces points est étudié dans un référentiel galiléen muni d'une base orthonormée. L'interaction entre deux corps j et k est modélisée par la force gravitationnelle. L'action exercée par le corps k sur le le corps j est décrite par la force $\vec{F}_{k/j} = G\dfrac{m_j m_k}{r_{jk}^3}\overrightarrow{P_jP_k}$ où r_{jk} est la distance séparant les corps j et k ($r_{jk} = \|\overrightarrow{P_jP_k}\|$) et $G = 6{,}67 \times 10^{-11}$ N·m^2·kg^{-2} la constante de gravitation universelle.

À tout instant t_i avec $i \in [\![0, n]\!]$, chaque corps de masse m_j est repéré par ses coordonnées cartésiennes (x_{ij}, y_{ij}, z_{ij}) et les composantes de son vecteur vitesse $(v_{xij}, v_{yij}, v_{zij})$ dans le référentiel de référence.

Trois listes sont utilisées pour représenter ce système en python

- `masse` conserve les masses de chaque corps : `masse[j]` = m_j ;
- `position` contient les positions successives de chaque corps : `position[i][j]` = $[x_{ij}, y_{ij}, z_{ij}]$;
- `vitesse` mémorise les vitesses successives de chaque corps : `vitesse[i][j]` = $[v_{xij}, v_{yij}, v_{zij}]$.

L'objet de la suite du problème est de construire ces listes en mettant en œuvre l'algorithme de Verlet.

III.A – Position du problème

III.A.1) Exprimer la force $\vec{F}_j$ exercée sur le corps P_j par l'ensemble des autres corps P_k, avec $k \neq j$.

III.A.2) Écrire une fonction python `force2(m1, p1, m2, p2)` qui prend en paramètre les masses (`m1` et `m2` en kilogrammes) et les positions (`p1` et `p2`, sous forme de listes de trois coordonnées cartésiennes en mètres) de deux corps 1 et 2 et qui renvoie la valeur de la force exercée par le corps 2 sur le corps 1, sous la forme d'une liste à trois éléments représentant les composantes de la force dans la base de référence, en newtons.

III.A.3) Écrire une fonction `forceN(j, m, pos)` qui prend en paramètre l'indice `j` d'un corps, la liste des masses des N corps du système étudié ainsi que la liste de leurs positions et qui renvoie $\vec{F}_j$, la force exercée par tous les autres corps sur le corps `j`, sous la forme d'une liste de ses trois composantes cartésiennes.

III.B – Approche numérique

III.B.1) Expliciter la structure et la signification de `position[i]` et `vitesse[i]`.

III.B.2) Écrire une fonction `pos_suiv(m, pos, vit, h)` qui prend en paramètres la liste des masses des N corps du système étudié (en kilogrammes), la liste de leurs positions (en mètres) à l'instant t_i, la liste de leurs vitesses (en mètres par seconde) au même instant et le pas d'intégration h (en secondes) et qui renvoie la liste des positions des N corps à l'instant t_{i+1} calculées en utilisant le schéma de Verlet.

III.B.3) Écrire une fonction `etat_suiv(m, pos, vit, h)` qui prend les mêmes paramètres que la fonction `pos_suiv` et qui renvoie la liste des positions (en mètres) et la liste des vitesses (en m/s) des N corps à l'instant t_{i+1} calculées en utilisant le schéma de Verlet.

III.B.4) En notant τ_N la durée des calculs pour un nombre N de corps, la mise en œuvre de la fonction `etat_suiv` a donné le résultat graphique de la figure 2 où on a porté $\ln(N)$ en abscisse et $\ln(\tau_N)$ en ordonnée.

a) Quelle relation simple peut-on établir entre $\ln(\tau_N)$ et $\ln(N)$ à partir de la figure 2 ?

b) Quelle hypothèse peut-on émettre quant à la complexité de l'algorithme étudié ?

III.B.5)

a) Estimer la complexité temporelle de la fonction `etat_suiv` sous la forme $O(N^\alpha)$.

b) Comparer avec le résultat obtenu à la question III.B.4.

