

INFORMATIQUE 2019



Union des Professeurs
de classes préparatoires
Scientifiques

Table des matières

Option informatique	1
X – ENS A (XULCR) (4h) [i193m1e]	
<i>Autour des sous-mots et des sur-mots</i>	1
Mines Ponts (3h) [i19mmoe]	
<i>Morphismes d'automates</i>	8
Centrale Supélec (4h) [i19cmoe]	
<i>Code de Gray et problème du sac à dos</i>	19
CCINP (4h) [i19pmoe]	
<i>Partie I (informatique commune) : inversions de permutations</i>	
<i>Partie II (option informatique) : automates et langages rationnels</i>	
<i>Partie III (option informatique) : mots sans facteur carré</i>	22
E3A (3h) [i19rmoe]	
<i>Recherche dichotomique, automates et langages de mots binaires, diamètre d'un graphe</i>	30
CAPES externe de Mathématiques, option Informatique [i19c3oe]	
<i>Base 2 équilibrée, compilation et algorithme</i>	38
Informatique commune	46
X – ENS B (XELCR) (2h) [i193m2e]	
<i>Tetris couleurs</i>	46
Mines Ponts (1,5h) [i19mice]	
<i>Autour des nombres premiers</i>	62
Centrale Supélec - MP, PC, PSI (3h) [i19cice]	
<i>Élasticité d'un brin d'ADN</i>	72
CCINP - PSI (3h) [i19psce]	
<i>Analyse de données, méthode knn, méthode Naive Bayes</i>	82
CCINP - TSI (3h) [i19pice]	
<i>Distance de Levenshtein, base de données, codage de couleurs et de l'information</i>	94
Modélisation	118
CCINP - PC (4h) [d19ppue]	
<i>Mouvement d'une plateforme en mer</i>	118
PT (4h) [d19dtue]	
<i>Figures de Chladni</i>	130

En guise d'introduction

Mais dis donc, on n'est quand même pas venus pour beurrer des sandwichs.
Michel Audiard – Les tontons flingueurs

Le recueil 2019 comporte trois parties, la première contenant les épreuves de l'option informatique de nos classes ainsi que quelques autres sujets, la deuxième contenant les épreuves d'informatique commune et la troisième contenant deux sujets de modélisation.

Le bulletin des concours Informatique n'est proposé aux adhérents que sous forme électronique. Dans ce recueil, le nom du fichier contenant chaque sujet figure dans la table des matières et en tête de chaque page. Les fichiers sont disponibles sur le site de l'UPS à l'adresse suivante.

<http://prepas.org/ups.php?module=Maths&voir=recherche>

Ce bulletin et ses prédecesseurs le sont dans la rubrique *Ressources et partage* (<http://prepas.org/ups.php?entree=partage>) de l'*Espace adhérents et associés* (<http://prepas.org/index.php?rubrique=4>) ou en suivant le lien direct suivant.

<http://prepas.org/ups.php?rubrique=146>

Un grand merci à toutes celles et tous ceux qui ont apporté leur aide à la conception de ce bulletin.

Laurent Sartre
laurent.sartre@prepas.org
Octobre 2019

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation sera **obligatoirement OCaml**.

Autour des sous-mots et des sur-mots

Préliminaires

Un mot est une suite de lettres $a_0 \cdots a_{n-1}$ tirées d'un alphabet fini $A = \{\mathbf{a}, \mathbf{b}, \dots\}$. On utilisera $u, v, u', u'', u_1, u_2, \dots$ pour dénoter les éléments de A^* , c.-à-d. les mots sur A . On note ε pour le mot vide et $|u|$ pour la longueur de u , de sorte que $|\varepsilon| = 0$.

Si un mot u se décompose sous la forme $u = u_1 vu_2$, alors v est un **facteur** de u , et même un préfixe (ou un suffixe) si $u_1 = \varepsilon$ (ou si $u_2 = \varepsilon$) dans cette décomposition. Dans le cas d'un mot $u = a_0 \cdots a_{n-1}$ on écrit « $u[i, j[$ », sous la condition $0 \leq i \leq j \leq n$, pour désigner le facteur $a_i \cdots a_{j-1}$. Cette notation s'étend à $u[i \cdots [$ et $u[i]$ pour désigner, respectivement, $u[i, n[$ et $u[i, i+1[$.

Ce que l'on appelle sous-mot de u correspond à la notion classique de sous-suite, ou de suite extraite, et ne doit pas être confondu avec un facteur. Pour $u = a_0 \cdots a_{n-1}$, on dira qu'un mot v de longueur m est un **sous-mot** de u , ce que l'on notera $v \preceq u$, s'il existe une suite strictement croissante $0 \leq p_0 < p_1 < \cdots < p_{m-1} < n$ telle que $v = a_{p_0} a_{p_1} \cdots a_{p_{m-1}}$. Par exemple, `caml` \preceq `bechame1`. Formellement, pour tout $n \in \mathbb{N}$, nous noterons $[n]$ pour l'ensemble $\{0, 1, 2, \dots, n-1\}$, de sorte que la suite p_0, p_1, \dots, p_{m-1} peut être vue comme une application strictement croissante $p : [m] \rightarrow [n]$. Pour une telle application, on note $v = u \circ p$ pour dire que v est le sous-mot extrait de u via p et on dit que p est un **plongement** de v dans u , noté $p : v \preceq u$. Notons qu'il peut exister plusieurs façons différentes de plonger v dans u .

Notre objectif ici est de développer des algorithmes impliquant à divers titres la notion de sous-mot : recherche d'un sous-mot à l'intérieur d'un texte, dénombrement des sous-mots, raisonnement sur l'ensemble des sous-mots d'un texte ou d'un langage.

Complexité. Par **complexité en temps** d'un algorithme A on entend le nombre d'opérations élémentaires (comparaison, addition, soustraction, multiplication, division, affectation, test, etc.) nécessaires à l'exécution de A dans le cas le pire. Par **complexité en espace** d'un algorithme A on entend l'espace mémoire minimal nécessaire à l'exécution de A dans le cas le pire. Lorsque la complexité en temps ou en espace dépend d'un ou plusieurs paramètres $\kappa_0, \dots, \kappa_{r-1}$, on dit que A a une complexité en $\mathcal{O}(f(\kappa_0, \dots, \kappa_{r-1}))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $\kappa_0, \dots, \kappa_{r-1}$ suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres $\kappa_0, \dots, \kappa_{r-1}$, la complexité est au plus $C f(\kappa_0, \dots, \kappa_{r-1})$.

OCaml. On rappelle quelques éléments du langage OCaml qui peuvent être utiles. Une chaîne de caractères s a le type `string`, sa longueur est obtenue avec `String.length s` et son i -ième caractère avec $s.[i]$, les caractères étant indexés à partir de 0. Un tableau t a le type τ `array`, où τ est le type des éléments, et sa longueur est obtenue avec `Array.length t`. Son i -ième élément est obtenu avec `t.(i)` et modifié avec `t.(i) <- val`, les éléments étant indexés à partir de 0. L'expression `Array.make n val` construit un tableau de taille n dont les éléments sont initialisés avec la valeur `val`. En OCaml, une matrice est un tableau de tableaux de même taille. L'expression `Array.make_matrix n m val` construit une matrice de n lignes et m colonnes, dont les éléments sont tous initialisés avec la valeur `val`. Le candidat est libre d'utiliser tout autre élément du langage OCaml et de sa bibliothèque standard.

Question 1. a. Montrez que pour deux mots u et u' et deux lettres a et a' , on a l'équivalence suivante :

$$ua \preccurlyeq u'a' \iff ua \preccurlyeq u' \text{ ou } (a = a' \text{ et } u \preccurlyeq u'). \quad (1)$$

b. Programmez une fonction OCaml `teste_sous_mot : string -> string -> bool` décidant en temps polynomial si un mot v est sous-mot d'un mot u . Détaillez et justifiez votre analyse de complexité.

I. Compter et construire

On note $\binom{u}{v}$ le nombre de plongements de v dans u , de sorte que $v \preccurlyeq u$ si et seulement si $\binom{u}{v} > 0$. Notons en particulier que $\binom{u}{\epsilon} = 1$ pour tout mot $u \in A^*$ car il n'existe qu'une injection de $[0]$, c.-à-d. \emptyset , dans $\{0, 1, \dots, |u| - 1\}$ et cette injection est bien un plongement.

Question 2. a. Montrez que $\binom{abab}{ab} = 3$.

b. Que vaut $\binom{a^n}{a^m}$ quand $a \in A$ est une lettre ?

On rappelle que a^n est le mot constitué de n occurrences de la lettre a .

c. Montrez que $\binom{ua}{va} = \binom{u}{va} + \binom{u}{v}$ pour tous mots $u, v \in A^*$ et toute lettre $a \in A$.

Question 3. Pour calculer $\binom{u}{v}$ on considère la fonction OCaml suivante.

```
let nb_plongements (v:string) (u:string) =
  let rec aux i j =
    if i = 0 then 1
    else if j = 0 then 0
    else if v.[i-1] = u.[j-1] then (aux (i - 1) (j - 1)) + (aux i (j - 1))
    else aux i (j - 1)
  in
  aux (String.length v) (String.length u)
```

a. Prouvez sa terminaison.

b. Justifiez sa correction, c.-à-d., expliquez pourquoi elle renvoie bien la valeur $\binom{u}{v}$.

On note $T(v, u)$ le nombre de fois où la fonction `aux` est appelée lors du calcul de `nb_plongements v u`.

- Question 4.** **a.** Montrez qu'il existe une constante C_1 telle que $T(v, u) < 2^{|u|} \cdot C_1$.
b. Montrez que l'on ne peut pas majorer $T(v, u)$ par une fonction polynomiale de $\binom{u}{v}$.
c. Montrez qu'il existe une constante C_2 telle que $T(v, u) \geq 2\binom{u}{v} + C_2$.

La question précédente a montré que la fonction `nb_plongements` proposée dans le sujet demande un temps de calcul parfois exponentiel en la taille $|u| + |v|$ de ses arguments. De meilleurs algorithmes existent...

Question 5. Programmez en OCaml une nouvelle fonction `nb_plongements_rapide : string -> string -> int` qui calcule $\binom{u}{v}$ en temps polynomial en $|u| + |v|$. Détaillez votre analyse de complexité en temps et en espace.

Indication : on pourra utiliser la programmation dynamique.

On cherche maintenant à dénombrer les sous-mots d'un mot u . On note $\downarrow u$ pour $\{v \mid v \preceq u\}$. Il s'agit d'un langage fini. Par exemple $\downarrow abab = \{\varepsilon, a, b, ab, aa, ba, bb, aab, aba, abb, bab, abab\}$ de sorte que $abab$ a 12 sous-mots distincts, ce que l'on note $\text{Card}(\downarrow abab) = 12$.

Les langages étant des ensembles (des parties L, L', \dots de A^*), on utilisera les notations $L \cup L'$, $L \setminus L'$, etc. avec leur signification ensembliste habituelle. On utilise aussi la notation $L \cdot L'$ pour désigner le produit de concaténation de deux langages : $L \cdot L' = \{uv \mid u \in L, v \in L'\}$. Dans le cas d'un singleton $L = \{u\}$, on écrit souvent $u \cdot L'$ au lieu de $\{u\} \cdot L'$.

Question 6. **a.** Montrez que, pour tous mots v, w et toute lettre a , on a

$$\downarrow wava = \downarrow wav \cup (\downarrow wav \setminus \downarrow w) \cdot a. \quad (\ddagger)$$

b. Montrer que l'union $\downarrow wav \cup (\downarrow wav \setminus \downarrow w) \cdot a$ est disjointe si et seulement si le mot v ne contient pas la lettre a .

Quand l'union est disjointe dans l'équation (\ddagger) , on peut obtenir $\text{Card}(\downarrow u)$, pour $u = wava$, en combinant $\text{Card}(\downarrow wav)$ et $\text{Card}(\downarrow w)$. Cette approche se généralise au cas d'un mot u quelconque.

Question 7. **a.** Donnez des équations récursives permettant de calculer $\text{Card}(\downarrow u)$ en se ramenant à des préfixes de u . On pourra considérer par exemple les diverses occurrences de la dernière lettre de u quand elle existe.

b. En se basant sur vos équations, programmez une fonction OCaml `nb_sousmots : string -> int` qui, pour un mot u donné, calcule $\text{Card}(\downarrow u)$ en temps polynomial en $|u|$. Justifiez votre analyse de complexité.

Un **sur-mot** commun à u et v est un mot w tel que $u \preceq w$ et $v \preceq w$. Il existe une infinité de tels mots. Parmi tous ces sur-mots communs à u et v , on s'intéresse à celui qui est le plus court, et qui est le premier dans l'ordre lexicographique pour départager les sur-mots communs de même longueur. Ce mot est noté $\text{pcsmc}(u, v)$ et par exemple $\text{pcsmc}(\text{informatique}, \text{difficile}) = \text{difnficormatilque}$.

- Question 8.**
- Soient a, b deux lettres distinctes. Montrez que si $\text{pcsmc}(ua, vb) = wa$ alors $w = \text{pcsmc}(u, vb)$, ceci pour tous mots u, v, w .
 - Généralisez la propriété précédente en donnant des équations qui permettent de caractériser $\text{pcsmc}(ua, vb)$ dans le cas général, y compris quand $a = b$.
 - Programmez une fonction OCaml calculant $\text{pcsmc}(u, v)$ en temps polynomial en $|u| + |v|$ pour des mots u et v arbitraires. Détaillez votre analyse de complexité.

II. Sous-mots et expressions rationnelles

On rappelle que les **expressions rationnelles** sont écrites à partir des expressions de base « ε », « \emptyset », ainsi que les lettres « a », « b », ..., que l'on peut combiner au moyen des opérateurs binaires « $+$ » et « \cdot » (désignant l'union et la concaténation de langages) ainsi que de l' $'\ast$ étoile de Kleene », un opérateur unaire « \ast » noté en exposant.

Le langage représenté par une expression rationnelle e est défini inductivement par $L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, ..., $L(e + e') = L(e) \cup L(e')$, $L(e \cdot e') = L(e) \cdot L(e')$ et enfin

$$L(e^*) = L(e)^* = \{\varepsilon\} \cup L(e) \cup L(e) \cdot L(e) \cup \dots = \bigcup_{i \in \mathbb{N}} \overbrace{L(e) \cdot L(e) \cdots L(e)}^i.$$

Pour manipuler des expressions rationnelles, on utilisera la définition OCaml suivante :

```
type ratexp =
  | Epsilon
  | Empty
  | Letter of char
  | Sum of ratexp * ratexp
  | Product of ratexp * ratexp
  | Star of ratexp
```

Par exemple, les expressions rationnelles $a \cdot (b + c)^*$ et $((\emptyset + \varepsilon)^*)^*$ seront représentées par

```
let e_exmp1 = Product (Letter 'a', Star (Sum (Letter 'b', Letter 'c'))))
let e_exmp2 = Star (Star (Sum (Empty, Epsilon)))
```

La taille d'une expression rationnelle, notée $|e|$, est le nombre de constructeurs apparaissant dans l'expression. On pourrait calculer $|e|$ en OCaml au moyen du code suivant :

```
let rec taille_ratexp (e : ratexp) =
  match e with
  | Empty -> 1 | Epsilon -> 1 | Letter _ -> 1
  | Sum (e1,e2) -> 1 + taille_ratexp(e1) + taille_ratexp(e2)
  | Product (e1,e2) -> 1 + taille_ratexp(e1) + taille_ratexp(e2)
  | Star (e1) -> 1 + taille_ratexp(e1)
```

Question 9. On définit les expressions rationnelles e_1 et e_2 par

```
let e1 = Product (Star (Product (Sum (Letter 'a', Empty), Letter 'c')),
                  Product (Letter 'b'),
                  Product (Empty,
                           Star (Product (Letter 'c', Letter 'c')))))
let e2 = Product (Star (Product (Letter 'b', Letter 'a'))),
                  Product (Sum (Epsilon, Letter 'a'), Star (Letter 'c')))
```

Pour chacun des langages $L(e_1)$ et $L(e_2)$, dites s'il contient un mot commençant par **a**; par **b**; par **c**.

Question 10. Programmez une fonction OCaml `peut_debuter_par : ratexp -> char -> bool` testant, pour une expression rationnelle e et une lettre a , si $L(e)$ contient un mot commençant par a .

Pour un langage L , on définit $\downarrow L = \bigcup_{w \in L} \downarrow w$. On s'intéresse maintenant à la question de savoir, pour un mot u et une expression rationnelle e , si u est dans $\downarrow L(e)$, c.-à-d. si u est sous-mot d'un des mots définis par e . Une solution possible passe par des calculs de résidus de langages. Formellement, pour un langage $L \subseteq A^*$ et un mot $u \in A^*$, on définit le **résidu de L par u** comme

$$\langle u \rangle L = \{v \in A^* \mid \exists w \text{ tel que } u \preceq w \text{ et } wv \in L\}.$$

Ainsi, u est sous-mot d'un mot de L si et seulement si $\varepsilon \in \langle u \rangle L$. Notons d'ailleurs que $\varepsilon \in \langle u \rangle L$ si et seulement si $\langle u \rangle L \neq \emptyset$.

Question 11. Pour chacune des égalités suivantes, dites lesquelles sont valides pour tous mots u et v , lettre a , et langages L, L_1, L_2 . Justifiez vos réponses négatives par un contre-exemple.

- | | |
|--|--|
| (1) $\langle \varepsilon \rangle L = L$, | (2) $\langle a \rangle (L_1 \cdot L_2) = (\langle a \rangle L_1) \cdot L_2 \cup \langle a \rangle L_2$, |
| (3) $\langle uv \rangle L = \langle u \rangle (\langle v \rangle L)$, | (4) $\langle u \rangle (L^*) = (\langle u \rangle L) \cdot L^*$. |

Question 12. a. Programmez une fonction OCaml `eps_residu_ratexp : ratexp -> ratexp` qui à partir d'une expression rationnelle e construit une expression rationnelle e' telle que $L(e') = \langle \varepsilon \rangle L(e)$.

b. Donnez (et justifiez) un majorant, en fonction de $|e|$, de la taille $|e'|$ de l'expression construite par votre programme.

Question 13. a. Programmez une fonction OCaml `char_residu_ratexp : char -> ratexp -> ratexp` qui, à partir de $a \in A$ et e , construit une expression e'' telle que $L(e'') = \langle a \rangle L(e)$.
 b. Donnez (et justifiez) un majorant, en fonction de $|e|$, de la taille $|e''|$ de l'expression construite par votre programme.

Question 14. a. Programmez une fonction OCaml `sousmot_de_ratexp : string -> ratexp -> bool` décidant si $u \in \downarrow L(e)$ pour un mot u et une expression rationnelle e .

Indication : on pourra utiliser la fonction `char_residu_ratexp`.

b. Votre programme s'exécute-t-il en temps polynomial en $|u| + |e|$? Justifiez brièvement votre réponse.

On développe maintenant une autre approche pour décider si $u \in \downarrow L(e)$. Pour un mot $u = a_0a_1 \cdots a_{n-1}$ et un langage $L \subseteq A^*$, on définit $FC(u, L)$ comme étant l'ensemble des couples (i, j) tels que $0 \leq i \leq j \leq |u|$ et $u[i, j[\in \downarrow L$. Quand $(i, j) \in FC(u, L)$ on dit que « L couvre le facteur $[i, j[$ de u ». On écrit aussi $FC(u, e)$ au lieu de $FC(u, L(e))$ quand e est une expression rationnelle.

Pour représenter un ensemble de couples tel que $FC(u, e)$, on utilisera une matrice booléenne M de dimension $(n+1) \times (n+1)$ telle que $M[i, j] = \text{true}$ ssi $(i, j) \in FC(u, e)$. Notons qu'en particulier $M[i, j] = \text{false}$ pour $j < i$.

Question 15. Programmez une fonction `facteurs_couverts : string -> ratexp -> bool array array` qui, pour un mot u et une expression e , calcule $FC(u, e)$. Indiquez et justifiez brièvement la complexité de votre fonction.

Pour ce code, il est suggéré de construire la matrice associée à une expression complexe e à partir des matrices associées aux sous-expressions de e .

Question 16. En utilisant la fonction `facteurs_couverts`, programmez une nouvelle version de la fonction `sousmot_de_ratexp` décidant si $u \in \downarrow L(e)$ pour un mot u et une expression rationnelle e (cf. question 14). Indiquez la complexité de la nouvelle version.

* *
*

A2019 – INFO MP

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

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

CONCOURS 2019**ÉPREUVE D'INFORMATIQUE MP**

Durée de l'épreuve : 3 heures

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

Cette épreuve concerne uniquement les candidats de la filière MP.

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

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 10 pages de texte.

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

L'épreuve est composée d'un unique problème, comportant 37 questions. Après un préliminaire, ce problème est divisé en 5 parties. Pour répondre à une question, un candidat pourra réutiliser le résultat d'une question antérieure, même s'il n'est pas parvenu à démontrer ce résultat.