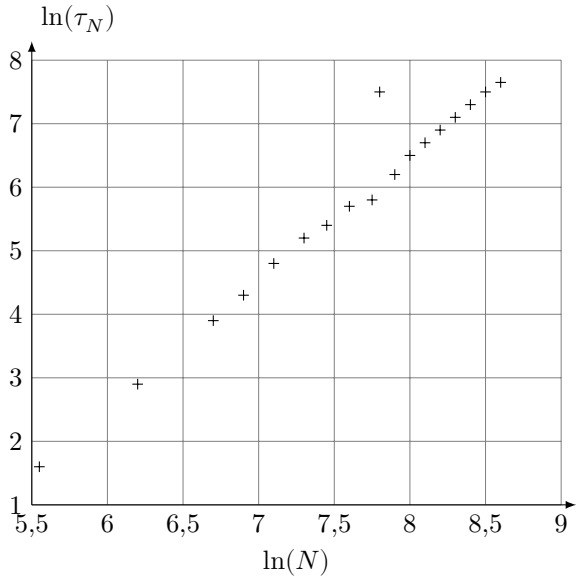

Figure 2

BY-NC-SA

IV Exploitation d'une base de données

À partir de mesures régulièrement effectuées par différents observatoires, une base de données des caractéristiques et des états des corps célestes de notre Système solaire est maintenue à jour. L'objectif de cette partie est d'extraire de cette base de données les informations nécessaires à la mise en œuvre des fonctions développées dans les parties précédentes, puis de les utiliser pour prévoir les positions futures des différentes planètes. Les données à extraire sont les masses des corps étudiés et leurs états (position et vitesse) à l'instant t_{min} du début de la simulation.

Une version simplifiée, réduite à deux tables, de la base de données du Système solaire est donnée figure 3. Les masses sont exprimées en kilogrammes, les distances en unités astronomiques ($1\,\text{au} = 1{,}5 \times 10^{11}\,\text{m}$) et les vitesses en kilomètres par seconde. Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son centre est situé à proximité du Soleil.

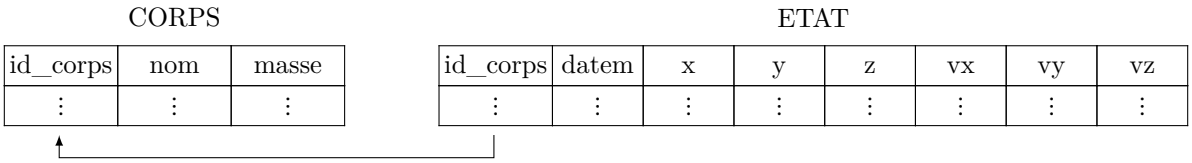

Figure 3 Schéma de la base de données

La table `CORPS` répertorie les corps étudiés, elle contient les colonnes
- `id_corps` (clé primaire) entier identifiant chaque corps ;
- `nom`, chaine de caractères, désigne le nom usuel du corps ;
- `masse` de type flottant, contient la masse du corps.

La table `ETAT` rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle est constituée de huit colonnes :
- `id_corps` de type entier, identifie le corps concerné ;
- `datem` est la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;
- trois colonnes de type flottant pour les composantes de la position `x`, `y`, `z` ;
- trois colonnes de type flottant pour les composantes de la vitesse `vx`, `vy`, `vz`.

IV.A – Écrire une requête SQL qui renvoie la liste des masses de tous les corps étudiés.

IV.B – Les états des différents corps ne sont pas forcément tous déterminés exactement au même instant. Nous allons assimiler l'état initial (à la date t_{min}) de chaque corps à son dernier état connu antérieur à t_{min}.

Dans toute la suite, on supposera que la valeur de t_{min}, sous le format utilisé dans la table `ETAT`, est accessible à toute requête SQL via l'expression `tmin()`.

IV.B.1) On souhaite d'abord vérifier que tous les corps étudiés disposent d'un état connu antérieur à `tmin()`.

Le nombre de corps présents dans la base est obtenu grâce à la requête `SELECT count(*) FROM corps`. Écrire une requête SQL qui renvoie le nombre de corps qui ont au moins un état connu antérieur à `tmin()`.

IV.B.2) Écrire une requête SQL qui renvoie, pour chaque corps, son identifiant et la date de son dernier état antérieur à `tmin()`.

IV.B.3) Le résultat de la requête précédente est stocké dans une nouvelle table `date_mesure` à deux colonnes :
- `id_corps` de type entier, contient l'identifiant du corps considéré ;
- `date_der` de type entier, correspond à la date du dernier état connu du corps considéré, antérieur à `tmin()`.

Pour simplifier la simulation, on décide de négliger l'influence des corps ayant une masse strictement inférieure à une valeur fixée `masse_min()` et de ne s'intéresser qu'aux corps situés dans un cube, centré sur l'origine du référentiel de référence et d'arête `arete()` donnée. Les faces de ce cube sont parallèles aux plans formés par les axes du référentiel de référence.

Écrire une requête SQL qui renvoie la masse et l'état initial (sous la forme `masse, x, y, z, vx, vy, vz`) de chaque corps retenu pour participer à la simulation. Classez les corps dans l'ordre croissant par rapport à leur distance à l'origine du référentiel.

IV.C – On dispose des variables python `t0`, `p0`, `v0` et `masse` initialisées à partir du résultat de la requête précédente. `t0` est un entier qui donne la date des conditions initiales : il correspond à t_{min} et à `tmin()`. `p0` est une liste de longueur N, chaque élément de `p0` est une liste à 3 éléments de la forme `[x, y, z]` représentant la position initiale d'un corps, en unité astronomique. `v0` a une structure identique mais indique les vitesses initiales des corps considérés, en km/s. `masse` est décrite en partie III.

Écrire la fonction python `simulation_verlet(deltat, n)` qui prend en paramètre un incrément de temps en secondes (`deltat > 0`) et un nombre d'itérations (`n > 0`) et qui renvoie la liste des positions des corps considérés pour chaque instant `t0`, `t0 + deltat`, ..., `t0 + n*deltat` (cf variable `position` définie en partie III). Les calculs seront menés en utilisant le schéma d'intégration de Verlet, le résultat sera fourni en unité astronomique.

• • • FIN • • •

BY-NC-SA

II.B – La vitesse limite peut être très distrayante : le record est détenu par un grain de sable supraluminique à $8{,}5 \times 10^{12}\,\mathrm{m{\cdot}s^{-1}}$!

II.C – Le nombre de Reynolds n'est pas connu de tous. Le jury accepte évidemment tout aussi bien l'utilisation de la viscosité cinématique que de la viscosité dynamique, mais la longueur caractéristique de l'écoulement autour du grain ne peut être la profondeur du bac de décantation.

II.D – Un calcul exact de $r_{\min}$ est souvent proposé. Une exploitation du tableau, pour peu que celui-ci ait été convenablement complété, est aussi acceptée. En revanche, sur une telle question comme sur la précédente, il est évident qu'un nombre excessif de chiffres significatifs est pénalisé.

II.E – Beaucoup de trajectoires sont proposées, parfois fantaisistes, alors qu'une simple composition des vitesses suffisait. Notons que des explications du style : « il y a sédimentation si la bille n'a pas le temps de faire le trajet avant de sédimenter » laissent rêveurs.

III. Décantation des boues résiduelles

III.A.1) Si le principe du bilan est majoritairement compris, la démonstration s'accommode souvent d'incohérences sur les signes.

III.A.2) La résolution de l'équation différentielle est l'occasion de solutions surprenantes. Certains candidats ont aussi oublié de se demander si le rapport v_ℓ/D qu'ils proposent est bien homogène à une longueur. L'énergie potentielle présente dans le facteur de Boltzmann doit ressembler à quelque chose comme m^*gz : le jury s'est réjoui de voir le lien établi de temps à autre, mais trop rarement à notre goût ! La valeur numérique de λ réserve encore bien des surprises. Une petite minorité constate correctement qu'il est superflu de prendre en compte la diffusion, en cohérence avec la suite de l'énoncé.

III.B.1) On trouve parfois $x = n^*/V$, mais est-ce bien homogène ? Le tracé de la courbe $v(n^*)$ fait apparaitre — comme sur certaines autres courbes — une confiance aveugle vis à vis de la calculatrice : il n'est ainsi pas rare d'observer des densités volumiques négatives !

III.B.2) Cette question ne présentait pas de difficulté particulière, et pourtant peu de candidats ont su exploiter la figure à laquelle il était fait référence.

III.B.3) Cette partie, arrivant en fin de problème, a semblé beaucoup plus délicate, le jury a fait preuve d'une plus grande mansuétude. Ceci étant, les candidats qui se lancent dans un bilan, quel qu'il soit, sont invités à bien définir leur système, ce qui n'est pas toujours le cas.

III.B.4) Cette partie n'a été traitée que de façon anecdotique, ce qui ne permet pas d'en tirer des conclusions générales. Le jury s'est surtout attaché à la cohérence des résultats, qui devaient en particulier tenir compte du signe.

Conclusions

Les remarques précédentes peuvent paraitre excessivement pessimistes, aussi le jury tient à féliciter les candidats qui ont su l'impressionner : qu'ils en soient remerciés !