Le but du problème est d'étudier les relations qui existent entre des automates qui reconnaissent un même langage grâce à la notion de morphismes d'automates.

Préliminaires

Concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation Caml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Enfin, si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile de tester si les hypothèses sont bien vérifiées dans le code de la fonction.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple n).

Définition mathématique d'un automate

Définition : Dans l'ensemble du sujet, le terme *automate* désigne un automate fini déterministe complet sur l'alphabet $\{a, b\}$, c'est-à-dire un quadruplet $\mathcal{A} = \langle Q, i, \delta, F \rangle$ où Q est l'ensemble des états, i l'état initial ($i \in Q$), $\delta : Q \times \{a, b\} \rightarrow Q$ l'application de transition et $F \subseteq Q$ l'ensemble des états finals.

On note ε le mot vide. Par extension de δ , on appelle δ^* l'application $Q \times \{a, b\}^* \rightarrow Q$ définie pour tout état q par $\delta^*(q, \varepsilon) = q$ et, si σ est une lettre de $\{a, b\}$ et w un mot de $\{a, b\}^*$, $\delta^*(q, \sigma w) = \delta^*(\delta(q, \sigma), w)$.

Un automate $\langle Q, i, \delta, F \rangle$ est représenté par un graphe orienté. Les sommets de ce graphe sont les éléments de Q . Ce graphe admet un arc $(p, q) \in Q \times Q$ étiqueté par la lettre a si et seulement si $\delta(p, a) = q$; de même, ce graphe admet un arc $(p, q) \in Q \times Q$ étiqueté par la lettre b si et seulement si $\delta(p, b) = q$. Une flèche venant de nulle part et pointant vers i indique l'état initial. Un état final est représenté par un double cercle.

Représentation d'automate en Caml

Indication Caml : Dans toutes les questions demandant d'implémenter une fonction en Caml, on identifie l'ensemble des états Q d'un automate $\langle Q, i, \delta, F \rangle$ avec l'ensemble des

entiers compris entre 0 et $|Q| - 1$. On convient de plus que l'état initial i est toujours identifié à l'entier 0. Les automates seront représentés par des triplets (n, δ, f) où

- n , de type `int`, est le nombre d'états de l'automate ; les états de l'automate sont les entiers de 0 à $n-1$,
- δ , de type `(int * int) array` de longueur n , est un tableau qui stocke les couples $(\delta(q, a), \delta(q, b))_{q \in Q}$,
- f , de type `bool array` de longueur n , est un tableau qui représente la fonction indicatrice de l'ensemble des états finals.

Dans l'ensemble du sujet, le type `automate` est défini par l'alias suivant.

```
type automate = int * (int * int) array * bool array;;
```

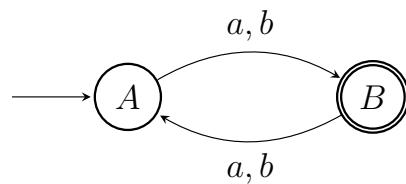
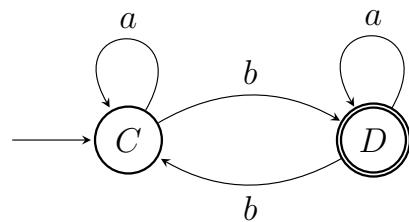
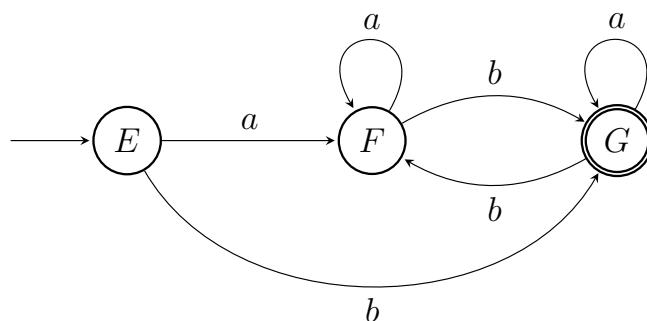
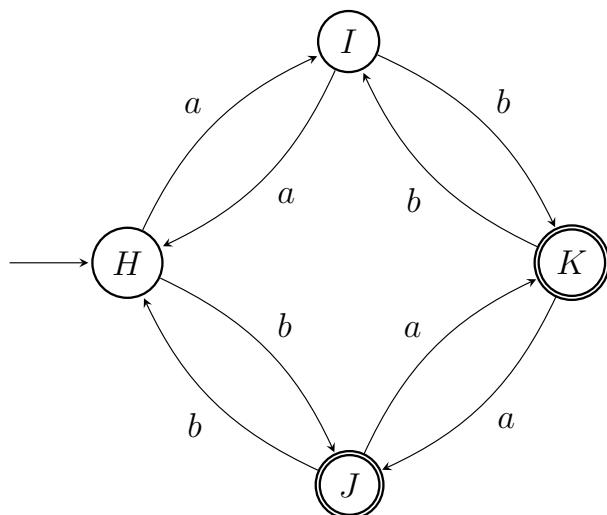
Ci-dessous sont donnés quelques exemples de son utilisation.

- **let** $(n, \delta, f) = \text{aut} \text{ in } \dots$
permet de récupérer les composantes d'une variable `aut` de type `automate`.
- **let** $(\text{succ_a}, \text{succ_b}) = \delta.(q) \text{ in } \dots$
permet ensuite de récupérer le successeur par la lettre a et le successeur par la lettre b de l'état q (qui est de type `int`).
- **if** $f.(q) \text{ then } \dots$
permet de tester si l'état q (qui est de type `int`) est final.

Indication Caml : On rappelle que la fonction `List.length`, de type `'a list -> int`, renvoie la longueur d'une liste. On rappelle que `Array.make n x` permet de créer un tableau de longueur n et initialisé avec la valeur x , que `Array.copy t` renvoie une copie d'un tableau t , que `Array.length t` renvoie la longueur d'un tableau t . On rappelle enfin que `Array.make_matrix n m x` permet de créer un tableau de tableaux de taille $n \times m$ dont toutes les cases sont initialisées avec la valeur x .

1 Premiers exemples

- 1 – Donner, sans preuve, une description courte en langue française du langage reconnu par l'automate \mathcal{A}_1 de la figure 1.
- 2 – Donner, sans preuve, une description courte en langue française du langage reconnu par l'automate \mathcal{A}_2 de la figure 2.
- 3 – Donner, sans preuve, une expression rationnelle qui dénote le langage reconnu par l'automate \mathcal{A}_1 de la figure 1.
- 4 – Donner, sans preuve, une expression rationnelle qui dénote le langage reconnu par l'automate \mathcal{A}_2 de la figure 2.
- 5 – Écrire en Caml, sans justification, la construction d'une instance du type `automate` qui corresponde à l'automate \mathcal{A}_2 de la figure 2.

FIGURE 1 – Automate \mathcal{A}_1 FIGURE 2 – Automate \mathcal{A}_2 FIGURE 3 – Automate \mathcal{A}_3 FIGURE 4 – Automate \mathcal{A}_4

2 États accessibles d'un automate

- 6 – Écrire une fonction `numero` de type `int -> int list -> int array` qui, à partir d'un entier n et une liste A d'entiers compris entre 0 et $n - 1$, renvoie un tableau T de taille n tel que, pour tout i compris entre 0 et $n - 1$, la $i^{\text{ième}}$ case $T[i]$ de T vaut -1 si i est absent de A et $T[i]$ représente l'indice de l'une des occurrences de i dans A sinon. Par exemple, `numero 5 [3;2;0] ;;` peut renvoyer `[|2; -1; 1; 0; -1|]`.

Définition : Un état q d'un automate $\mathcal{A} = \langle Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ est dit *accessible* s'il existe un mot $w \in \{a, b\}^*$ tel que $\delta^*(i_{\mathcal{A}}, w) = q$, autrement dit s'il existe un chemin qui relie l'état initial $i_{\mathcal{A}}$ à l'état q dans sa représentation graphique. (On notera que l'état initial $i_{\mathcal{A}}$ est toujours accessible.).

Soit Q' l'ensemble des états accessibles de l'automate \mathcal{A} . On appelle *partie accessible* de l'automate \mathcal{A} le nouvel automate $\mathcal{A}' = \langle Q', i_{\mathcal{A}}, \delta', F_{\mathcal{A}} \cap Q' \rangle$ où δ' est la restriction de l'application $\delta_{\mathcal{A}}$ au domaine $Q' \times \{a, b\}$.

On dit qu'un automate est *accessible* lorsque tous ses états sont accessibles.

- 7 – Écrire une fonction `etats_accessible`, de type `automate -> int list`, qui renvoie la liste des états accessibles de l'automate donné en argument et que l'on obtient par un parcours de graphe en profondeur depuis l'état initial. La liste renvoyée doit suivre l'ordre dans lequel les états sont rencontrés pour la première fois et ne doit pas contenir de doublons. Donner la complexité de la fonction écrite.
- 8 – Écrire une fonction `partie_accessible` de type `automate -> automate` qui construit la partie accessible de l'automate donné en argument. On pourra réemployer les fonctions implémentées aux questions 6 et 7.

3 Morphismes d'automates

Définition : Soient deux automates $\mathcal{A} = \langle Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ et $\mathcal{B} = \langle Q_{\mathcal{B}}, i_{\mathcal{B}}, \delta_{\mathcal{B}}, F_{\mathcal{B}} \rangle$. Une application $\varphi : Q_{\mathcal{A}} \rightarrow Q_{\mathcal{B}}$ est appelée *morphisme d'automates* de l'automate \mathcal{A} vers l'automate \mathcal{B} , et est notée $\varphi : \mathcal{A} \rightarrow \mathcal{B}$, si elle satisfait les conditions suivantes :

$$\varphi \text{ est surjective,} \tag{1}$$

$$\varphi(i_{\mathcal{A}}) = i_{\mathcal{B}}, \tag{2}$$

$$\forall q \in Q_{\mathcal{A}}, \quad \forall \sigma \in \{a, b\}, \quad \varphi(\delta_{\mathcal{A}}(q, \sigma)) = \delta_{\mathcal{B}}(\varphi(q), \sigma), \tag{3}$$

$$\forall q \in Q_{\mathcal{A}}, \quad q \in F_{\mathcal{A}} \iff \varphi(q) \in F_{\mathcal{B}} \tag{4}$$

Indication Caml : En Caml, on représente un morphisme $\varphi : \mathcal{A} \rightarrow \mathcal{B}$ par le tableau $[\varphi(q)]_{q \in Q_{\mathcal{A}}}$ de longueur $|Q_{\mathcal{A}}$, de type int array, formé d'entiers compris entre 0 et $|Q_{\mathcal{B}}| - 1$. On pourra utiliser le type morphisme, défini par l'alias

```
type morphisme = int array;;
```

3.1 Exemples de morphismes d'automates

- 9 – À partir des figures 2 et 3 représentant les automates \mathcal{A}_2 et \mathcal{A}_3 , recopier le tableau suivant et le compléter sans justification par des états de \mathcal{A}_2 de sorte que ce tableau représente un morphisme d'automates φ de l'automate \mathcal{A}_3 vers l'automate \mathcal{A}_2 .

q	$\varphi(q)$
E	
F	
G	

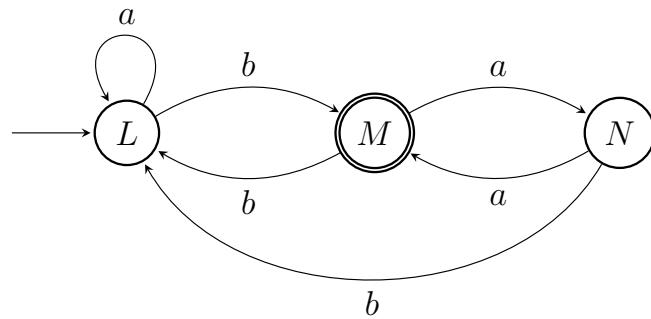
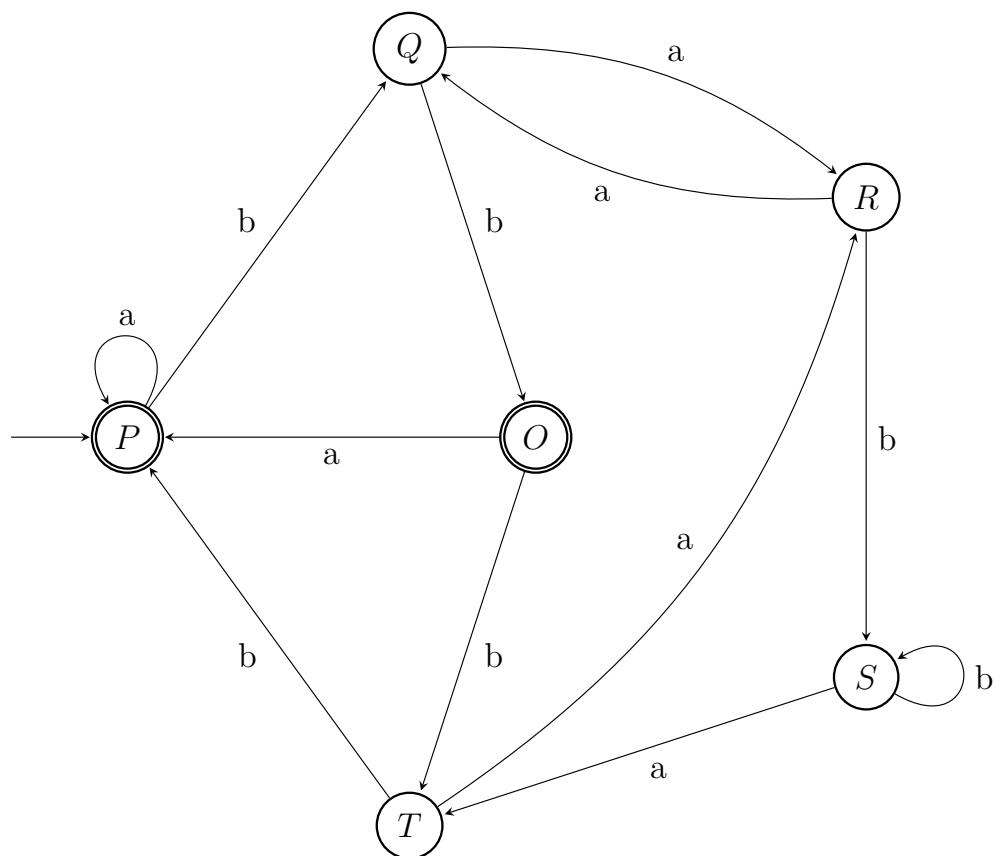
- 10 – À partir des figures 2 et 4 représentant les automates \mathcal{A}_2 et \mathcal{A}_4 , donner, sans en justifier l'expression, un morphisme d'automates de l'automate \mathcal{A}_4 vers l'automate \mathcal{A}_2 .
- 11 – À partir des figures 1 et 2, montrer qu'il n'existe pas de morphisme d'automates de l'automate \mathcal{A}_1 vers l'automate \mathcal{A}_2 .
- 12 – À partir des figures 2 et 5, montrer qu'il n'existe pas de morphisme d'automates de l'automate \mathcal{A}_5 vers l'automate \mathcal{A}_2 .

3.2 Propriétés des morphismes d'automates

- 13 – Montrer que deux automates acceptent le même langage dès lors qu'il existe un morphisme d'automates de l'un des automates vers l'autre.
- 14 – Montrer qu'un morphisme φ entre deux automates ayant le même nombre d'états est nécessairement une application bijective et que l'application φ^{-1} est encore un morphisme d'automates.

On dit dans ce cas que φ est un *isomorphisme* d'automates.

- 15 – Montrer que la composition de deux morphismes d'automates est encore un morphisme d'automates.

FIGURE 5 – Automate \mathcal{A}_5 FIGURE 6 – Automate \mathcal{A}_6

3.3 Existence de morphismes d'automates entre automates accessibles

- 16 – Montrer que le point (1) de la définition des morphismes d'automates découle des points (2), (3) et (4) quand les deux automates considérés sont accessibles.
- 17 – Écrire en Caml une fonction `existe_morphisme` de type `automate -> automate -> bool * morphisme` qui, sous l'hypothèse que les deux automates en argument sont accessibles, renvoie d'une part un booléen indiquant l'existence d'un morphisme d'automates du premier argument vers le second et d'autre part un tel morphisme lorsqu'il existe. Lorsqu'un tel morphisme n'existe pas, la seconde composante de la valeur de retour est un tableau quelconque. On pourra expliquer le principe de l'algorithme avant d'en donner le code.

4 Constructions de morphismes d'automates

4.1 Automate produit

Définition : Soient $\mathcal{A} = \langle Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ et $\mathcal{A}' = \langle Q_{\mathcal{A}'}, i_{\mathcal{A}'}, \delta_{\mathcal{A}'}, F_{\mathcal{A}'} \rangle$ deux automates. On appelle *automate produit*, et on note $\mathcal{A} \times \mathcal{A}'$, le nouvel automate

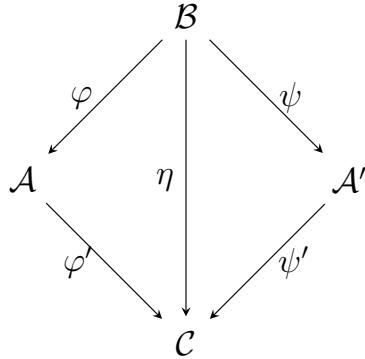
$$\mathcal{A} \times \mathcal{A}' = \langle Q_{\mathcal{A}} \times Q_{\mathcal{A}'}, (i_{\mathcal{A}}, i_{\mathcal{A}'}), \delta_{\mathcal{A} \times \mathcal{A}'}, F_{\mathcal{A}} \times F_{\mathcal{A}'} \rangle$$

où l'application $\delta_{\mathcal{A} \times \mathcal{A}'}$ est définie, pour tout couple d'états $(q, q') \in Q_{\mathcal{A}} \times Q_{\mathcal{A}'}$ et pour toute lettre $\sigma \in \{a, b\}$, par $\delta_{\mathcal{A} \times \mathcal{A}'}((q, q'), \sigma) = (\delta_{\mathcal{A}}(q, \sigma), \delta_{\mathcal{A}'}(q', \sigma))$.

- 18 – On considère les automates \mathcal{A}_3 et \mathcal{A}_4 qui sont représentés par les figures 3 et 4. Dessiner, sans justification, la partie accessible du produit d'automates $\mathcal{A}_3 \times \mathcal{A}_4$.
- 19 – Écrire une fonction `produit` de type `automate -> automate -> automate` qui renvoie le produit des deux automates donnés en argument.
- 20 – Soit (q, q') un état accessible du produit de deux automates qui acceptent le même langage. Montrer que q est un état final du premier automate si et seulement si q' est un état final du second automate.
- 21 – Montrer qu'il existe toujours un morphisme d'automates de la partie accessible du produit de deux automates accessibles qui acceptent le même langage vers chacun de ces deux automates.

4.2 Diagramme d'automates

Dans toute la sous-section 4.2, on considère qu'il existe trois automates accessibles \mathcal{A} , \mathcal{A}' et $\mathcal{B} = \langle Q_{\mathcal{B}}, i_{\mathcal{B}}, \delta_{\mathcal{B}}, F_{\mathcal{B}} \rangle$ et deux morphismes d'automates $\varphi : \mathcal{B} \rightarrow \mathcal{A}$ et $\psi : \mathcal{B} \rightarrow \mathcal{A}'$. Le but de cette sous-section est de construire un nouvel automate accessible \mathcal{C} et trois morphismes φ' , ψ' et η dont la situation est résumée dans le diagramme suivant.



Définition : On définit une relation sur $Q_{\mathcal{B}}$, notée \equiv . Pour tout couple d'états (p, q) appartenant à $Q_{\mathcal{B}}^2$, $p \equiv q$ s'il existe une suite finie de longueur $k + 1$ (avec $k \in \mathbb{N}$) constituée des termes $p = q_0, q_1, q_2, \dots, q_k = q$ d'états de $Q_{\mathcal{B}}$ telle que

$$\forall 0 \leq j < k, \quad \varphi(q_j) = \varphi(q_{j+1}) \text{ ou } \psi(q_j) = \psi(q_{j+1}).$$

- 22 – Montrer que la relation \equiv définie sur l'ensemble $Q_{\mathcal{B}}$ est une relation d'équivalence.
- 23 – Montrer que, pour tout couple d'états $(p, q) \in Q_{\mathcal{B}}^2$, si $p \equiv q$, alors, pour toute lettre $\sigma \in \{a, b\}$, on a $\delta_{\mathcal{B}}(p, \sigma) \equiv \delta_{\mathcal{B}}(q, \sigma)$.
- 24 – Montrer que, pour tout couple d'états $(p, q) \in Q_{\mathcal{B}}^2$, si $p \equiv q$, alors p est un état final de l'automate \mathcal{B} si et seulement si q est un état final de l'automate \mathcal{B} .

Définition : La classe d'équivalence, notée $[q]$, d'un état $q \in Q_{\mathcal{B}}$ est l'ensemble

$$[q] = \{p \in Q_{\mathcal{B}}; q \equiv p\}.$$

Dans ce qui suit, on appelle ℓ le nombre de classes d'équivalence de la relation \equiv et on note $S_0, S_1, \dots, S_{\ell-1}$ ces classes. On choisira S_0 de sorte que $i_{\mathcal{B}} \in S_0$. On note η l'application $Q_{\mathcal{B}} \rightarrow \{S_0, S_1, \dots, S_{\ell-1}\}$ qui, à chaque état $q \in Q_{\mathcal{B}}$, associe la classe d'équivalence $[q]$.

Indication Caml : En Caml, on représentera la classe d'équivalence S_j par l'indice j .

- 25 – Construire un automate accessible \mathcal{C} dont l'ensemble d'états est $\{S_0, S_1, \dots, S_{\ell-1}\}$ et tel que η est un morphisme d'automates de l'automate \mathcal{B} vers l'automate \mathcal{C} . Justifier.
- 26 – Construire deux morphismes d'automates $\varphi' : \mathcal{A} \rightarrow \mathcal{C}$ et $\psi' : \mathcal{A}' \rightarrow \mathcal{C}$, où \mathcal{C} est l'automate construit à la question 25.
- 27 – Écrire une fonction `renomme`, de type `int array -> int array`, qui renomme le contenu d'un tableau contenant des entiers positifs prenant ℓ valeurs distinctes en utilisant les entiers entre 0 et $\ell - 1$. Le premier élément du résultat doit de plus être égal à 0. Par exemple `renomme [|4;4;5;0;4;5|] ;;` peut renvoyer `[|0;0;1;2;0;1|]`. Préciser la complexité de la fonction proposée.
- 28 – Écrire une fonction `relation`, de type `morphisme -> morphisme -> morphisme`, qui, à partir des tableaux $[\varphi(q)]_{q \in Q_B}$ et $[\psi(q)]_{q \in Q_B}$, renvoie le tableau $[\eta(q)]_{q \in Q_B}$, autrement dit, qui renvoie un tableau `t` d'entiers compris entre 0 et $\ell - 1$ tel que pour tout couple d'états $(p, q) \in Q_B^2$, les valeurs `t.(q)` et `t.(p)` sont égales si et seulement si $p \equiv q$ et tel que `t.(0)` vaut 0.

5 Réduction d'automates

5.1 Existence et unicité

- 29 – Montrer que si deux automates accessibles \mathcal{A} et \mathcal{A}' acceptent le même langage, alors on peut construire un automate \mathcal{C} et deux morphismes $\varphi' : \mathcal{A} \rightarrow \mathcal{C}$ et $\psi' : \mathcal{A}' \rightarrow \mathcal{C}$.
- 30 – Déterminer l'automate \mathcal{C} défini à la question 29 pour les automates \mathcal{A}_3 et \mathcal{A}_4 (figures 3 et 4) et préciser les applications φ' et ψ' .

Soit L un langage rationnel et \mathfrak{K}_L l'ensemble des automates (complets déterministes) accessibles qui acceptent le langage L . On note m_L le plus petit nombre d'états d'un automate de \mathfrak{K}_L .

- 31 – Montrer que deux automates de \mathfrak{K}_L ayant m_L états sont nécessairement isomorphes.
- 32 – Montrer que, pour tout automate \mathcal{A} dans \mathfrak{K}_L , il existe un morphisme $\varphi : \mathcal{A} \rightarrow \mathcal{M}_L$, où \mathcal{M}_L est un automate de \mathfrak{K}_L à m_L états.

5.2 Construction d'un automate réduit par fusion d'états

Définition : Soient $\mathcal{A} = \langle Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ et $\mathcal{A}' = \langle Q_{\mathcal{A}'}, i_{\mathcal{A}'}, \delta_{\mathcal{A}'}, F_{\mathcal{A}'} \rangle$ deux automates. On dit que deux états p et q de l'automate \mathcal{A} ont été fusionnés dans l'automate \mathcal{A}' s'il existe un morphisme d'automates de \mathcal{A} vers \mathcal{A}' tel que $\varphi(p) = \varphi(q)$ et si le nombre d'état satisfait $|Q_{\mathcal{A}'}| < |Q_{\mathcal{A}}|$.

- 33 – On considère l'automate \mathcal{A}_6 de la figure 6. Dessiner un automate $\mathcal{A}_6^{O,P}$ dans lequel les états O et P ont été fusionnés. On donnera un morphisme d'automates $\mathcal{A}_6 \rightarrow \mathcal{A}_6^{O,P}$.
- 34 – Expliquer brièvement pourquoi il n'est pas possible de construire un automate $\mathcal{A}_6^{Q,R}$ muni d'un morphisme d'automates $\psi : \mathcal{A}_6 \rightarrow \mathcal{A}_6^{Q,R}$ tel que $\psi(Q) = \psi(R)$.
- 35 – Quels états faut-il encore fusionner dans $\mathcal{A}_6^{O,P}$ pour obtenir un automate à trois états \mathcal{M}_{L_6} , qui reconnaît le même langage que \mathcal{A}_6 ?

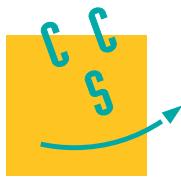
- 36 – Soit $\mathcal{A} = \langle Q, i, \delta, F \rangle$ un automate accessible. On appelle P le graphe orienté de sommets $Q \times Q$ et, pour toute lettre $\sigma \in \{a, b\}$, d'arcs allant du sommet $(p, q) \in Q \times Q$ vers le sommet $(\delta(p, \sigma), \delta(q, \sigma)) \in Q \times Q$.

Écrire en Caml une fonction `table_de_predecesseurs` de type `automate -> bool array array` qui prend en entrée un automate accessible $\mathcal{A} = \langle Q, i, \delta, F \rangle$ à n états et renvoie en sortie une matrice de taille $n \times n$. Pour tous les états p et q de l'automate \mathcal{A} , la valeur de la case (p, q) de la matrice vaut true si et seulement s'il existe deux états $(p_0, q_0) \in Q^2$ et un chemin du sommet (p_0, q_0) vers le sommet (p, q) dans le graphe P tel que $p_0 \in F$ et $q_0 \notin F$ ou $p_0 \notin F$ et $q_0 \in F$.

On essaiera de ne pas dépasser une complexité en $O(n^2)$.

- 37 – En décrire le principe, le justifier, puis écrire en Caml une fonction `reduit`, de type `automate -> automate` qui prend en entrée un automate \mathcal{A} et renvoie l'automate \mathcal{M}_L associé au langage L reconnu par \mathcal{A} .

FIN DE L'ÉPREUVE



CONCOURS CENTRALE-SUPÉLEC

Option informatique

MP

2019

4 heures

Calculatrice autorisée

Les candidat devront utiliser le langage Caml, à l'exclusion de tout autre, pour traiter ce sujet. Ils devront donner le type de chaque fonction écrite.

Dans tout le problème, n désigne un entier naturel non nul.

Si $x \in \{0; 1\}$, \bar{x} désigne $1 - x$.

On note \oplus l'opérateur *ou exclusif* (également appelé *XOR*). Il est défini sur $\{0; 1\}$ par la table suivante :

	0	1
0	0	1
1	1	0

On prolonge cette définition à \mathbb{N} de la manière suivante.

Soit $(x, y) \in \mathbb{N}^2$ vérifiant $x < 2^p$ et $y < 2^p$ où p est un entier naturel non nul. On décompose x et y en base 2 :

$$x = \sum_{k=0}^p a_k 2^k \quad \text{et} \quad y = \sum_{k=0}^p b_k 2^k,$$

où les coefficients a_k et b_k sont des éléments de $\{0; 1\}$. On définit alors $x \oplus y$ par :

$$x \oplus y = \sum_{k=0}^p (a_k \oplus b_k) 2^k.$$

I Code binaire de Gray

On cherche à obtenir tous les n -uplets composés de 0 et 1 de deux manières différentes.

Ces n -uplets seront implémentés à l'aide de listes.

I.A – Dans cette sous-partie, on obtient ces n -uplets dans l'ordre lexicographique.

Q 1. Écrire une fonction **suivant** transformant un n -uplet en son suivant dans l'ordre lexicographique. Par exemple, si $t = (0; 1; 0; 0; 1; 1; 1)$, après exécution de **suivant(t)**, on a $t = (0; 1; 0; 1; 0; 0; 0)$.

Cette fonction modifie la liste qu'elle reçoit en argument. De plus elle renvoie un booléen, valant **vrai** si elle a pu déterminer un n -uplet suivant, ou bien **faux** si le n -uplet fourni en argument était le dernier. Dans ce dernier cas, la valeur de ce n -uplet après exécution de la fonction est non spécifiée.

Q 2. Écrire une fonction affichant tous les n -uplets dans l'ordre lexicographique.

On pourra commencer par écrire une fonction affichant les éléments d'un n -uplet :

```
affiche_nuplet : int list -> unit
```

I.B – Pour certaines applications, par exemple pour éviter des états transitoires intermédiaires dans les circuits logiques ou pour faciliter la correction d'erreur dans les transmissions numériques, on souhaite que le passage d'un n -uplet au suivant ne modifie qu'un seul bit. Dans un brevet de 1953, Frank Gray¹ définit un ordre des n -uplets possédant cette propriété.

Pour produire la liste des n -uplets dans l'ordre de Gray, un algorithme consiste à partir de la liste des 1-uplets (0, 1) et à construire la liste des $(n + 1)$ -uplets à partir de celles de n -uplets en ajoutant un 0 en tête de chaque n -uplet puis un 1 en tête de chaque n -uplet en parcourant leur liste à l'envers. On obtient ainsi pour $n = 2$, la liste (00, 01, 11, 10), pour $n = 3$, la liste (000, 001, 011, 010, 110, 111, 101, 100), etc.

Les n -uplets sont représentés par des listes, une liste de n -uplets sera donc une liste de listes.

Q 3. Écrire une fonction **ajout** telle que si a est un entier et l une liste de n -uplets d'entiers, **ajout a l** renvoie une liste contenant les éléments de l auxquels on a ajouté a en tête.

¹ Frank Gray (1887-1969) était un chercheur travaillant chez Bell Labs ; il a en particulier produit de nombreuses innovations dans le domaine de la télévision.

Q 4. Écrire deux fonctions mutuellement récursives `monte` et `descend` prenant en argument un entier n et renvoyant les n -uplets dans l'ordre de Gray. L'une les renverra de 00...0 à 10...0, l'autre en sens inverse.

Q 5. Évaluer la complexité de ces fonctions en termes d'appels à `monte` et `descend`.

Q 6. Décrire une façon simple d'améliorer cette complexité.

I.C – Dans tout ce qui suit, l'expression *représentation binaire*, désigne la représentation traditionnelle en base 2. Étant donné $k \in \mathbb{N}^*$, on a de manière unique

$$k = \sum_{i=0}^p a_i 2^i \quad \text{avec} \quad p \in \mathbb{N}, \quad a_i \in \{0; 1\}, \quad a_p \neq 0$$

La représentation binaire canonique de k est $a_p \dots a_0$ (le bit de poids fort, qui est non nul, en premier). On dira que tout n -uplet constitué d'un nombre quelconque de 0 suivis de la représentation binaire canonique de k est une représentation binaire de k .

On définit une fonction g sur \mathbb{N} de la manière suivante. Pour $k \in \mathbb{N}$ et n tel que $k < 2^n$, on considère la liste dans l'ordre de Gray des 2^n n -uplets ; on indice cette liste de 0 à $2^n - 1$; $g(k)$ est le nombre dont une représentation binaire est le n -uplet d'indice k .

Par exemple, si on énumère les 3-uplets selon l'ordre de Gray, on a (000, 001, 011, 010, 110, 111, 101, 100). Dans cette liste, l'élément d'indice 0 est 000 donc $g(0) = 0$, le 3-uplet d'indice 7 est 100 donc $g(7) = 4$.

Soit n un entier naturel et k un entier compris entre 2^n et $2^{n+1} - 1$: $k = 2^n + r$ avec $0 \leq r < 2^n$.

Q 7. Démontrer que $g(k) = 2^n + g(2^n - 1 - r)$.

Q 8. En déduire que, si la représentation binaire de k est $b_n \dots b_0$ et si on pose $b_{n+1} = 0$, la représentation binaire de $g(k)$ est $a_n \dots a_0$ où, pour tout j entre 0 et n , $a_j = b_j \oplus b_{j+1}$.

Q 9. Exprimer, pour $k \in \mathbb{N}$, $g(k)$ en fonction de k (à l'aide de \oplus).

Q 10. Montrer que g est une bijection de \mathbb{N} dans \mathbb{N} et, avec les notations précédentes, exprimer b_j en fonction de a_k , $k \geq j$.

I.D – On cherche à réaliser les fonctions g et g^{-1} avec des circuits logiques. Une porte logique sera représentée par un rectangle contenant son nom (AND, OR, XOR, NOT) et on indiquera les entrées et sorties par des flèches. Par exemple :



On se place d'abord dans le cas $n = 3$.

Q 11. Donner un circuit logique à 3 entrées représentant les trois bits d'un entier k inférieur ou égal à 7 et à trois sorties représentant les trois bits de $g(k)$.

On pourra utiliser la porte logique XOR.

Q 12. Donner un circuit pour l'opération inverse.

Q 13. Si $n \geq 2$, donner un circuit permettant de passer d'un nombre k à n bits à $g(k)$. Faire de même pour l'opération inverse en utilisant le moins possible de portes. Préciser le nombre de portes utilisées.

II Enumération des combinaisons

On souhaite maintenant parcourir toutes les combinaisons de p éléments pris dans un ensemble de cardinal n avec $0 \leq p \leq n$. On supposera que cet ensemble est celui des n premiers entiers naturels. On le note E_n ; $E_n = \{0, \dots, n-1\}$. Les éléments d'une combinaison de E_n seront systématiquement considérés dans l'ordre croissant.

II.A – On se place dans le cas $p = 3$.

Q 14. Écrire une fonction d'argument n affichant les combinaisons de 3 éléments pris dans $\{0, \dots, n-1\}$; on souhaite que ces combinaisons apparaissent dans l'ordre lexicographique. Par exemple pour $n = 4$: [0, 1, 2], [0, 1, 3], [0, 2, 3], [1, 2, 3].

II.B – On revient au cas p entier quelconque entre 1 et n .

Q 15. Donner la première et la dernière combinaison de p éléments de E_n ($p \leq n$) lorsqu'on énumère ces combinaisons dans l'ordre lexicographique.

On suppose que $c_0 \dots c_{p-1}$ est une combinaison de E_n vérifiant $c_0 < \dots < c_{p-1}$, stockée sous la forme d'un vecteur ou d'un tableau ; on suppose également que cette combinaison n'est pas la dernière.

Q 16. Écrire une fonction `comb_suivante` permettant de transformer une combinaison en la combinaison suivante dans l'ordre lexicographique.

On pourra commencer par chercher le plus petit indice j tel que $c_{j+1} > c_j + 1$.

Q 17. En déduire une fonction d'arguments n et p , énumérant et affichant toutes les combinaisons de p éléments de E_n dans l'ordre lexicographique.

Q 18. Déterminer le nombre de combinaisons affichées avant d'arriver à la combinaison $c_0 \cdots c_{p-1}$.

Q 19. En déduire que tout nombre entier naturel non nul N peut s'écrire sous la forme

$$N = \sum_{k=1}^p \binom{n_k}{k}$$

où $0 \leq n_1 < n_2 < \cdots < n_p$. On dira qu'il s'agit d'une décomposition combinatoire de degré p pour l'entier N .

Q 20. Démontrer que si m et p sont deux entiers naturels tels que $p \leq m$, alors

$$\sum_{k=0}^{p-1} \binom{m-k}{p-k} = \binom{m+1}{p} - 1$$

On pourra commencer par vérifier que, pour k entre 0 et $p-1$, $\binom{m-k}{p-k} = \binom{m-k+1}{m-p+1} - \binom{m-k}{m-p+1}$.

Q 21. En déduire l'unicité, pour $N \in \mathbb{N}$, d'une décomposition combinatoire de degré p pour N .

II.C – On suppose que l'on dispose de n ($n \geq 3$) objets o_0, \dots, o_{n-1} et d'un sac dont la charge maximale est M . Chaque objet o_i a une valeur v_i et une masse m_i . Soit p un entier entre 1 et $n-2$. On veut choisir p objets parmi les n de façon à ce que la masse totale de ces p objets soit inférieure à M et leur valeur totale soit la plus grande possible.

Q 22. De quelle manière peut-on utiliser la fonction ci-dessus pour résoudre ce problème ?

Q 23. Évaluer le nombre d'additions effectuées (on ne s'intéressera qu'aux additions entre m_i et entre v_i).

II.D – Pour améliorer cet algorithme, on souhaite passer d'une combinaison à une autre en ne faisant que deux modifications.

Q 24. Expliquer comment représenter une combinaison de p éléments pris parmi n par un n -uplet de 0 et 1.

Q 25. Modifier les fonctions `monte` et `descend` de la question 4 pour qu'elles renvoient les n -uplets représentant les combinaisons de p éléments pris parmi n , dans le même ordre que précédemment.

Q 26. Déterminer le premier et le dernier n -uplet renvoyé par l'appel `monte n p`.

Q 27. Montrer qu'entre deux éléments successifs de la liste de n -uplets renvoyés par `monte` ou `descend`, seuls deux bits changent. Montrer que cette propriété est également vraie entre le dernier et le premier n -uplets.

Soit a l'un de nos n -uplets, et soit a' son successeur. On admet qu'il existe un unique entier j , $1 \leq j \leq \max(p, n-p)$ tel que $g^{-1}(a) - g^{-1}(a') \equiv 2^j \pmod{2^n}$

Q 28. En déduire un algorithme simple pour passer d'un n -uplet contenant p bits à 1 au suivant. Le décrire et le programmer sous la forme d'une fonction `suivant`.

II.E –

Q 29. En déduire un algorithme qui donne un remplissage optimal du sac. Cet algorithme aura comme entrée :

- les valeurs des n objets,
- les masses des n objets, dans le même ordre,
- l'entier p ,
- la masse maximum M .

Il rendra en résultat les p indices des objets choisis.

On ne demande pas la programmation explicite de cet algorithme.

• • • FIN • • •

SESSION 2019

MPIN007

**ÉPREUVE SPÉCIFIQUE - FILIÈRE MP****INFORMATIQUE****Jeudi 2 mai : 14 h - 18 h**

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet est composé de trois parties indépendantes.

Partie I - Inversions de permutations (Informatique pour tous)

Pour tout $n \in \mathbb{N} \setminus \{0\}$, une **permutation de taille n** est une bijection de l'ensemble $\{0, 1, \dots, n-1\}$ dans lui-même. Dans la suite, l'ensemble des permutations de taille n est noté \mathfrak{S}_n . Étant donné une permutation σ de taille n , on la représente sous la forme $\sigma_0 \sigma_1 \dots \sigma_{n-1}$ où :

$$\forall i \in \{0, \dots, n-1\}, \sigma_i = \sigma(i).$$

Par exemple, $\sigma = 1032$ représente la permutation envoyant 0 sur 1, 1 sur 0, 2 sur 3 et 3 sur 2.

Soit $\sigma = \sigma_0 \dots \sigma_{n-1}$ une permutation de taille n . On dit que $(i, j) \in \{0, 1, \dots, n-1\}^2$ est une **inversion** de σ si $\sigma_i > \sigma_j$ et $i < j$. On note $\text{inv}(\sigma)$ le nombre d'inversions de σ . Tout d'abord, on relie ce nombre avec un algorithme de tri : le tri à bulles. Puis, on s'intéresse à la table d'inversions d'une permutation, un objet qui caractérise une permutation.

Dans la suite, une permutation de taille n est représentée en Python par une liste sans répétition contenant tous les entiers compris entre 0 et $n-1$. Par exemple, la liste $[1, 0, 3, 2]$ représente la permutation $\sigma = 1032$.

Si A est un ensemble fini, $\text{Card}(A)$ désigne le cardinal de l'ensemble A . Pour tout entier $n \geq 1$, on pose $E_n = \prod_{k=1}^n \{0, 1, \dots, n-k\}$. Par exemple, on a $E_3 = \{0, 1, 2\} \times \{0, 1\} \times \{0\}$.

I.1 - Tri et inversions

Q1. Déterminer l'ensemble des inversions de la permutation $\sigma = 140253$.

On rappelle l'algorithme de tri à bulles :

Algorithme 1 - Tri à bulles

Entrées : Une liste d'entiers L

pour i allant de (taille de L)-1 à 1 (bornes incluses) **faire**

pour j allant de (taille de L)-1 à (taille de L)- i (bornes incluses) **faire**

si $L[j] < L[j-1]$ **alors**

| Echanger $L[j]$ et $L[j-1]$

fin

fin

fin

Q2. Écrire une fonction Python `tri_bulle(L)` qui prend en argument une liste d'entiers L et qui trie cette liste à l'aide du tri à bulles. L'issue de la fonction, la liste est triée.