« Ce qui se conçoit bien s'énonce clairement et les mots pour le dire arrivent aisément. » Tout le monde connait bien ce vieil adage et devrait le faire sien ! En effet, le concours s'efforce de faire émerger les talents de demain, et comment reconnait-on un ingénieur talentueux, si ce n'est par ce qu'il produit, par les problématiques qu'il résout ? Ainsi, il aura non seulement besoin de trouver des solutions inédites, mais aussi de se faire comprendre en utilisant les outils adéquats, qu'il saura manier avec soin. Il expliquera alors le plus clairement possible la démarche dans laquelle il s'inscrit.

C'est en ce sens que le futur candidat est engagé à travailler : après s'être approprié les problématiques essentielles du sujet (et avoir lu les questions dans leur intégralité), il doit s'attacher à analyser la situation à la lumière des différentes compétences qu'il a acquises tout au long de sa formation. Celles-ci doivent lui permettre de résoudre la problématique qui s'offre à lui. Mais son travail ne s'arrête pas là ! Tout candidat sérieux doit penser à valider son ouvrage grâce à son bon sens. Des arguments d'homogénéité s'imposent d'emblée. Ceci lui permettra d'éviter bien des errements.

Évidemment, il lui est aussi demandé de communiquer ! Il serait dommage de négliger cet aspect absolument fondamental : un ingénieur se doit d'être clair, précis, rigoureux, honnête, humble. Voilà quelques-unes des qualités qui feront d'un étudiant préparationnaire un bon candidat et un bon ingénieur.

Nous invitons chacun à relire les programmes officiels qui permettent de bien saisir l'esprit dans lequel nous cherchons à travailler, en souhaitant à chaque candidat de faire fructifier au mieux ses talents !

Concours Centrale-Supélec 2015 filière PC

Chimie

Présentation du sujet

Le sujet de cette année présente le procédé Wacker, la réaction de Heck (prix Nobel 2010) et une synthèse présentant une étape d'oxydation chimio-sélective (réaction de Wacker-Tsuji), trois réactions importantes qui utilisent des complexes du palladium comme catalyseurs.

Les notions mises en jeu font appel à de nombreux domaines abordés dans le programme de première et de seconde années des classes préparatoires (architecture de la matière condensée, transformations chimiques en solution aqueuse, orbitales moléculaires, cycles catalytiques...).

Le sujet comporte à la fois des questions « simples » (questions d'application directe du cours), des études nécessitant davantage de réflexion (questions « complexes » et plus ouvertes) et des problématiques directement liés au domaine expérimental. Il permet de valoriser la réflexion des candidats plutôt que leur technicité calculatoire. Plusieurs approches documentaires (extrait du site de Wikipédia sur les caractéristiques du palladium, de revue et d'ouvrages spécialisés sur la formation de liaisons carbone-carbone) permettent d'évaluer les capacités des candidats à analyser, ainsi qu'à synthétiser l'information.

Les compétences évaluées dans cette épreuve sont :

— décrire la mise en œuvre de quelques techniques de laboratoire et analyser l'influence de quelques paramètres physico-chimiques des processus mis en jeu lors de procédés industriels ou de synthèses au laboratoire. Ainsi sont décrites et analysées les conditions dans lesquelles est réalisé le procédé Wacker en termes de « chimie durable », la mise en œuvre des opérations de lavage et de purification lors d'une synthèse organique, la détermination du rendement d'obtention du produit recherché ;

— étudier l'influence de la structure chimique des réactifs utilisés et des conditions expérimentales dans une stratégie de synthèse. Par exemple, est étudiée l'influence de la structure du réactif — symétrique ou pas, bifonctionnel ou pas — sur la nécessité de réaliser des étapes de protection-déprotection ;

— utiliser des modèles théoriques permettant d'analyser la réactivité des substrats ou d'écrire quelques mécanismes réactionnels ; le diagramme d'orbitales moléculaires permet ainsi d'analyser la stabilité et la réactivité d'un complexe du palladium vis-à-vis de substrats électrophiles, l'étude des interactions orbitalaires doit permettre de comprendre la modification de la réactivité d'un alcène par coordination ;

— maitriser le vocabulaire scientifique dans la description des phénomènes étudiés : analyser la chimiosélectivité d'une transformation, nommer les étapes d'un cycle catalytique par exemple.

Analyse globale des résultats

Sur l'ensemble des copies, au moins une bonne réponse a été apportée à chaque question.

La description et l'analyse des techniques ou résultats expérimentaux ne sont pas menées avec une rigueur suffisante. Trop peu de candidats sont capables d'expliquer précisément l'intérêt des opérations de lavage (le rôle du chlorure de sodium dans l'opération de relargage, la nature des bases