Dans la suite, on admet que l'algorithme du tri à bulles est bien un algorithme de tri.

Q3. Montrer que si on effectue exactement un échange dans une liste lors du tri à bulles, la nouvelle liste a exactement une inversion en moins.

Q4. En déduire une fonction Python `nombre_inversions(L)` qui prend en argument une liste L correspondant à une permutation et renvoyant le nombre d'inversions de celle-ci. Cette fonction sera une légère modification du tri à bulles.

I.2 - Table d'inversions d'une permutation

Définition 1 (Table d'inversions). Soit σ une permutation de taille $n \geq 1$. La **table d'inversions** de σ est le n -uplet $(\alpha_0, \dots, \alpha_{n-1})$ tel que :

$$\forall i \in \{0, \dots, n-1\}, \alpha_i = \text{Card}(\{j \in \{i+1, \dots, n-1\} \mid \sigma_j < \sigma_i\}).$$

Elle est notée \mathbf{Tab}_σ . De plus, pour tout $i \in \{0, \dots, n-1\}$, $\mathbf{Tab}_\sigma[i]$ désigne α_i .

Pour tout $n \geq 1$, on désigne par $\mathbf{Tab} : \mathfrak{S}_n \rightarrow E_n$ l'application qui associe à une permutation de taille n sa table d'inversions.

- Q5.** Déterminer la table d'inversions de la permutation $\sigma = 140253$.
- Q6.** Montrer que pour toute permutation σ de taille $n \geq 1$, \mathbf{Tab}_σ est bien un élément de E_n .
- Q7.** Soit σ une permutation de taille $n \geq 1$. Montrer que $\sigma_0 = \mathbf{Tab}_\sigma[0]$.
- Q8.** Montrer que pour tout entier $n \geq 1$, l'application **Tab** est bijective.
- Q9.** Écrire une fonction Python `permutation_vers_table(L)` qui prend en argument une permutation représentée par la liste L et qui renvoie la table d'inversions correspondante.
- Q10.** Écrire une fonction Python `table_vers_permutation(L)` qui prend en argument une liste L qui correspond à une table d'inversions et qui renvoie la permutation qui lui est associée.

Partie II - Théorie des automates et des langages rationnels

Dans toute cette partie, la lettre ε désigne le mot vide, Σ désigne un alphabet et Σ^* l'ensemble des mots finis sur Σ .

II.1 - Définitions

Définition 2 (Automate déterministe). Un **automate déterministe** A est un quintuplet $A = (Q, \Sigma, q_0, F, \delta)$, avec :

- Q un ensemble d'états ;
- Σ un alphabet ;
- q_0 l'état initial ;
- $F \subseteq Q$ un ensemble d'états finaux ;
- $\delta : Q \times \Sigma \rightarrow Q$ une application de transition.

Définition 3 (Langage). Un **langage** sur Σ est une partie de Σ^* .

Définition 4 (Application de transition étendue aux mots). Soit $A = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe. On définit de manière récursive $\delta^* : Q \times \Sigma^* \rightarrow Q$ par :

$$\begin{aligned} \forall q \in Q, \quad \delta^*(q, \varepsilon) &= q \\ \forall q \in Q, \forall a \in \Sigma, \forall w \in \Sigma^*, \quad \delta^*(q, aw) &= \delta^*(\delta(q, a), w). \end{aligned}$$

Cette application δ^* vérifie alors la propriété admise suivante :

$$\forall q \in Q, \forall v \in \Sigma^*, \forall w \in \Sigma^*, \delta^*(q, vw) = \delta^*(\delta^*(q, v), w).$$

Définition 5 (Langages rationnels). On rappelle que les langages rationnels sont définis de manière inductive par :

- l'ensemble \emptyset est un langage rationnel ;
- les langages $\{a\}$ où a est une lettre, sont rationnels ;
- si L et L' sont des langages rationnels, $L \cdot L'$, $L \cap L'$, $L \cup L'$ sont des langages rationnels ;
- si L est un langage rationnel, L^* est un langage rationnel.

Définition 6 (Carré d'un langage). Soit L un langage sur Σ . Le **carré du langage** L est l'ensemble $\{uu, u \in L\}$. Il est noté $L \odot L$.

Définition 7 (Racine carrée d'un langage). Soit L un langage sur Σ . La **racine carrée du langage** L est l'ensemble $\{u \in \Sigma^* | uu \in L\}$. Elle est notée \sqrt{L} .

On pourra utiliser sans démonstration le théorème ci-dessous :

Théorème 1. Soit L un langage. Il est rationnel si et seulement s'il existe un automate déterministe et fini le reconnaissant.

Dans la suite, on décrit un langage rationnel par une expression rationnelle.

II.2 - Racine carrée d'un langage

Exemples

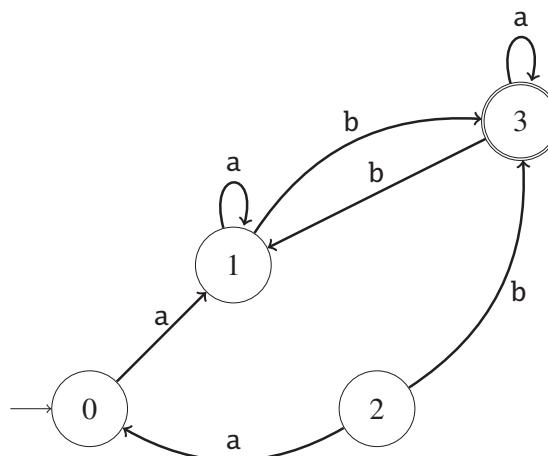
Q11. Décrire \sqrt{L} lorsque $\Sigma = \{a, b\}$ et L est décrit par l'expression rationnelle a^*b^* .

Q12. Décrire \sqrt{L} lorsque $\Sigma = \{a, b\}$ et L est décrit par l'expression rationnelle $b^*a^*b^*$.

Construction d'automates

Définition 8. Soient $A = (Q, \Sigma, q_0, F, \delta)$ un automate fini déterministe, q' un élément de Q et F' une partie de Q . L'automate $(Q, \Sigma, q', F', \delta)$ est noté $A_{q',F'}$. Si on note L le langage reconnu par A , $L_{q',F'}$ désigne le langage reconnu par $A_{q',F'}$.

Ici, L désigne le langage reconnu par l'automate A suivant :



Q13. Construire un automate reconnaissant $L_{3,\{1\}}$ en modifiant légèrement l'automate A .

Q14. On veut construire l'automate de Glushkov de L décrit par $a(a + ba^*b)^*ba^*$.

1. Décrire L' , le linéarisé de L .
2. Déterminer les préfixes de L' de longueur 1, les suffixes de L' de longueur 1 et les facteurs de L' de longueur 2.
3. En déduire l'automate de Glushkov G de L .

Q15. Déterminiser l'automate G .

Propriétés de la racine carrée d'un langage rationnel

Ici, on fixe L un langage rationnel sur un alphabet Σ et $A = (Q, \Sigma, q_0, F, \delta)$ un automate fini reconnaissant celui-ci.

Q16. Soit u un mot de Σ^* . Montrer que u est un élément de \sqrt{L} si et seulement s'il existe un $q \in Q$ tel que $u \in L_{q_0,\{q\}}$ et $u \in L_{q,F}$.

Q17. En déduire que \sqrt{L} est un langage rationnel.

Q18. Montrer que l'on a $(\sqrt{L} \odot \sqrt{L}) \subset L$.

Partie III - Algorithmique des mots sans facteur carré

L'objectif de cette partie est de construire différents algorithmes pour vérifier si un mot comporte des facteurs carrés ou non.

Dans toute la suite, $\Sigma = \{a_1 < \dots < a_p\}$ désigne un alphabet totalement ordonné comportant p lettres, ε représente le mot vide et Σ^* est l'ensemble des mots finis obtenus à partir de Σ . Pour tout réel x , on note $\lfloor x \rfloor$ la partie entière de x .

III.1 - Définitions

Définition 9 (Longueur d'un mot). Soit $w = w_0 \cdots w_{n-1}$ un mot de Σ^* . La **longueur** n de w est notée $|w|$, pour tout $0 \leq i \leq j < n$, $w[i, j]$ désigne le mot $w_i \cdots w_j$. Par convention, si $j < i$, $w[i, j]$ désigne ε .

Définition 10 (Mot carré). Soit w un mot de Σ^* . On dit que w est un **carré** s'il existe un mot x tel que $w = x \cdot x$.

Définition 11 (Facteur d'un mot). Soient v et w deux mots de Σ^* . On dit que v est un **facteur** de w s'il existe r et s deux mots (éventuellement vides) tels que $w = rvs$.

Définition 12 (Répétition). On dit qu'un mot w contient une **répétition** s'il contient un facteur carré différent de ε .

Dans la suite, un mot sera représenté en Caml par la liste de ses lettres. Par exemple, le mot *baba* est représenté par la liste `['b'; 'a'; 'b'; 'a']` et le mot vide est représenté par la liste `[]`.

III.2 - Fonctions utiles sur les listes

Q19. Écrire une fonction récursive Caml de signature `longueur` : `'a list -> int` qui renvoie la longueur de la liste.

Q20. Écrire une fonction Caml de signature `sous_liste` : `'a list -> int -> int -> 'a list` où `sous_liste L k long` renvoie une liste `S` qui est la sous-liste de `L` commençant à l'indice `k` et de longueur `long`. On suppose que l'indexation des listes commence à 0.

On pourra dans la suite de l'énoncé utiliser les fonctions `longueur` et `sous_liste`.

III.3 - Un algorithme naïf

Q21. Préciser si les mots suivants contiennent ou non une répétition.

1. `aabfa`
2. `abfdanq`
3. `ababa`
4. `avba`.

Q22. Soit w un mot contenant au plus deux lettres différentes. Montrer que si $|w| \geq 4$ alors w contient au moins une répétition.

Q23. Écrire une fonction Caml de signature `estCarre` : `'a list -> bool` prenant en argument une liste w et retournant `true` si w est un carré et `false` sinon.

Q24. Déterminer la complexité en nombre de comparaisons de lettres de la fonction `estCarre`.

Q25. Écrire une fonction Caml de signature `contientRepetitionAux` : `'a list -> int -> bool` prenant en argument une liste w et un entier m et retournant `true` si w contient une répétition de la forme xx avec x de longueur m et `false` sinon.

Q26. Montrer que toute répétition d'un mot w de longueur n est de la forme xx avec $|x| \leq \frac{n}{2}$.

Q27. En déduire une fonction Caml de signature `contientRepetition` : `'a list -> bool` prenant en argument une liste w retournant `true` si w contient une répétition et `false` sinon.

Q28. Quelle est la complexité en nombre de comparaisons de caractères de la fonction `contientRepetition` ?

III.4 - Algorithme de Main-Lorentz

L'algorithme de Main-Lorentz permet de détecter de manière plus efficace des répétitions d'un mot w . Il comporte essentiellement deux parties :

- la première consiste à voir si étant donné deux mots u et v , le mot uv contient un carré non nul issu de la concaténation ;
- la deuxième s'appuie sur le principe de “diviser pour régner”.

Remarquons qu'un mot uv contient une répétition si et seulement si u ou v contiennent une répétition ou uv contient des répétitions provenant de la concaténation. Pour déterminer si un mot uv contient de nouvelles répétitions, on commence par effectuer des prétraitements consistant à calculer des tables de valeurs de u et de v qui sont généralement appelées tables de préfixes (ou suffixes). Avant de présenter des algorithmes permettant de générer ces tables, on commence par justifier leur application dans la détection de répétitions.

Définition 13 (Carré centré). Soient u et v deux mots. On dit que uv contient un **carré centré** sur u (respectivement sur v) s'il existe un mot w non vide et des mots u', v'', w', w'' tels que $u = u'ww'$, $v = w''v''$, $w = w'w''$ (respectivement $u = u'w'$, $v = w''wv''$, $w = w'w''$).

Définition 14 (Plus long préfixe commun, plus long suffixe commun). Soient u et v deux mots de Σ^* . Le **plus long préfixe** (respectivement suffixe) **commun** de u et v est le plus long mot w tel qu'il existe deux mots r et s tels que $u = wr$ et $v = ws$ (respectivement $u = rw$ et $v = sw$). On le note $\text{lcp}(u, v)$ (respectivement $\text{lcs}(u, v)$).

À propos des carrés centrés

Q29. Dans cette question, $\Sigma = \{a, b\}$. Soient $u = abababaa$ et $v = ababaaa$. Déterminer le plus grand préfixe commun de u et v .

Q30. Soient u et v deux mots de Σ^* . Montrer que uv contient un carré centré sur u si et seulement s'il existe $i \in \{0, \dots, |u| - 1\}$ tel que $|\text{lcs}(u[0, i - 1], u)| + |\text{lcp}(u[i, |u| - 1], v)| \geq |u| - i$.

De la même manière, on peut montrer que uv contient un carré centré sur v si et seulement s'il existe $j \in \{1, \dots, |v| - 1\}$ tel que $|\text{lcs}(v[0, j - 1], u)| + |\text{lcp}(v, v[j, |v| - 1])| \geq |v| - j$.

Ainsi, pour pouvoir déterminer s'il existe un carré centré sur u ou v , on peut utiliser les valeurs :

$$|\text{lcs}(u[0, i - 1], u)|, |\text{lcp}(u[i, |u| - 1], v)|, |\text{lcs}(v[0, j - 1], u)|, |\text{lcp}(v, v[j, |v| - 1])|.$$

Dans la suite, étant donné deux mots u et v , on note pref_u , $\text{pref}_{u,v}$, suff_u et $\text{suff}_{u,v}$ les tableaux vérifiant :

$$\forall i \in \{0, \dots, |u| - 1\}, \quad \text{pref}_u[i] = |\text{lcp}(u[i, |u| - 1], u)|, \quad \text{pref}_{u,v}[i] = |\text{lcp}(u[i, |u| - 1], v)| \\ \text{suff}_u[i] = |\text{lcs}(u[0, i], u)|, \quad \text{suff}_{u,v}[i] = |\text{lcs}(u[0, i], v)|.$$

Calcul de table de préfixes

On présente un algorithme permettant le calcul de la table pref_u ainsi que sa complexité en nombre de comparaisons de caractères en page 8. En adaptant cet algorithme, il est également possible de calculer la table $\text{pref}_{u,v}$ en $O(|u|)$ de comparaisons de caractères.

Q31. On pose $u = aabbba$ et $v = abbaab$. Déterminer les tableaux pref_u et $\text{pref}_{u,v}$ sans justification.

Q32. En déroulant l'**algorithme 2** de la page suivante appliqué au mot $u = aaabaaabaaab$, compléter le tableau

$i =$	f	g	$\text{pref}[i]$
0	—	0	12
1	1	3	2
2	·	·	·
⋮	⋮	⋮	⋮
11	4	12	0

de la façon suivante : pour une valeur i donnée, on indique les valeurs de $f, g, \text{pref}[i]$ à l'issue des instructions internes de la boucle.

Par exemple, à l'initialisation, $i = 0$, f n'est pas définie, g vaut 0 et $\text{pref}[0] = 12$. Pour $i = 1$, à l'issue des instructions internes à la boucle, on a $f = 1, g = 3, \text{pref}[1] = 2$.

Q33. Déduire de l'**algorithme 2** une procédure calculant suff_u .

Dans la suite, on suppose que l'algorithme `tabpref(u, v)` qui prend en argument deux chaînes de caractères u et v et qui renvoie la table $\text{pref}_{u,v}$ nous est donné. On admet que la complexité de cet algorithme est de $O(|u|)$ en nombre de comparaisons de caractères.

Q34. Déduire des questions précédentes un algorithme qui, étant donnés deux mots u et v , renvoie VRAI s'il existe un carré centré sur u et FAUX sinon.

Q35. Quelle est la complexité de cet algorithme en nombre de comparaisons de caractères ?

Application des tables

Q36. Déduire des questions précédentes un algorithme récursif qui prend en argument une chaîne de caractères et qui renvoie VRAI si la chaîne contient une répétition et FAUX sinon.

Q37. Déterminer la complexité de cet algorithme en nombre de comparaisons de caractères.

Algorithme 2 - Calcul de la table pref_u

Entrées : une chaîne de caractères u

Sorties : un tableau pref_u

$i \leftarrow 0$, $\text{pref} \leftarrow$ tableau de taille $|u|$ initialisé à 0, $\text{pref}[i] \leftarrow |u|$, $g \leftarrow 0$

pour i allant de 1 à $|u| - 1$ **faire**

si $i < g$ et $\text{pref}[i-f] < g - i$ **alors**

| $\text{pref}[i] \leftarrow \text{pref}[i-f]$

fin

sinon si $i < g$ et $\text{pref}[i-f] > g - i$ **alors**

| $\text{pref}[i] \leftarrow g - i$

fin

sinon

$(f, g) \leftarrow (i, \max(g, i))$

tant que $g < |u|$ et $u[g] == u[g-f]$ **faire**

| $g \leftarrow g + 1$

fin

$\text{pref}[i] \leftarrow g - f$

fin

fin

On admet que la complexité est de $O(|u|)$ en nombre de comparaisons de caractères.

FIN



Épreuve d’Informatique MP

Durée 3 h

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

L’usage de calculatrices est interdit.

AVERTISSEMENT

- L'épreuve est composée de 3 exercices indépendants.
- Un candidat pourra toujours admettre le résultat des questions qu'il n'a pas faites pour faire les questions suivantes.
- Les programmes devront être écrits dans le langage de programmation Python pour l'exercice 1 et OCaml pour les exercices 2 et 3.

La **présentation**, la lisibilité, l'orthographe, la qualité de la **rédaction**, la clarté et la précision des raisonnements entreront pour une **part importante** dans l'**appréciation des copies**. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

EXERCICE 1 – AUTOOUR DE LA RECHERCHE PAR DICHOTOMIE

Partie 1 – Questions de cours

- 1** Rappeler le principe de la recherche dichotomique dans une liste d'entiers. Quel intérêt présente cette méthode ?
- 2** La mise en œuvre d'une recherche dichotomique est-elle possible sur une liste de couples d'entiers ? de chaînes de caractères ?

Partie 2 – Étude d'une fonction dicho

Voici le code d'une fonction Python élaboré pour tester par dichotomie si un entier x se trouve dans une liste d'entiers `liste` :

```
(1) def dicho(liste,x):
(2)     # Pré-conditions: x est un entier, liste est une
(3)     # liste d'entiers triée dans l'ordre croissant
(4)     n = len(liste)
(5)     if n == 0:
(6)         return False
(7)     g, d = 0, n-1
(8)     while d-g > 0:
(9)         m = (g+d)//2
(10)        if liste[m] >= x:
(11)            d = m
(12)        else:
(13)            g = m
(14)    return liste[g] == x
```

- 3** Pour quelles raisons ne remplace-t-on pas la précondition de la ligne (3) par un appel à une fonction qui trierait la liste `liste` dans l'ordre croissant ?
- 4** Justifier que le prédicat
 \mathcal{P} : « l'entier x apparaît dans la sous-liste `liste[g : d+1]` des éléments de `liste` d'indices g à d » est préservé à chaque tour de la boucle `while`.
- 5** Il s'avère que la fonction `dicho` ne termine pas. Donner un exemple où la fonction boucle.
- 6** Indiquer sans justification la ou les corrections à apporter pour que la fonction `dicho` termine, tout en restant correcte.
- 7** Justifier que la fonction corrigée termine et est correcte.

Partie 3 – Extensions du principe

Une première extension consiste à réduire la taille du problème non plus en 2 mais en 3 : c'est le principe de trichotomie.

8 Écrire en Python une fonction `tricho(liste,x)` qui renvoie `True` si l'élément `x` se trouve dans la liste `liste` et `False` sinon. Cette fonction sera récursive ou fera appel à une ou des fonctions auxiliaires récursives.

9 Estimer la complexité de la fonction `tricho(liste,x)` pour une liste `liste` à n éléments. Comparer avec la méthode par dichotomie.

Une seconde extension consiste à adapter le principe de dichotomie au cas d'une matrice d'entiers dont les éléments sont triés en colonne de haut en bas et de gauche à droite. L'illustration ci-dessous indique l'ordre des éléments d'une matrice à 4 lignes et 6 colonnes :

$$\begin{pmatrix} 0 & 4 & 8 & 12 & 16 & 20 \\ 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \end{pmatrix}$$

Une matrice sera implémentée en Python par une liste de ses lignes, elles-mêmes implémentées par des listes.

10 Écrire en Python une fonction `dicho_matrice(mat,x)` qui prend en argument un entier `x` et une matrice `mat` à n lignes et p colonnes ($n \geq 1$ et $p \geq 1$), d'entiers triés en colonne de haut en bas et de gauche à droite, et qui renvoie :

- le couple d'indices (i,j) minimal pour l'ordre défini plus haut tel que `x` se trouve en ligne `i` et colonne `j`, si l'entier `x` est bien présent dans la matrice `mat` ;
- le couple $(-1,-1)$ si `x` n'est pas présent dans la matrice `mat`.

Cette fonction devra être de complexité logarithmique en $\max(n,p)$.

EXERCICE 2 – AUTOMATES ET LANGAGES DE MOTS BINAIRES

Dans cet exercice, on étudie différents langages sur l'alphabet $A = \{0, 1\}$ à deux lettres. On note A^* l'ensemble des mots construits sur l'alphabet A . Le mot vide est noté ε . En OCaml, un mot sur A est implémenté par le type

```
type mot = bool list;;
```

à savoir par une liste de booléens où la lettre 0 est représentée par le booléen `false` et la lettre 1 par le booléen `true`.

11 On note L_1 le langage des mots de A qui représentent l'écriture binaire d'un entier naturel, où le bit de poids faible se situe en fin de mot. Pour assurer l'unicité de la représentation, l'écriture binaire d'un entier ne commence jamais par 0. C'est pourquoi l'entier nul est représenté par le mot vide.

- 11a)** Donner l'écriture binaire de l'entier 41.
- 11b)** Donner l'entier représenté par le mot 10101010.
- 11c)** Pour un automate, que signifie la propriété d'être local standard ?
- 11d)** Dessiner un automate local standard \mathcal{A}_1 reconnaissant le langage L_1 .
- 11e)** Écrire en OCaml une fonction `langage_1` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_1 .

12 On note L_2 le langage dénoté par l'expression rationnelle $(0 + 1)^* \cdot 0$.

- 12a)** Justifier que L_2 est un langage local.
- 12b)** Dessiner un automate déterministe \mathcal{A}_2 reconnaissant le langage L_2 .
- 12c)** Écrire en OCaml une fonction `langage_2` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_2 .

13 On note L_3 le langage reconnu par l'automate déterministe \mathcal{A}_3 défini par

- l'ensemble d'états $Q = \{0, 1, 2\}$;
- l'état initial $i = 0$;
- l'ensemble d'états finals $F = \{0\}$;
- la fonction de transition $\delta : Q \times A \longrightarrow Q$ définie par

$$\forall q \in Q \quad \forall a \in A \quad \delta(q, a) = (2q + a) \bmod 3$$

On rappelle que $(n \bmod 3)$ désigne le reste de la division euclidienne de l'entier n par 3.

- 13a)** Dessiner l'automate \mathcal{A}_3 .
- 13b)** Écrire en OCaml une fonction `langage_3` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_3 .
- 13c)** Pour tout $n \in \mathbb{N}^*$, démontrer la propriété $\mathcal{P}(n)$ suivante :

$$\forall \omega_1, \dots, \omega_n \in A \quad \delta^*(0, \omega_1 \cdots \omega_n) = \sum_{k=1}^n \omega_k 2^{n-k} \bmod 3$$

où la fonction $\delta^* : Q \times A^* \longrightarrow Q$ est définie par

$$\forall q \in Q \quad \forall \omega \in A^* \quad \forall a \in A \quad \delta^*(q, \varepsilon) = q \quad \text{et} \quad \delta^*(q, \omega \cdot a) = \delta(\delta^*(q, \omega), a)$$

14 On note $L_4 = L_1 \cap L_2 \cap L_3$.

- 14a)** Décrire simplement l'ensemble des mots du langage L_4 .
- 14b)** Existe-t-il un automate reconnaissant le langage L_4 ?

EXERCICE 3 – DIAMÈTRE D’UN GRAPHE

Dans cet exercice, on considère des graphes non orientés connexes. Les sommets d’un graphe à n sommets ($n \in \mathbb{N}^*$) sont numérotés de 0 à $n - 1$. On suppose qu’aucune arête ne boucle sur un même sommet.

Un *chemin de longueur* $p \in \mathbb{N}$ d’un sommet a vers un sommet b dans un graphe est la donnée de $p + 1$ sommets s_0, s_1, \dots, s_p tels que $s_0 = a$, $s_p = b$ et, pour tout $1 \leq k \leq p$, les sommets s_{k-1} et s_k sont reliés par une arête.

Un *plus court chemin* d’un sommet a vers un sommet b dans un graphe G est un chemin de longueur minimale parmi tous les chemins de a vers b . Sa longueur est notée $d_G(a, b)$.

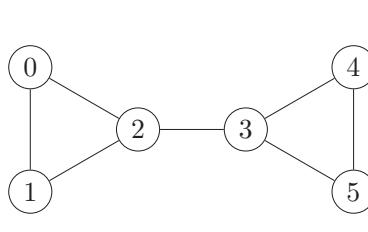
Le *diamètre* d’un graphe G , noté $\text{diam}(G)$, vaut le maximum des longueurs des plus courts chemins entre deux sommets du graphe G . Autrement dit,

$$\text{diam}(G) = \underset{a, b \text{ sommets de } G}{\text{Max}} d_G(a, b)$$

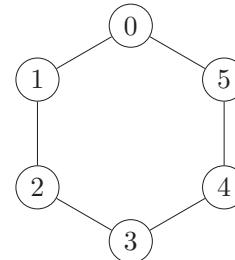
Un *chemin maximal* d’un graphe G est un plus court chemin de G de longueur $\text{diam}(G)$.

Partie 1 – Exemples de graphes

15 Donner sans justification le diamètre et les chemins maximaux pour chacun des deux graphes G_1 et G_2 ci-dessous.



graphe G_1



graphe G_2

En OCaml, les graphes sont représentés par liste d’adjacence et implémentés par le type

`type graphe = int list array;;`

16 Graphes de diamètre maximal.

16a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus grand possible.

16b) Écrire en OCaml une fonction `diam_max` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre maximal.

17 Graphes de diamètre minimal.

17a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus petit possible.

17b) Écrire en OCaml une fonction `diam_min` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre minimal.

Partie 2 – Algorithmes de calcul du diamètre

Dans cette partie, on suppose que les graphes sont représentés par listes d'adjacence.

- 18** Donner l'entrée et la sortie de l'algorithme de Dijkstra. Comment cet algorithme permet-il de calculer le diamètre d'un graphe ?
- 19** Quel parcours de graphe peut être utilisé pour le calcul du diamètre ?
- 20** Laquelle des deux méthodes précédentes est la mieux adaptée pour calculer le diamètre d'un graphe ?

Partie 3 – Diamètre d'un arbre binaire

Dans cette partie, on s'intéresse aux arbres binaires, qui sont des cas particuliers de graphes. On travaille avec une représentation spécifique de ces graphes particuliers, implémentée en OCaml par le type suivant :

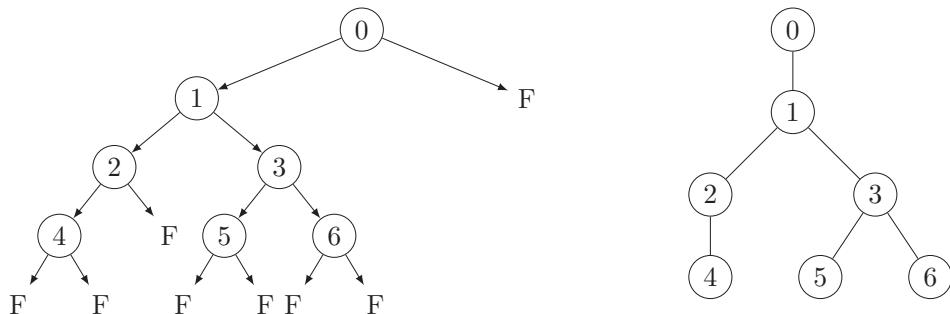
```
type arbre = Feuille | Noeud of int * arbre * arbre ;;
```

Le *graphe sous-jacent* $G_{\mathcal{A}}$ à un arbre binaire \mathcal{A} est défini comme le graphe orienté dont

- les sommets correspondent aux nœuds de l'arbre (et pas aux feuilles) ;
- les arêtes correspondent aux branches de l'arbre reliant deux nœuds (et non pas celles reliant un nœud à une feuille).

Le diamètre d'un arbre binaire est alors défini comme le diamètre de son graphe sous-jacent.

Voici un exemple d'arbre binaire \mathcal{A} (à gauche) et de son graphe sous-jacent $G_{\mathcal{A}}$ (à droite) :



- 21** Donner l'expression OCaml représentant l'arbre \mathcal{A} de l'exemple. Donner le diamètre de \mathcal{A} et les chemins maximaux du graphe sous-jacent $G_{\mathcal{A}}$.

- 22** Quel est le nombre r d'arêtes du graphe sous-jacent à un arbre binaire possédant n nœuds ?

Une première approche pour calculer le diamètre d'un arbre consiste à le transformer en un graphe et à employer un algorithme général sur les graphes de la partie 2.

- 23** Écrire en OCaml une fonction `nb_noeuds` de type `arbre -> int` qui renvoie le nombre de nœuds d'un arbre binaire donné en argument.

- 24** Écrire en OCaml une fonction `numerotation` de type `arbre -> arbre` qui prend en argument un arbre binaire \mathcal{A} à n nœuds et qui renvoie un arbre binaire \mathcal{A}' de même graphe sous-jacent que \mathcal{A} et dont les nœuds sont étiquetés de 0 à $n - 1$.

25 Écrire en OCaml une fonction `arbre_vers_graphe` de type `arbre -> graphe` qui prend en argument un arbre binaire \mathcal{A} à n nœuds étiquetés de 0 à $n - 1$ et qui renvoie le graphe $G_{\mathcal{A}}$ sous-jacent à \mathcal{A} (le type `graphe` est défini dans la partie 1).

26 Décrire un algorithme qui calcule le diamètre d'un arbre de type `arbre` en se ramenant à un graphe. Quelle est sa complexité ?

Une seconde approche pour calculer le diamètre d'un arbre consiste à employer une technique divisor-pour-régner. Pour tout arbre \mathcal{A} non réduit à une feuille, de la forme `Noeud (x, arbre_g, arbre_d)`, on note

- \mathcal{A}_g le fils gauche de \mathcal{A} , représenté par `arbre_g`;
- \mathcal{A}_d le fils droit de \mathcal{A} , représenté par `arbre_d`.

La hauteur de l'arbre \mathcal{A} , notée $h(\mathcal{A})$, est la longueur du plus long chemin descendant de la racine vers une feuille. Dans l'arbre exemple de la partie 3, l'arbre \mathcal{A} est de hauteur 4.

27 Quelle est la longueur d'un chemin maximal passant par la racine ?

28 Écrire en OCaml une fonction `diam_arbre` de type `arbre -> int` qui calcule le diamètre d'un arbre donné en argument. Cette fonction devra être de complexité linéaire en le nombre de nœuds de l'arbre.

FIN D'ÉPREUVE

Cette épreuve est constituée de deux problèmes indépendants.

On rappelle que tous les programmes demandés doivent être rédigés dans le langage Python.

Problème 1 : base 3 équilibrée

Notation. \mathbb{N}^* désigne l'ensemble des entiers naturels non nuls.

En informatique, on utilise traditionnellement la base 2 pour représenter les entiers. Ce choix est dicté par la technologie utilisée par les ordinateurs actuels. Dans les années 60, d'autres technologies avaient été envisagées, dont l'utilisation d'une base de numération appelée base 3 équilibrée.

En base 3 classique, on écrit les nombres à l'aide de trois chiffres : 0, 1 et 2 : par exemple, $23 = \overline{212}^3 = 2 \times 3^2 + 1 \times 3 + 2$, $46 = \overline{1201}^3 = 1 \times 3^3 + 2 \times 3^2 + 0 \times 3 + 1$.

En base 3 équilibrée, les « chiffres » utilisés sont 0, 1 et -1 :

$$23 = \overline{10(-1)(-1)}^e = 1 \times 3^3 + 0 \times 3^2 - 1 \times 3 - 1, \quad 46 = \overline{1(-1)(-1)01}^e = 1 \times 3^4 - 1 \times 3^3 - 1 \times 3^2 + 0 \times 3 + 1.$$

Dans toute la suite, on s'intéresse à l'écriture en base 3 équilibrée des entiers naturels **non nuls**. On note $C = \{-1, 0, 1\}$ l'ensemble des chiffres utilisés dans l'écriture en base 3 équilibrée.

L'écriture en base 3 équilibrée d'un entier naturel **non nul** sera notée comme ci-dessus, par la suite des chiffres surmontée d'un trait avec un exposant e . Si a_0, a_1, \dots, a_{p-1} sont dans C , on a donc :

$$\overline{a_0 a_1 \dots a_{p-1}}^e = \sum_{k=0}^{p-1} a_k 3^{p-1-k}.$$

L'écriture en base 3 équilibrée d'un entier naturel non nul, lire de gauche à droite, commence toujours par le chiffre 1.

Partie I – généralités

Question 1.

1.a Donner la valeur de $\overline{1(-1)0(-1)}^e$, de $\overline{1111}^e$ et de $\overline{1(-1)(-1)(-1)(-1)}^e$.

1.b Écrire en base 3 équilibrée les entiers de 1 à 9.

Question 2. Soit k un entier naturel.

Déterminer la valeur de $A_k = \overline{1 \underbrace{11 \dots 1}_{k \text{ chiffres}}}^e$ et de $B_k = \overline{1 \underbrace{(-1)(-1) \dots (-1)}_{k \text{ chiffres}}}^e$.

(On aura bien noté que ces deux écritures possèdent $k+1$ chiffres au total.)

Question 3. On représente l'écriture en base 3 équilibrée $\overline{a_0 \dots a_{p-1}}^e$ par la liste Python $[a_{p-1}, a_{p-2}, \dots, a_0]$ (attention, les chiffres sont écrits dans la liste en lisant de droite à gauche l'écriture en base 3 équilibrée, le dernier d'entre eux est donc toujours le chiffre 1).

Écrire une fonction `valeur(L)` qui prend en argument une liste de chiffres et renvoie l'entier naturel non nul dont c'est une écriture en base 3 équilibrée.

Par exemple `valeur([1, 0, -1, -1, 1])` renvoie l'entier 46.

Partie II – existence et unicité de l'écriture en base 3 équilibrée

Question 4. On veut montrer que tout entier naturel non nul n admet une écriture en base 3 équilibrée.

Soit $n \in \mathbb{N}^*$. On note q et r le quotient et le reste dans la division euclidienne de n par 3 : $n = 3q + r$ et $r \in \{0, 1, 2\}$.

4.a On suppose que q est non nul et admet une écriture en base 3 équilibrée $q = \overline{a_0 \dots a_{p-1}}^e$. Déterminer une écriture en base 3 équilibrée de n dans le cas où $r = 0$ ou $r = 1$.

4.b On suppose que q est non nul et que $q + 1$ admet une écriture en base 3 équilibrée $q + 1 = \overline{b_0 \dots b_{p-1}}^e$. Déterminer une écriture en base 3 équilibrée de n dans le cas où $r = 2$.

4.c Montrer par récurrence que tout entier naturel non nul admet une écriture en base 3 équilibrée.

Question 5. On souhaite écrire en Python une fonction `incremente(L)` qui prend en argument une liste `L` de chiffres représentant une écriture en base 3 équilibrée d'un entier naturel non nul n et qui renvoie une liste représentant une écriture en base 3 équilibrée de $n + 1$.

5.a On propose la fonction récursive `incrementeR(L)` suivante.

Expliquer la ligne 9. Pourquoi a-t-il fallu écrire les lignes 1 et 2 ?

```

0 def incrementeR(L):
1     if len(L)==0:
2         return [1]
3     elif L[0]==0:
4         return [1]+L[1:]
5     elif L[0]==-1:
6         return [0]+L[1:]
7     else:
8         # ici L[0]==1
9         return [-1]+incrementeR(L[1:])

```

5.b On souhaite écrire une version non récursive `incremente(L)` de cette fonction.

Recopier et compléter le script suivant pour répondre à cette question :

```

def incremente(L):
    p = len(L)
    M = [] # liste finale
    k = 0
    while k < p and L[k] == 1:
        M.append(-1)
        k += 1
    if k == p:
        M.append(...)
    elif L[k] == 0:
        M.append(...)
        M = M + L[k+1:]
    elif L[k] == -1:
        ...
        ...
    return M

```

Question 6. On souhaite montrer que l'écriture en base 3 équilibrée d'un entier naturel n non nul est unique.

6.a Soit a_0, \dots, a_{p-1} des chiffres de C (avec $a_0 = 1$ et $p \geq 1$). Quel est le reste dans la division euclidienne par 3 du nombre $\overline{a_0a_1\dots a_{p-1}}_e$?

6.b En déduire que l'écriture en base 3 équilibrée d'un entier naturel n non nul est unique.

Question 7. Écrire une fonction `base3e(n)` qui prend en argument un entier naturel n non nul et renvoie la liste `L` de chiffres qui correspond à l'écriture en base 3 équilibrée de n . Par exemple `base3e(46)` renvoie `[1,0,-1,-1,1]`.

Partie III – chemins de Delannoy

On considère le plan euclidien P rapporté à un repère orthonormé.

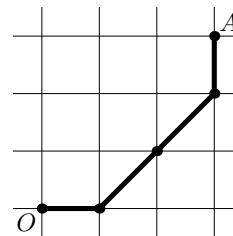
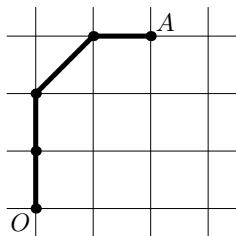
Un chemin de Delannoy du plan P est un arc suivant une ligne brisée partant de l'origine O et arrivant en un point A de coordonnées (a, b) où a et b sont deux entiers naturels, constituée d'arcs à choisir parmi trois types : un arc horizontal (associé au vecteur $(1, 0)$), un arc vertical (associé au vecteur $(0, 1)$) ou un arc diagonal (associé au vecteur $(1, 1)$). Dans le parcours d'un chemin de Delannoy, on va donc toujours vers la droite, vers le haut, ou en diagonale vers la droite et vers le haut en même temps.

On associe à tout chemin de Delannoy un entier naturel non nul n défini par son écriture en base 3 équilibrée de la façon suivante : le premier chiffre de cette écriture est toujours égal à 1 (comme pour toute écriture en base 3 équilibrée), les chiffres suivants caractérisent les arcs consécutifs du chemin, le chiffre 1 est associé à un segment horizontal, le chiffre -1 à un arc vertical, et le chiffre 0 à un arc diagonal. Inversement à tout entier naturel non nul on peut associer le chemin de Delannoy qui est associé à son écriture en base 3 équilibrée.

On fera attention que le premier chiffre de l'écriture en base 3 équilibrée, toujours égal à 1, n'est pas pris en compte dans la construction du chemin.

Remarque : $n = 1 = \overline{1}^e$ est l'entier associé à un chemin de Delannoy de longueur nulle.

Par exemple, les entiers associés aux deux chemins de Delannoy ci-dessous sont respectivement $46 = \overline{1}(-1)(-1)\overline{0}\overline{1}^e$ pour la figure de gauche et $107 = \overline{1}\overline{1}00(-1)^e$ pour la figure de droite.



Question 8.

8.a Dessiner le chemin de Delannoy associé à l'entier 2019.

8.b On note (a, b) les coordonnées entières du point d'arrivée d'un chemin de Delannoy. Écrire la fonction `arrivee(n)` qui renvoie ce couple de coordonnées quand on lui donne en argument l'entier non nul n associé au chemin de Delannoy. Par exemple `arrivee(46)` renvoie le couple $(2, 3)$ et `arrivee(107)` renvoie le couple $(3, 3)$.

On pourra s'inspirer de la fonction `base3e`.

Question 9.

Soit a et b deux entiers naturels, et A le point de coordonnées (a, b) . Il existe plusieurs chemins de Delannoy allant de l'origine au point A , chacun d'eux étant associé à un entier naturel. Soit $N(a, b)$ l'ensemble de ces entiers.

9.a Soit $a \in \mathbb{N}^*$. Déterminer $\min N(a, a)$ et $\max N(a, a)$.

9.b On suppose que les entiers a et b vérifient $0 < a < b$. Déterminer $\min N(a, b)$ et $\max N(a, b)$.

Problème 2 : compilation et algorithmes

Notation. Pour m et n deux entiers naturels, $\llbracket m, n \rrbracket$ désigne l'ensemble des entiers k tels que $m \leq k \leq n$.

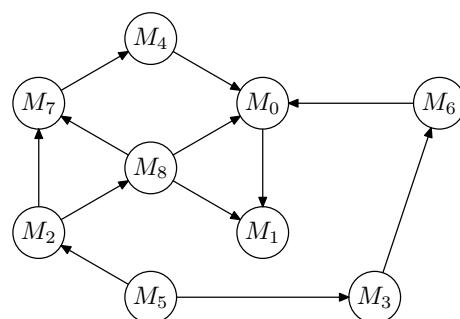
Partie A – ordre topologique sur un graphe

Lors de l'écriture d'un programme complexe, on le découpe souvent en modules distincts, mais dépendants les uns des autres. Le compilateur a besoin de connaître l'ordre dans lequel compiler ces modules. Pour ce faire, on utilise un graphe de dépendances, que nous décrivons brièvement ici.

On suppose qu'on dispose de n modules numérotés M_0, M_1, \dots, M_{n-1} et on représente leurs relations de dépendance à l'aide d'un graphe Γ dont les sommets sont les modules, et dont une arête allant du module M_i au module M_j indique que le module M_i doit être compilé avant le module M_j .

Par exemple, avec $n = 9$, aux contraintes du tableau de gauche ci-dessous, on peut associer le graphe Γ de droite.

le module	est compilé APRÈS les modules
M_0	M_4, M_6, M_8
M_1	M_0, M_8
M_2	M_5
M_3	M_5
M_4	M_7
M_5	—
M_6	M_3
M_7	M_2, M_8
M_8	M_2

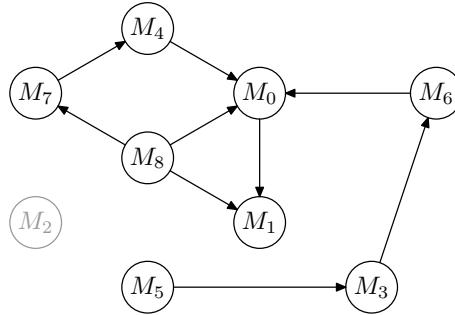


On aura besoin de travailler sur des sous-graphes G du graphe Γ . C'est pourquoi, pour toute la suite, on suppose fixé l'entier n (qu'on pourra supposer supérieur à 2), et les graphes considérés n'utiliseront que quelques-uns des n sommets M_0, M_1, \dots, M_{n-1} . Pour simplifier la rédaction, on pourra utiliser la notation suivante : si s est un sommet d'un graphe G , on note $[s]$ son numéro, de sorte que $s = M_{[s]}$.

On représente un graphe G dont les sommets sont pris parmi les n sommets listés ci-dessus par une liste $G=[\text{App}, \text{Succ}, p]$ où App est une liste de n valeurs booléennes (c'est-à-dire `True` ou `False`) ; Succ est une liste de n listes ; p est un entier. Plus précisément :

- pour $i \in \llbracket 0, n-1 \rrbracket$, $\text{App}[i]$ est `True` si et seulement si le module M_i est présent dans le graphe ;
- p est le nombre de sommets du graphe considéré, donc le nombre d'occurrences de `True` dans la liste App ;
- pour $i \in \llbracket 0, n-1 \rrbracket$, $\text{Succ}[i]$ est la liste vide si M_i n'est pas présent dans le graphe, et sinon $\text{Succ}[i]$ est la liste (éventuellement vide) des numéros j des sommets M_j tels qu'il y a un arc de M_i vers M_j .

Par exemple, le graphe G de la figure suivante (obtenu en supprimant dans Γ le sommet M_2) :



est représenté par la liste $G=[\text{App}, \text{Succ}, p]$ où

- $\text{App} = [\text{True}, \text{True}, \text{False}, \text{True}, \text{True}, \text{True}, \text{True}, \text{True}, \text{True}]$;
- $\text{Succ} = [[1], [], [], [6], [0], [3], [0], [4], [0, 1, 7]]$;
- $p=8$.

Dans toute la suite, on suppose que la valeur n , qui est fixée, est stockée dans une variable globale `n`. Les variables `i` et `j` désignent des numéros de sommets, donc des entiers de l'intervalle $\llbracket 0, n-1 \rrbracket$.

Question 10.

Écrire une fonction `mem(i,G)` qui prend en argument un entier i et la représentation G d'un graphe (telle qu'elle est décrite ci-dessus) et renvoie le booléen indiquant si le sommet de numéro i est dans le graphe.

Question 11.

Écrire une fonction `pred(i,G)` qui renvoie la liste, éventuellement vide, des numéros j des sommets du graphe représenté par G tels que l'arc qui va de M_j à M_i soit présent dans le graphe.

Par exemple, pour le graphe de la figure ci-dessus, `pred(0,G)` renvoie la liste `[4, 6, 8]`.

Question 12.

Écrire une fonction `sansSuccesseur(G)` qui renvoie le numéro i d'un sommet M_i du graphe qui n'a pas de successeur, s'il en existe, ou qui renvoie `-1` s'il n'y a pas de tel sommet. Dans l'exemple du graphe ci-dessus, `sansSuccesseur(G)` renvoie `1`.

Question 13.

Compléter la fonction suivante `suppr(i,G)` qui renvoie **un nouveau graphe H** obtenu à partir de G en supprimant le sommet de numéro i et toutes les flèches qui lui sont reliées.

```

def suppr(i, G):
    H = copy.deepcopy(G)
    H[2] = H[2] - 1
    ...
    ...
    ...
    return H

```

L'appel `H = copy.deepcopy(G)` permet d'instancier la variable `H` avec une copie de `G`.

Étant donné un graphe de dépendances G possédant p sommets (avec $p \leq n$), un ordre topologique sur le graphe est la donnée d'une remémoration des sommets du graphe qui respecte les arcs du graphe, c'est-à-dire une fonction N bijective, définie sur l'ensemble des numéros des sommets présents dans le graphe G , à valeurs dans $\llbracket 0, p-1 \rrbracket$ telle que s'il existe un arc du sommet M_i vers le sommet M_j on a $N(i) < N(j)$.

Par exemple, pour le graphe de la dernière figure ci-dessus, il existe un ordre topologique, défini par $N(0) = 6$, $N(1) = 7$, $N(3) = 1$, $N(4) = 5$, $N(5) = 0$, $N(6) = 2$, $N(7) = 4$ et $N(8) = 3$.

Un chemin dans un graphe est une suite d'au moins deux sommets reliés par des arcs consécutifs. Un cycle est un chemin qui retourne au sommet de départ.

Question 14.

Dans cette question, on considère un graphe orienté G possédant p sommets (avec $p \leq n$) qui sont pris parmi M_0, M_1, \dots, M_{n-1} . On suppose que chaque sommet admet au moins un successeur.

14.a On effectue une promenade dans le graphe : on choisit un sommet s_0 , puis un successeur s_1 de s_0 , puis un successeur s_2 de s_1 , etc. On peut ainsi obtenir une suite s_0, s_1, \dots, s_n . Montrer qu'il existe deux indices i et j tels que $0 \leq i < j \leq n$ et $s_i = s_j$.

14.b Justifier qu'il existe au moins un cycle dans ce graphe.

14.c Montrer qu'il est impossible de trouver un ordre topologique sur ce graphe.

Question 15.

Soit G un graphe ayant p sommets (avec $p \leq n$) parmi M_0, \dots, M_{n-1} . On suppose que G admet au moins un sommet s sans successeur. On appelle H le graphe obtenu à partir de G en supprimant le sommet s et tous les arcs reliés à s . Montrer que si H possède un ordre topologique N_H à valeurs dans $\llbracket 0, p-2 \rrbracket$, on peut l'étendre à un ordre topologique N sur G en posant $N([s]) = p-1$ et $N([s']) = N_H([s'])$ pour tous les sommets s' de H .

Question 16.

On peut déduire de cette étude un algorithme de détermination d'un ordre topologique N sur un graphe G , quand il en existe. Un tel ordre topologique est représenté par une liste L à n éléments : si le sommet M_i n'est pas dans le graphe G , on a $L[i] = -1$; si M_i est dans G , $L[i]$ contient la valeur $N(i)$.

16.a Recopier et compléter la fonction suivante pour répondre à la question : `ordreTopologique(G)` renvoie `None` s'il n'existe pas d'ordre topologique sur le graphe G et une liste L qui représente un ordre topologique s'il en existe un. La fonction `parcoursReussi` peut procéder à des appels récursifs.

```

def ordreTopologique(G):
    n = len(G[0])
    L = [-1] * n

    def parcoursReussi(G):
        p = G[2]
        if p != 0:
            s = sanssuccesseur(G)
            if s == -1:
                return ...
            else:
                ...
                ...
                return ...

        else:
            return ...

    b = parcoursReussi(G)
    if b:
        return L
    else:
        return None

```

16.b Prouver que l'algorithme termine toujours.

Partie B – allocation de registres

Quand le code est en phase finale de compilation, on peut supposer, pour simplifier, qu'il se présente comme une succession d'affectations de variables ou d'utilisation de ces variables à l'aide de fonctions simples (opérations arithmétiques ou logiques). L'accès à la mémoire étant plus coûteux en temps que le calcul lui-même, il est préférable de conserver le plus possible de résultats intermédiaires dans les registres du processeur plutôt que dans la mémoire vive. Mais le processeur n'a qu'un nombre limité de registres : il s'agit donc de savoir si tous les calculs peuvent être menés en se confinant à cet espace limité.

On se limite ici à un modèle extrêmement simplifié, où le code du programme ne comporte que des affectations de variables $x = \dots$ ou des appels de fonctions $f(a, b, \dots)$, comme le code Python suivant :

```

0 d = 1
1 b = 2 * d
2 a = 3
3 d = a * b
4 print(d)
5 c = 2 * a - b
6 print(a)
7 d = c + b
8 b = 2 * a
9 print(c, d)

```

Comme un programme ne peut utiliser qu'un nombre fini de variables, on suppose dans toute la suite qu'on dispose d'une numérotation des variables, ce qui permet de représenter une variable par un entier naturel. Dans l'exemple ci-dessus, on supposera que a est numérotée 0, b est numérotée 1, etc.

Une affectation de variable est représentée par un couple $(i, [j, k, \dots])$ constitué du numéro de la variable qui figure à gauche du symbole $=$ et de la liste des numéros des variables qui figurent à droite. Par exemple, l'affectation $c = 2 * a - b$ est représentée par le couple $(2, [0, 1])$.

Un appel de fonction est représenté par un couple $(None, [j, k, \dots])$ dont le deuxième élément est la liste des numéros des variables utilisées dans l'appel de fonction. Par exemple, l'appel $\text{print}(c, d)$ est représenté par le couple $(None, [2, 3])$.

Les constantes apparaissant dans les différentes instructions ne sont pas prises en compte dans cette représentation.

Le programme précédent est ainsi représenté par la liste de couples suivante :

```
prog = [(3,[]), (1,[3]), (0,[]), (3,[0,1]), (None,[3]), (2,[0,1]), (None,[0]), (3,[1,2]), (1,[0]), (None,[2,3])]
```

On cherche à savoir à quels moments il est utile de conserver dans les registres du processeur les valeurs des différentes variables.

Dans l'exemple du programme ci-dessus, si on s'intéresse à la variable d , on observe que :

- elle est initialisée en ligne 0, et cette valeur est utilisée en ligne 1 ;
- elle est modifiée en ligne 3, et la nouvelle valeur est utilisée en ligne 4 ;
- elle est modifiée en ligne 7, et la nouvelle valeur est utilisée en ligne 9.

La variable d a donc trois périodes de vie, qu'on peut représenter par les intervalles $]0, 1]$, $]3, 4]$ et $]7, 9]$.

La variable a a une seule période de vie : l'intervalle $]2, 8]$; la variable b n'a également qu'une période de vie : l'intervalle $]1, 7]$. En effet, b est modifiée en ligne 8 mais sa valeur n'est pas utilisée dans la suite, donc on ne tient pas compte de l'intervalle $]8, 9]$.

On dit qu'une variable est vivante pour chaque instruction d'une période de vie, et morte pour les autres.

L'algorithme suivant permet de déterminer pour chaque ligne de code si chaque variable est morte ou vivante.

- on commence par indiquer que chaque variable est morte ;
- pour chaque ligne de code, **en partant du bas** :
 - s'il s'agit d'un appel de fonction, chaque variable utilisée est marquée vivante ;
 - s'il s'agit d'une affectation, la variable affectée est marquée morte sauf si elle figure également à droite du symbole $=$, et toutes les variables à droite du symbole $=$ sont marquées vivantes.

Question 17.

Recopier et compléter les lignes 0 à 4 du tableau suivant qui indique l'état de chaque variable pour chaque instruction du programme ci-dessus. (Un V indique une variable vivante, un M une variable morte.)

ligne	a	b	c	d
0				
1				
2				
3				
4				
5	V	V	M	M
6	V	V	V	M
7	V	V	V	M
8	V	M	V	V
9	M	M	V	V

Question 18. Comment interpréter le cas où en ligne 0 du tableau on trouve une variable vivante ? Le programme est-il exécutable ?

Question 19.

Pour un programme comportant n instructions et utilisant p variables numérotées de 0 à $p - 1$, le tableau qui indique l'état de chaque variable pour chaque instruction peut être représenté par une liste de listes `vie` telle que pour la ligne i (avec $0 \leq i < n$) et la variable v (avec $0 \leq v < p$), `vie[i][v]` est `True` si la variable est vivante et `False` si la variable est morte.

Écrire une fonction `determineVie(prog, p)` qui prend en argument un programme et le nombre p de variables à considérer et qui renvoie le tableau `vie` correspondant.

Question 20.

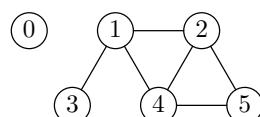
Recopier et compléter la fonction suivante `periodesVie(vie, v)` qui prend en arguments la table `vie` obtenue par la fonction `determineVie` et le numéro d'une variable et qui renvoie la liste des périodes de vie de cette variable. Ainsi, pour l'exemple du programme considéré, et la variable `d`, la fonction `periodesVie` renvoie la liste `[(7, 9), (3, 4), (0, 1)]` (l'ordre n'a pas d'importance).

```
def periodesVie(vie, v):
    n = len(vie)
    periodes = []
    i = n-1
    encours = False
    while i >= 0:
        if vie[i][v]:
            if not encours:
                ...
                ...
            else:
                if ...:
                    ...
                    ...
        i -= 1
    return periodes
```

Partie C – graphe de coexistence

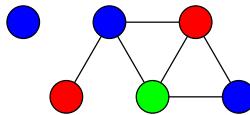
Des techniques analogues à celle présentée dans la partie B permettent de construire un graphe de coexistence des variables occupant des registres. Il s'agit d'un graphe non orienté tel que deux sommets reliés dans ce graphe correspondent à deux variables vivantes au même moment, ce qui oblige à les ranger dans des registres différents.

La figure ci-dessous montre un exemple de graphe de coexistence : les variables numérotées 1, 2, 4 doivent être rangées dans trois registres distincts. Mais les variables numérotées 1 et 5 peuvent être rangées dans le même registre, puisque les sommets correspondants ne sont pas reliés.



Pour déterminer le nombre de registres nécessaire, il suffit de colorier le graphe de sorte que deux sommets adjacents soient de couleurs différentes, en utilisant le moins de couleurs possible. Ce nombre minimal de couleur est le nombre cherché de registres.

Dans l'exemple du graphe ci-dessus, trois couleurs suffisent.



Question 21.

Le problème général de la détermination du nombre minimal de couleurs nécessaire à la coloration d'un graphe est connu pour être un problème *NP*-complet. Que signifie l'expression « ce problème est dans la classe *NP* » ? et que signifie l'adjonction de l'adjectif « complet » ?

Dans la suite, un graphe non orienté possédant n sommets numérotés de 0 à $n - 1$ est représenté par une liste G de n listes : $G[i]$ est la liste (éventuellement vide) des sommets reliés au sommet i . Autrement dit, $G[i]$ est la liste des voisins du sommet i .

L'exemple du graphe précédent est représenté par la liste $G = [[], [2,3,4], [1,4,5], [1], [1,2,5], [2,4]]$.

Une coloration du graphe est représentée par une liste donnant le numéro de la couleur de chaque sommet. Dans l'exemple ci-dessus, si bleu est numéro 0, rouge numéro 1 et vert numéro 2, la coloration proposée est représentée par la liste couleur = [0,0,1,1,2,0].

Question 22.

Écrire une fonction bonnesCouleurs(G , couleur) qui prend en argument un graphe G et une coloration couleur de ses sommets et qui renvoie True si et seulement si la coloration est correcte, c'est-à-dire que deux sommets adjacents n'ont jamais la même couleur.

Question 23.

Il est facile d'écrire un algorithme qui permet de déterminer une coloration correcte d'un graphe, mais sans garantir qu'on utilise le nombre minimal de couleurs.

Par exemple, en partant d'un graphe dont aucun sommet n'est colorié.

1. choisir la première couleur disponible ;
2. tant qu'il existe un sommet non colorié répéter les étapes 3 à 6 ;
3. choisir un sommet s non colorié, et le colorier avec la couleur courante ;
4. répéter l'étape 5 pour tout sommet t non colorié ;
5. si t n'est adjacent à aucun sommet colorié avec la couleur courante, le colorier avec la couleur courante ;
6. choisir une nouvelle couleur ;
7. fin de l'algorithme.

Écrire une fonction coloriage(G) qui prend en argument un graphe et renvoie à l'aide de l'algorithme ci-dessus une coloration possible de ses sommets. On pourra utiliser (sans la recopier) la fonction auxiliaire suivante qui vérifie qu'aucun sommet de la liste voisins n'est de la couleur c .

```
def coloriable(voisins, couleur, c):
    for v in voisins:
        if couleur[v] == c: return False
    return True
```

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation sera **obligatoirement** Python.

Tetris Couleurs

On souhaite programmer un jeu de puzzle dont le principe ressemble au jeu de Tetris : des blocs de couleur apparaissent à l'écran en haut d'une aire de jeu et descendent sous l'effet de la « gravité » jusqu'à reposer sur un bloc déjà présent, ou sur le bas de l'aire de jeu. Les nouveaux blocs de couleur apparaissent par groupe de k blocs superposés en une tour verticale appelée « *barreau* ».

Lors de la descente, le joueur peut déplacer le barreau selon certaines règles. Il peut :

- déplacer le barreau vers la gauche ou vers la droite,
- permutez l'ordre des blocs dans le barreau,
- faire descendre le barreau « rapidement ».

Le but du jeu est de réaliser le plus grand nombre possible d'alignements d'au moins trois blocs de la même couleur. Chaque alignement de trois blocs de la même couleur (horizontalement, verticalement ou en diagonale) donne un point au joueur. Les blocs appartenant à des alignements sont retirés de l'aire de jeu, et les blocs restants sont tassés (ils descendent sous l'effet de la gravité). Les blocs ainsi tassés peuvent à nouveau former des alignements unicolores qui sont à leur tour retirés, et ainsi de suite jusqu'à ce qu'il n'y ait plus aucun alignement unicolore dans l'aire de jeu.

La partie se termine quand l'aire de jeu est trop remplie pour accueillir un nouveau barreau. Le score du joueur est alors la somme des points accumulés lors de la partie.

Complexité. La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de P . Lorsque cette complexité dépend de deux paramètres n et m , on dira que P a une complexité en $\mathcal{O}(\varphi(n, m))$ lorsqu'il existe trois constantes A , n_0 et m_0 telles que la complexité de P est inférieure ou égale à $A \cdot \varphi(n, m)$, pour tout $n \geq n_0$ et $m \geq m_0$.

Lorsqu'il est demandé de donner la complexité d'un programme, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Python. Dans ce sujet, nous adopterons la syntaxe du langage Python. On rappelle qu'en Python, les listes sont des tableaux dynamiques à une dimension. Sur les listes, on dispose des opérations suivantes, qui ont toutes une complexité constante :

- `[]` crée une liste vide (c'est-à-dire ne contenant aucun élément).
- `len(liste)` renvoie la longueur de la liste `liste`.
- `liste.append(x)` ajoute l'élément `x` à la fin de la liste `liste`.
- `liste[i]` renvoie le $(i + 1)$ -ième élément de la liste `liste` s'il existe ou produit une erreur sinon (noter que le premier élément de la liste est `liste[0]`).

L'expression `[k for i in range(n)]` construit une liste de longueur `n` contenant `n` occurrences de `k`.

Important : L'utilisation de toute autre fonction sur les listes telle que `liste.insert(i, x)`, `liste.remove(x)`, `liste.index(x)`, ou encore `liste.sort(x)` est rigoureusement interdite. Ces fonctions devront être réécrites explicitement si nécessaire.

On rappelle que l'on peut récupérer directement les valeurs contenues dans un tuple de la façon suivante : après l'instruction `a, b, c = (1, 2, 4)`, la variable `a` contient la valeur 1, `b` contient la valeur 2 et `c` contient la valeur 4. Cette instruction engendre une erreur si le nombre de variables à gauche est différent de la taille du tuple à droite.

Dans la suite, nous distinguerons *fonctions* et *procédures* : les fonctions renvoient une valeur (un entier, une liste, un couple, etc.) tandis que les procédures ne renvoient aucune valeur.

La partie I contient les principales définitions, les parties II, III, IV et V sont indépendantes.

Nous attacherons la plus grande importance à la lisibilité du code produit par les candidats ; aussi, nous encourageons les candidats à utiliser des commentaires et à introduire des procédures ou des fonctions intermédiaires pour faciliter la compréhension du code.

Partie I. Initialisation et affichage de l'aire de jeu

L'aire de jeu est représentée par une grille de dimensions `largeur`×`hauteur`. Dans l'exemple de la Figure 1(a), la grille est de dimensions 6×12 .

Chaque case de la grille contient une valeur qui représente soit une case vide, soit une couleur. On utilisera les constantes entières suivantes pour représenter l'état des cases de la grille (une constante est une variable globale qui n'est jamais modifiée après création) :

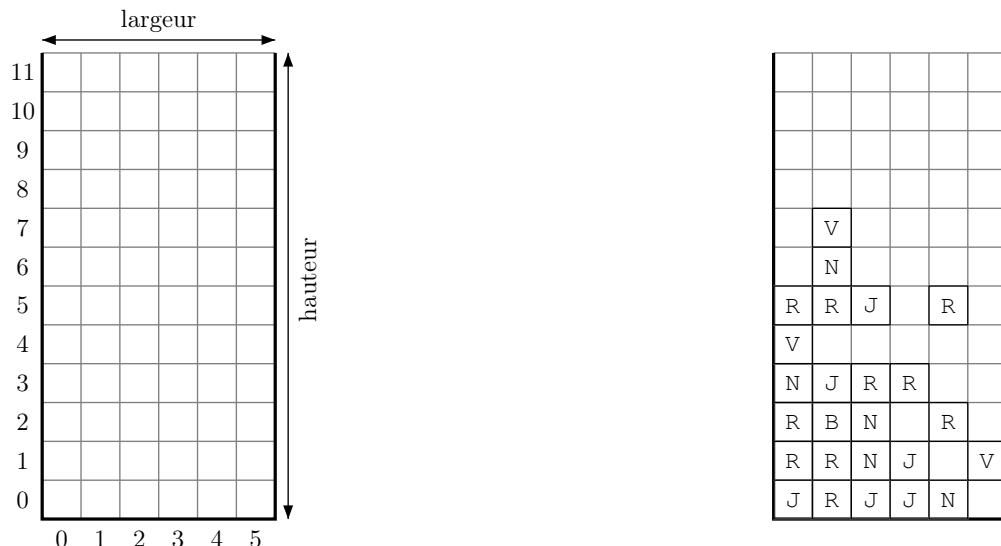
- `VIDE` pour une case vide ;
- `R`, `V`, `B`, `N`, `J` pour les couleurs.

On supposera bien sûr que ces constantes sont deux à deux distinctes et on n'utilisera aucune autre variable globale. La Figure 1(b) montre un exemple d'aire de jeu en cours de partie. Ci-dessous, sa représentation en Python sous la forme d'un tableau de tableaux, ou plus précisément d'un tableau de taille `largeur` (ici 6) contenant dans chaque case une colonne, qui est elle-même un tableau de taille `hauteur` (ici 12) :

```
grille = [ [J,     R,     R,     N,     V,     R,     VIDE, VIDE, VIDE, VIDE, VIDE],
           [R,     R,     B,     J,     VIDE, R,     N,     V,     VIDE, VIDE, VIDE, VIDE],
           [J,     N,     N,     R,     VIDE, J,     VIDE, VIDE, VIDE, VIDE, VIDE, VIDE],
           [J,     J,     VIDE, R,     VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE],
           [N,     VIDE, R,     VIDE, VIDE, R,     VIDE, VIDE, VIDE, VIDE, VIDE, VIDE],
           [VIDE, V,     VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE, VIDE] ]
```

À noter que la valeur de la case supérieure droite de la grille est `grille[5][11]`. On veillera à respecter l'ordre des dimensions afin que la case de coordonnées (i, j) avec $0 \leq i < \text{largeur}$ et $0 \leq j < \text{hauteur}$ (voir Figure 1(a)) corresponde bien à la valeur `grille[i][j]` dans sa représentation en Python.

Question 1. Écrire une fonction `creerGrille(largeur, hauteur)` qui renvoie une grille de dimensions `largeur`×`hauteur` dont toutes les cases sont vides.



(a) Aire de jeu représentée par une grille.

(b) Grille en cours de partie.

FIGURE 1 – L'aire de jeu.

On souhaite pouvoir afficher à l'écran le contenu de l'aire de jeu. Pour cela, on suppose l'existence des procédures suivantes :

- `afficheCouleur(c)` qui prend en argument une constante de couleur $c \in \{R, V, B, N, J\}$ et affiche le caractère correspondant ;
- `afficheBlanc()` qui affiche un espace vide ;
- `nouvelleLigne()` qui déplace le curseur au début de la ligne suivante.

Important : Il est *rigoureusement interdit* d'utiliser toute autre fonction d'affichage, notamment la commande `print`.

Question 2. Écrire une procédure `afficheGrille(grille)` qui affiche à l'écran le contenu de l'aire de jeu, encodé dans le tableau `grille`. On veillera à bien respecter *l'orientation verticale* de la grille : la ligne apparaissant en bas de l'écran doit correspondre aux éléments `grille[i][0]` pour $0 \leq i < \text{largeur}$. Par exemple, la grille de la Figure 1(b) sera affichée comme le montre la capture d'écran ci-contre.

```
V
N
RRJ R
V
NJRR
RBN R
RRNJ V
JRJJN
```

Partie II. Creation et mouvement du barreau

Au cours d'une partie, le joueur voit apparaître à l'écran un *barreau* consistant en une tour de k blocs colorés, où $k \geq 3$. Le barreau apparaît en pointillé sur les figures. La valeur de k et la couleur des blocs du barreau sont choisies aléatoirement. La position d'un barreau est donnée par les coordonnées (x, y) de son bloc inférieur. Dans la suite, on supposera toujours que les coordonnées (x, y) définissent une case dans la grille (c'est-à-dire $0 \leq x < \text{largeur}$ et $0 \leq y < \text{hauteur}$) et que $y + k \leq \text{hauteur}$ (et donc $k \leq \text{hauteur}$).

Un barreau de taille k peut naître au sommet d'une colonne ayant ses k cases supérieures vides (voir Figure 2). Si aucune colonne n'a ses k cases supérieures vides, la partie s'arrête.

Question 3. Écrire une fonction `grilleLibre(grille, k)` qui renvoie `True` si dans au moins une colonne de la grille, les k premières cases (en partant du haut de la grille) sont vides, et renvoie `False` sinon. On ne fait pas d'hypothèse sur le contenu de la grille (en particulier, la grille n'est pas nécessairement tassée). Quelle est la complexité de votre fonction ?

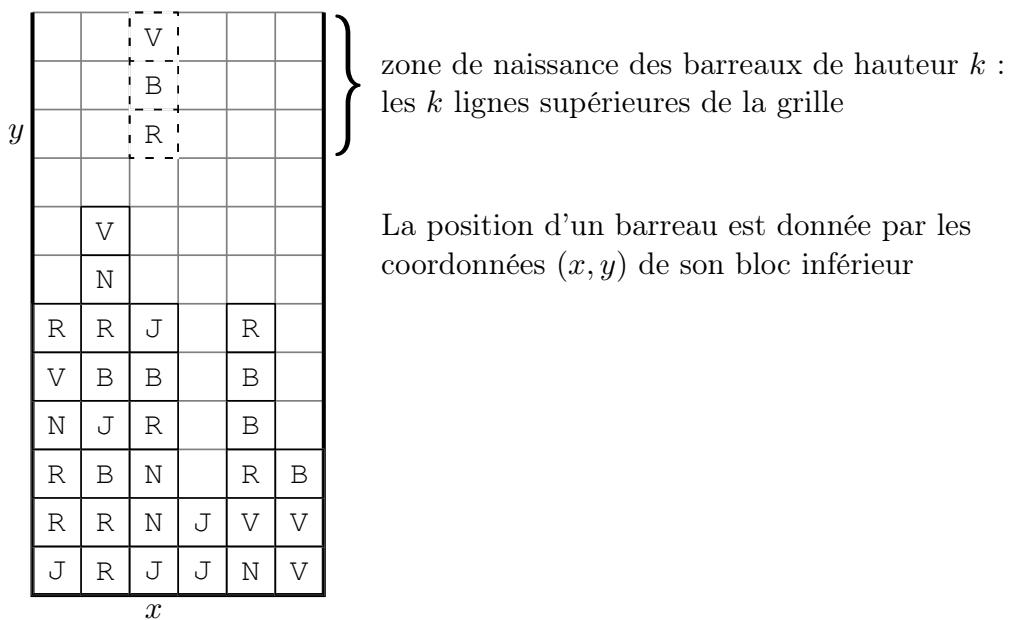


FIGURE 2 – Zone de naissance des barreaux.

En l'absence d'intervention du joueur, le barreau descend d'une case à chaque unité de temps. La descente s'arrête dès que le barreau atteint le bas de la grille, ou rencontre un bloc déjà présent (le passage du temps est illustré à la Figure 3).

Question 4. Écrire une procédure `descente(grille, x, y, k)` qui prend en arguments une grille et les coordonnées (x, y) du bloc inférieur d'un barreau de hauteur k , et modifie la grille en faisant descendre le barreau d'une case. Si le barreau ne peut pas descendre, la grille n'est pas modifiée.

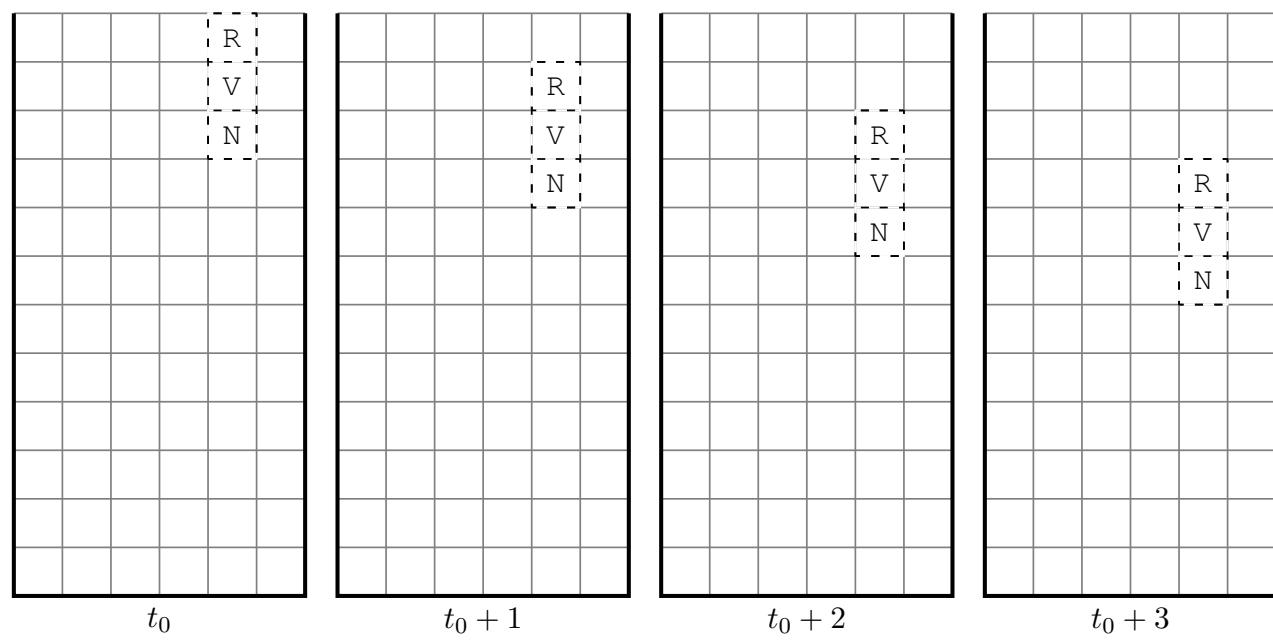


FIGURE 3 – Descente du barreau.

Le joueur peut déplacer le barreau d'une colonne vers la gauche (ou vers la droite) en utilisant les flèches du clavier. Le déplacement du barreau vers la droite n'est possible que si le barreau n'est pas contre le bord droit de la grille et que les k cases se trouvant à sa droite sont toutes vides. Symétriquement, le déplacement du barreau vers la gauche n'est possible que si le barreau n'est pas contre le bord gauche de la grille et que les k cases se trouvant à sa gauche sont toutes vides (Figure 4).

Question 5. Écrire une procédure `deplacerBarreau(grille, x, y, k, direction)` qui prend en argument une grille et les coordonnées (x,y) du bloc inférieur d'un barreau de hauteur k , et un entier $\text{direction} \in \{-1, 1\}$, et qui modifie la grille en déplaçant le barreau d'une case vers la gauche (si $\text{direction} = -1$) ou vers la droite (si $\text{direction} = 1$). Si le déplacement du barreau n'est pas possible, la grille reste inchangée.

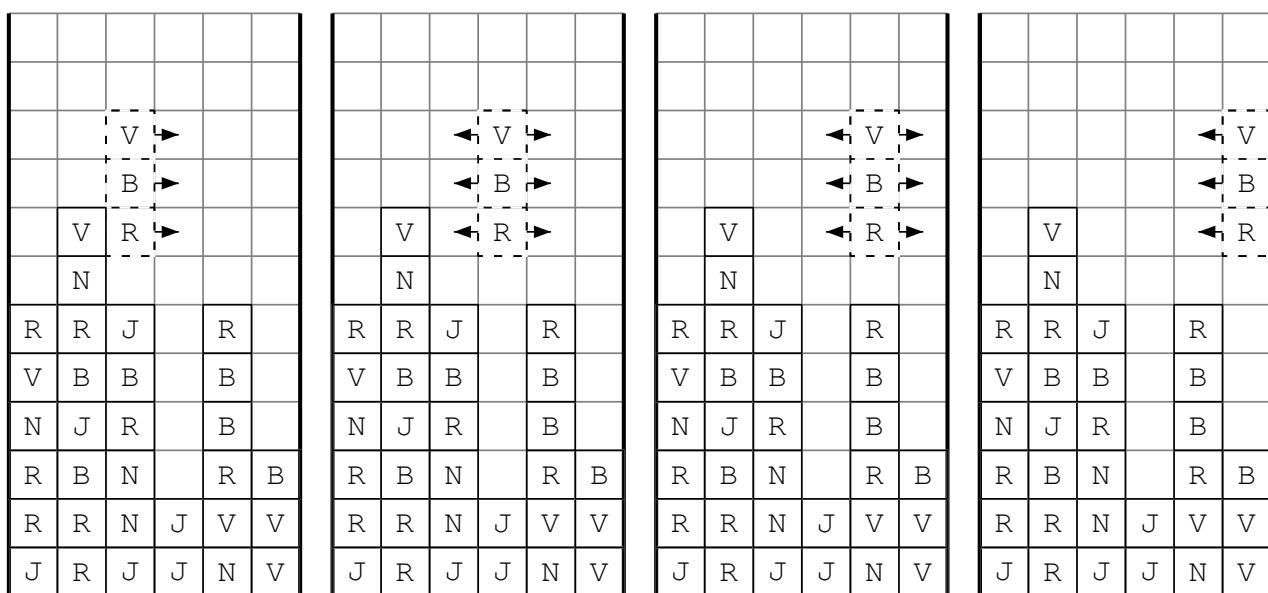


FIGURE 4 – Déplacements possibles du barreau.

Le joueur peut aussi modifier l'ordre des blocs dans le barreau par une permutation circulaire qui fait remonter chaque bloc d'un case, sauf le bloc le plus haut qui redescend à la place du bloc le plus bas (Figure 5).

Question 6. Écrire une procédure `permuterBarreau(grille, x, y, k)` qui modifie la grille en effectuant, comme décrit ci-dessus, une permutation circulaire des couleurs du barreau de hauteur k dont le bloc inférieur est en position (x,y) dans la grille donnée en argument.

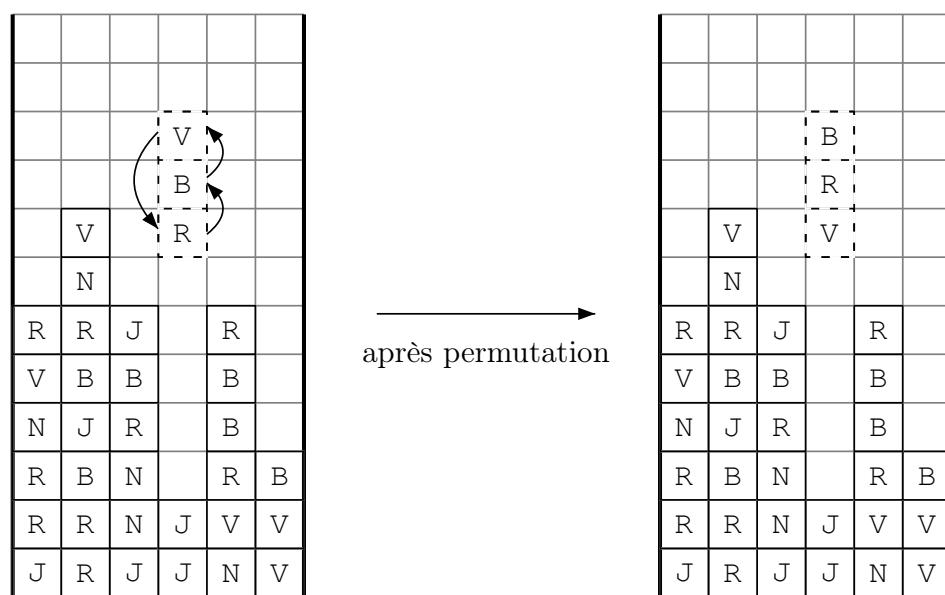


FIGURE 5 – Permutation circulaire du barreau.

Enfin, le joueur peut faire descendre le barreau « rapidement », ce qui signifie que le barreau descend du nombre maximum possible de cases (Figure 6), pour venir reposer au-dessus de la première case non vide de la grille située sous le barreau, ou sur le fond de la grille (si toutes les cases sous le barreau sont vides). Dans l'exemple de la Figure 6, la première case non vide sous le barreau a pour coordonnées (3, 1).

Question 7. Écrire une procédure `descenteRapide(grille, x, y, k)` qui prend en argument une grille et les coordonnées (x, y) du bloc inférieur d'un barreau de hauteur k , et modifie la grille en faisant descendre le barreau « rapidement ». On demande que la fonction ait une complexité en $\mathcal{O}(k + \text{hauteur})$ (et non $\mathcal{O}(k \times \text{hauteur})$).

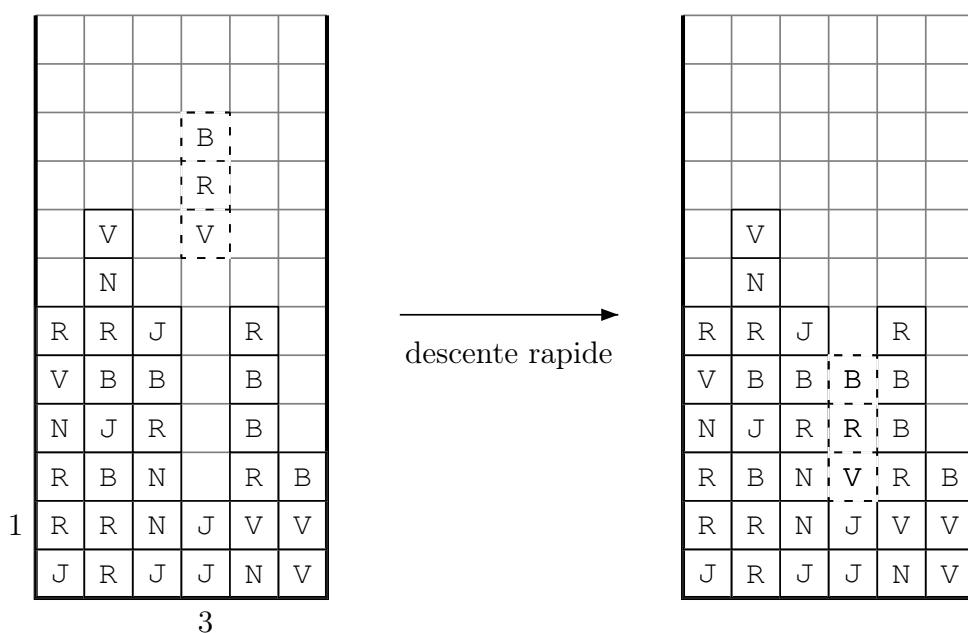


FIGURE 6 – Descente rapide du barreau.

Partie III. Détection des alignements et calcul du score

Lorsqu'un barreau ne peut plus descendre, le joueur gagne des points si des alignements d'au moins trois blocs de la même couleur sont réalisés dans la grille. Les alignements peuvent être réalisés sur une ligne, sur une colonne, ou en diagonale. Notre but est maintenant de détecter les alignements unicolores et de calculer le score du joueur.

Chaque alignement unicolore de longueur $m \geq 3$ donne $m - 2$ points au joueur. Cette règle ne s'applique que si l'alignement de longueur m n'est pas lui-même inclus dans un alignement de longueur plus grande que m , donc on ne prend en considération que les alignements de longueur maximale pour calculer le score. Par exemple l'alignement horizontal de quatre blocs de couleur B dans la Figure 7 donne 2 points, ceux en diagonale de trois blocs de couleur B et V donnent chacun 1 point. Le bloc B de coordonnées (3,4) compte à la fois pour l'alignement horizontal et l'alignement en diagonale. Le joueur marque donc 4 points dans cette configuration.

Tous les blocs appartenant à de tels alignements sont ensuite *simultanément* retirés de la grille et remplacés par des cases vides (deuxième grille de la Figure 7). Les blocs restants sont ensuite tassés, c'est-à-dire qu'ils descendent du maximum possible de cases. Il se peut que de nouveaux alignements se forment après le tassemement de la grille (comme les cinq blocs de couleur R dans la troisième grille de la Figure 7). Ces nouveaux alignements donnent à nouveau des points au joueur (selon le même barème que précédemment) et le même processus d'élimination des alignements et de tassemement de la grille est réalisé. Ce processus se poursuit jusqu'à ce qu'il n'y ait plus d'alignement unicolore de longueur $m \geq 3$ dans la grille. Dans l'exemple, le joueur marque au total 7 points.

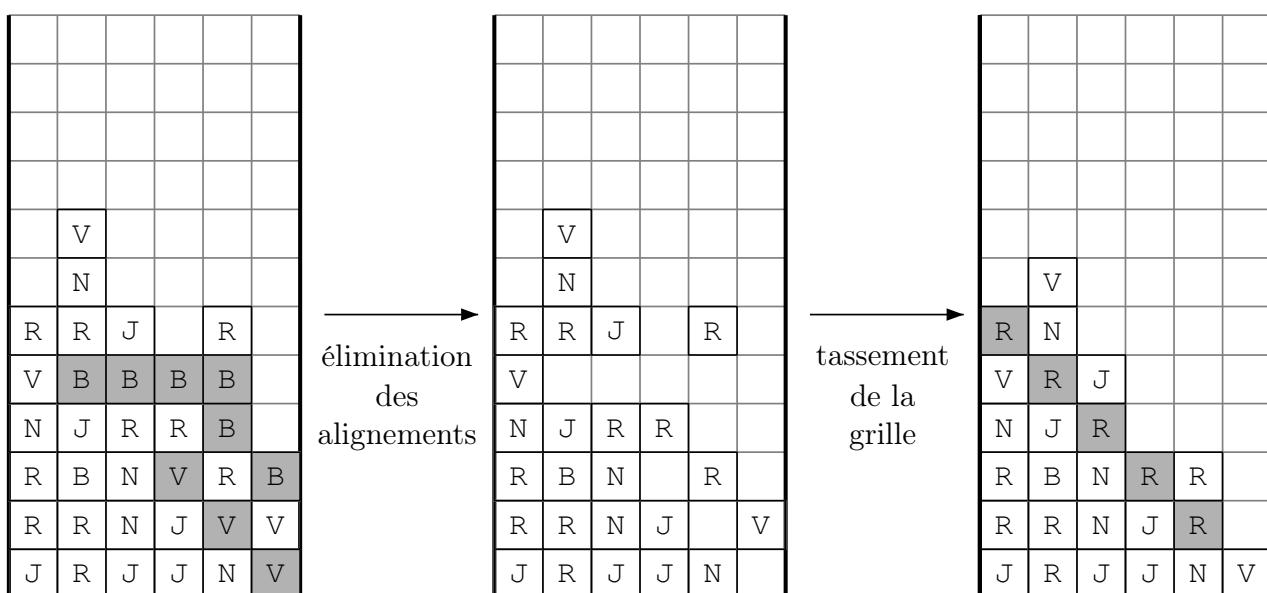


FIGURE 7 – Élimination des alignements unicolores et tassemement de la grille.

Question 8. Calculer le nombre de points marqués par le joueur s'il laisse descendre le barreau dans l'aire de jeu ci-dessous. Justifier votre réponse.

			B		
		R			
	J		N		
	R		N		
			R		
B	B		N	J	
N	R		V	N	
N	R	R	R	V	V
N	B	J	B	V	V
V	N	J	B	V	J

Pour réaliser la fonction qui va détecter et comptabiliser les alignements unicolores de la grille, on va d'abord construire une fonction qui réalise cela sur un tableau simple (à une dimension).

Question 9. Écrire une fonction `detecteAlignement(rangee)` qui prend en argument un tableau `rangee` non vide à une dimension contenant des valeurs dans l'ensemble $\{\text{VIDE}, \text{R}, \text{V}, \text{B}, \text{N}, \text{J}\}$, et qui renvoie un tuple `(marking, score)` de deux éléments :

- `marking` est un tableau de la même taille que `rangee` contenant des Booléens, tel que `marking[i] = True` si et seulement si `rangee[i]` appartient à un alignement unicolore de longueur au moins 3.
- `score` est le nombre de points obtenus par le joueur pour les alignements présents dans `rangee` (selon le barème donné plus haut).

Par exemple, pour le tableau `rangee = [B, R, R, R, R, J, J, J, VIDE, VIDE, VIDE]`, la fonction `detecteAlignement(rangee)` renvoie :

- `marking = [False, True, True, True, True, True, True, True, False, False, False]`
- `score = 3`.

On demande que lors du traitement, la fonction `detecteAlignement(rangee)` n'accède qu'une seule fois à chaque élément du tableau `rangee`.

Indice : Parcourir le tableau et détecter les changements de couleur.

Pour détecter les alignements unicolores de la grille et comptabiliser les points marqués, nous allons explorer toutes les lignes, colonnes et diagonales de la grille. Pendant le traitement, on va conserver une copie de travail de la grille dans laquelle on remplacera les cases appartenant à un alignement unicolore par des cases vides. Dans la question suivante, on suppose que g est cette copie de travail.

Question 10. Écrire une fonction `scoreRangee(grille, g, i, j, dx, dy)` qui prend en argument une grille `grille` et une grille `g` de même dimensions que `grille`, les coordonnées (i, j) d'une case dans la grille et une direction (dx, dy) donnée par $dx, dy \in \{-1, 0, 1\}$. La fonction construit un tableau `rangee` à une dimension contenant les valeur des cases de `grille` dont les coordonnées sont $(i, j), (i+dx, j+dy), (i+2dx, j+2dy)$, etc. puis utilise la fonction `detecteAlignement(rangee)` de la Question 9 pour détecter les alignements unicolores dans le tableau `rangee`, déterminer le nombre de points marqués et mettre à jour la grille `g` pour que les cases appartenant à un alignement unicolore dans `grille` soient vides dans `g`. La grille `grille` ne doit pas être modifiée. La fonction renvoie le nombre de points marqués.

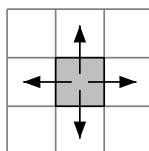
Question 11. Écrire une fonction `effaceAlignement(grille)` qui prend en argument une grille et renvoie un tuple $(g, score)$ de deux éléments :

- `g` est la grille mise à jour où tous les blocs appartenant à un alignement unicolore sont remplacés par des cases vides.
- `score` est le nombre total de points obtenus par le joueur pour les alignements présents dans la grille.

La grille `grille` ne doit pas être modifiée. Donner la complexité de votre fonction.

Question 12. Écrire une procédure `tassementGrille(grille)` qui modifie la grille donnée en argument en effectuant le tassement de ses cases non vides.

Question 13. Écrire une fonction `calculScore(grille)` qui met à jour la grille `grille` après élimination des alignements et tassement, répétés jusqu'à ce que la grille ne contienne plus aucun alignement unicolore de longueur $m \geq 3$, et qui renvoie le nombre total de points marqués par le joueur pour les alignements éliminés de la grille.



(a) Voisins d'une case.

						R
R	R			R	N	
V	R	R	R	R	J	
J	V	R	J	R	J	
J	R	R	J	R	V	
N	J	R	R	R	J	

(b) Une région unicolore maximale.

FIGURE 8 – Régions unicolores (Partie IV).

Partie IV. Variante du jeu : régions unicolores

Dans cette partie, on considère une variante du jeu où le but du joueur est de former des régions unicolore au lieu d'alignements (par exemple un carré de 2×2 cases de la même couleur).

Une région unicolore est un ensemble de cases, toutes de la même couleur, qui est connexe pour la relation d'adjacence suivante : deux cases sont adjacentes si elles partagent un côté (Figure 8(a)).

La région grisée de la Figure 8(b) est une région unicolore maximale : si on y ajoute une case quelconque, elle ne reste plus connexe et unicolore. En particulier, la case supérieure droite ne peut être ajoutée car elle n'est adjacente à aucune case de la région grisée.

Question 14. Écrire une fonction `récursive tailleRegionUnicolore(grille, x, y)` qui renvoie le nombre de cases appartenant à la plus grande région unicolore de la grille contenant la case (x,y) . Justifier la terminaison de votre fonction.

Considérons le code Python aux Figure 9 et Figure 10, dont le but est de réaliser le même travail que la fonction `tailleRegionUnicolore` de la Question 14 sans utiliser la récursivité.

Intuitivement, la fonction `exploreRegion(grille, x, y)` effectue un balayage de la grille, d'abord verticalement à partir de la case (x,y) , puis verticalement à partir de chaque voisine horizontale des cases déjà explorées.

Question 15. Déterminer si la fonction `exploreRegion(grille, x, y)` renvoie le nombre de cases appartenant à la plus grande région unicolore de la grille contenant la case (x,y) .

Si oui, justifier soigneusement que la fonction renvoie toujours une valeur correcte.

Si non, donner un exemple de paramètres `grille, x, y` pour lesquels la valeur renvoyée par la fonction est incorrecte (on pourra dessiner la grille).

```
def xDansGrille(grille, x):
    largeur = len(grille)
    return (x >= 0) and (x < largeur)
```

```
def yDansGrille(grille, y):
    hauteur = len(grille[0])
    return (y >= 0) and (y < hauteur)
```

```
def exploreVertical(grille, x, y, dir):
    hauteur = len(grille[0])
    couleur = grille[x][y]
    v = y + dir

    while yDansGrille(grille, v):
        if grille[x][v] != couleur:
            return v - dir
        v = v + dir

    if dir == 1:
        return hauteur - 1
    else:
        return 0
```

```
def exploreRegion(grille, x, y):

    inf = exploreVertical(grille, x, y, -1)      # explore vers le bas
    sup = exploreVertical(grille, x, y, 1)        # explore vers le haut

    d = exploreHorizontal(grille, x, y, 1)        # explore vers la droite
    g = exploreHorizontal(grille, x, y, -1)        # explore vers la gauche

    score = sup - inf + 1 + d + g
    return score
```

FIGURE 9 – Code Python pour la Question 15.

```

def exploreHorizontal(grille, x, y, dir):
    largeur = len(grille)
    couleur = grille[x][y]

    inf = exploreVertical(grille, x, y, -1)      # explore vers le bas
    sup = exploreVertical(grille, x, y, 1)        # explore vers le haut

    score = 0
    u = x + dir

    while xDansGrille(grille, u) and (inf <= sup):
        v = inf
        infNew = 1           # initialement, infNew > supNew
        supNew = 0

        while (v <= sup):
            if grille[u-dir][v] == couleur and grille[u][v] == couleur:
                infNew = exploreVertical(grille, u, v, -1)      # explore vers le bas
                supNew = exploreVertical(grille, u, v, 1)        # explore vers le haut
                score = score + supNew - infNew + 1
                v = supNew + 2

            while (v <= sup):
                if grille[u-dir][v] == couleur and grille[u][v] == couleur:
                    supNew = exploreVertical(grille, u, v, 1)    # explore vers le haut
                    score = score + supNew - v + 1
                    v = supNew + 1
                v = v + 1
            v = v + 1

        inf = infNew
        sup = supNew
        u = u + dir

    return score

```

FIGURE 10 – Code Python pour la Question 15.

Partie V. Gestion des scores en SQL

On souhaite utiliser une base de données pour stocker les résultats obtenus par une communauté de joueurs. On suppose que l'on dispose d'une base de données comportant les tables JOUEURS(id_j, nom, pays) et PARTIES(id_p, date, durée, score, id_joueur) où :

- `id_j`, de type entier, est la clé primaire de la table JOUEURS,
- `nom` est une chaîne de caractères donnant le nom du joueur,
- `pays` est une chaîne de caractères donnant le pays du joueur,
- `id_p`, de type entier, est la clé primaire de la table PARTIES,
- `date` est la date (AAAAMMJJ) de la partie,
- `durée`, de type entier, est la durée en secondes de la partie,
- `score`, de type entier, est le nombre de points marqués au cours de la partie,
- `id_joueur` est un entier qui identifie le joueur de la partie.

Question 16. Étant donné une chaîne de caractères `cc` contenant le nom d'un joueur, écrire une requête SQL qui renvoie la date, la durée et le score de toutes les parties jouées par le joueur `cc`, listées par ordre chronologique (au choix, croissant ou décroissant).

Question 17. Étant donné un entier `s` (le score que vient de réaliser une joueuse nommée Alice), écrire une requête SQL qui renvoie la position qu'aura le score `s` dans le classement des parties par ordre de score (on suppose que la dernière partie d'Alice n'a pas encore été insérée dans la table des parties). En cas d'`ex aequo` pour le score `s` (une ou plusieurs parties déjà présentes ayant le score `s`), le rang sera le même que s'il n'y avait qu'une seule partie avec le score `s`.

Par exemple, la requête renverra 1 (le score d'Alice est “1^{er}”) si aucun score n'est meilleur que `s`. Autre exemple, si la base de données contient 6 parties dont les scores sont 87, 75, 75, 63, 60, 60, alors le rang de `s = 75` sera 2, le rang de `s = 70` sera 4 et le rang de `s = 60` sera 5.

Question 18. Écrire une requête SQL qui renvoie le record de France de Tetris couleur, c'est-à-dire le meilleur score réalisé par un joueur dont le pays est la France.

Question 19. Étant donné une chaîne de caractères `cc` contenant le nom d'un joueur (ayant déjà joué au moins une partie de Tetris couleur), écrire une requête SQL qui renvoie le rang du joueur `cc`, c'est-à-dire sa position dans le classement des joueurs par ordre de leur meilleur score dans une partie de Tetris couleur (on traitera les `ex aequo` de la même manière qu'à la question 17).

F	I	N
---	---	---

* *
*

A2019 – INFO

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

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

CONCOURS 2019**ÉPREUVE D'INFORMATIQUE COMMUNE**

Durée de l'épreuve : 1 heure 30 minutes

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

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

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

INFORMATIQUE COMMUNE

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

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

AUTOUR DES NOMBRES PREMIERS

Préambule

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

Le sujet étudie différentes questions sur les nombres premiers.

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

Définitions, rappels et notations

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

Partie I. Préliminaires

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

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

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

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

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

```

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

```

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

□ **Q5** – On donne le code suivant :

```

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

```

L'exécution de ce code produit le résultat :

```
x1: 1.0
x2: 0.999999999980838
```

Commenter.

Partie II. Génération de nombres premiers

II.a Approche systématique

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

```

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

Algorithme 1 : Crible d'Ératosthène

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

- **Q6** – Sachant que le langage Python traite les listes de booléens comme une liste d'éléments de 32 bits, quel est (approximativement) la valeur maximale de N pour laquelle $liste_bool$ est stockable dans une mémoire vive de 4 Go ?
- **Q7** – Quel facteur peut-on gagner sur la valeur maximale de N en utilisant une bibliothèque permettant de coder les booléens non pas sur 32 bits mais dans le plus petit espace mémoire possible pour ce type de données (on demande de le préciser) ?
- **Q8** – Écrire la fonction `erato_iter(N)` qui implémente l'algorithme 1 pour un paramètre N qui est un entier supérieur ou égal à 1.
- **Q9** – Quelle est la complexité algorithmique du crible d'Ératosthène en fonction de N ? On admettra que :

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

La réponse devra être justifiée.

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

II.b Génération rapide de nombres premiers

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

réussit un de ces tests alors la probabilité qu'il ne soit pas premier est prouvée être inférieure à un seuil calculable.

En suivant cette idée, une nouvelle approche est la suivante :

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

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

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

où M est le produit de deux nombres premiers quelconques et x_0 une valeur initiale nommée « graine » choisie aléatoirement. On utilise ici l'horloge de l'ordinateur comme source pour x_0 .

Puis, pour chaque x_i , s'il est impair, on additionne 2^i à A .

□ Q11 – On répète (2) pour i parcourant $\llbracket 1, N - 1 \rrbracket$, quelle sera la valeur de A si x_i est impair à chaque itération ?

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

```

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

```

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

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

□ Q14 – On souhaite caractériser le taux d'erreurs de `premier_rapide`.

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

Partie III. Compter les nombres premiers

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

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

□ **Q15** – Écrire une fonction `Pi(N)` qui calcule la valeur exacte de $\pi(n)$ pour tout entier n de $\llbracket 1, N \rrbracket$. Les nombres premiers sont déduits de la liste `liste_bool` renvoyée par la fonction `erato_iter` de la question 8. On demande que `Pi(N)` renvoie son résultat sous la forme d'une liste de $[n, \pi(n)]$. Par exemple `Pi(4)` renvoie la liste $[[1, 0], [2, 1], [3, 2], [4, 2]]$.

Un seul appel à `erato_iter` est autorisé et on exige une fonction dont la complexité, en dehors de cet appel, est linéaire en fonction de N . Le paramètre N est un entier supérieur à 1.

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

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

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

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

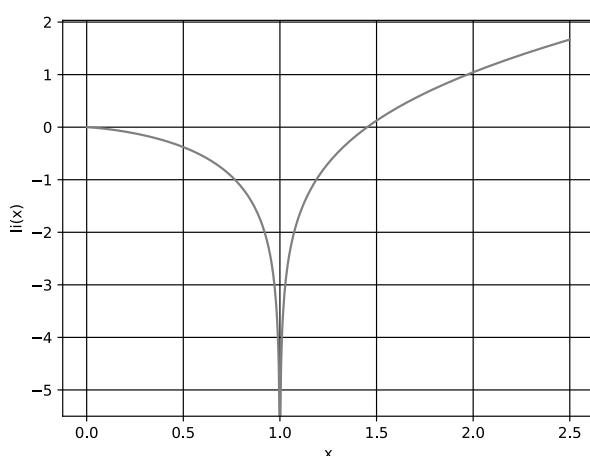


FIGURE 1 – Allure de li sur $[0, 2.5]$

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

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

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

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

$$\lim_{x \rightarrow \infty} \frac{\pi(\lfloor x \rfloor)}{\text{li}(x)} = 1 \quad (5)$$

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

Estimation de li par quadrature numérique

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

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

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

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

On considère que le réel $b - a$ est un multiple du réel *pas*.

Les paramètres `a`, `b` et `pas` sont des flottants.

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

Analyse des résultats de `li_d`

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

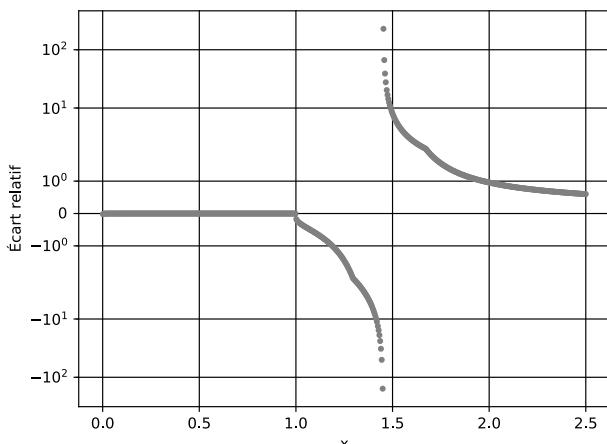


FIGURE 2 – Écart relatif entre `li_d` ($\text{pas} = 10^{-4}$) et l'implémentation de référence `ref_li`.

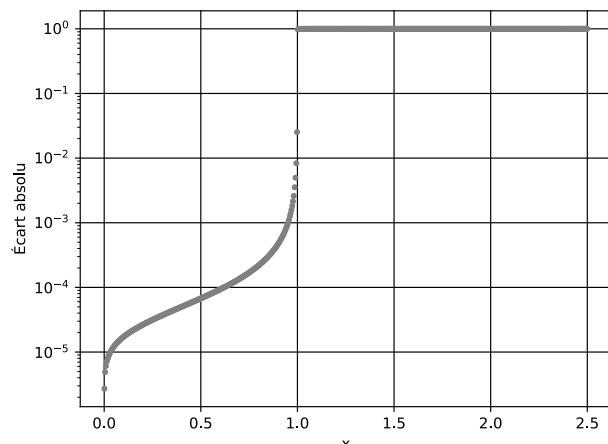


FIGURE 3 – Écart absolu entre `li_d` ($\text{pas} = 10^{-4}$) et l'implémentation de référence `ref_li`.

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

□ **Q22** – On constate un écart absolu important entre `li_d` et `ref_li` au delà de $x = 1$, illustré figure 3. Expliquer succinctement d'où vient ce phénomène. On ne demande pas une démonstration mathématique rigoureuse.

Pour répondre à cette question on pourra remarquer qu'au premier ordre $\frac{1}{\ln(1+\varepsilon)} \simeq -\frac{1}{\ln(1-\varepsilon)}$ quand $\varepsilon \rightarrow 0$ et s'interroger sur la valeur que devrait avoir l'intégrale impropre de $\frac{1}{\ln(x)}$ sur un intervalle $[1 - \varepsilon, 1 + \varepsilon]$ avec $\varepsilon \ll 1$. Une analyse géométrique de la figure 4 peut aussi s'avérer utile.

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

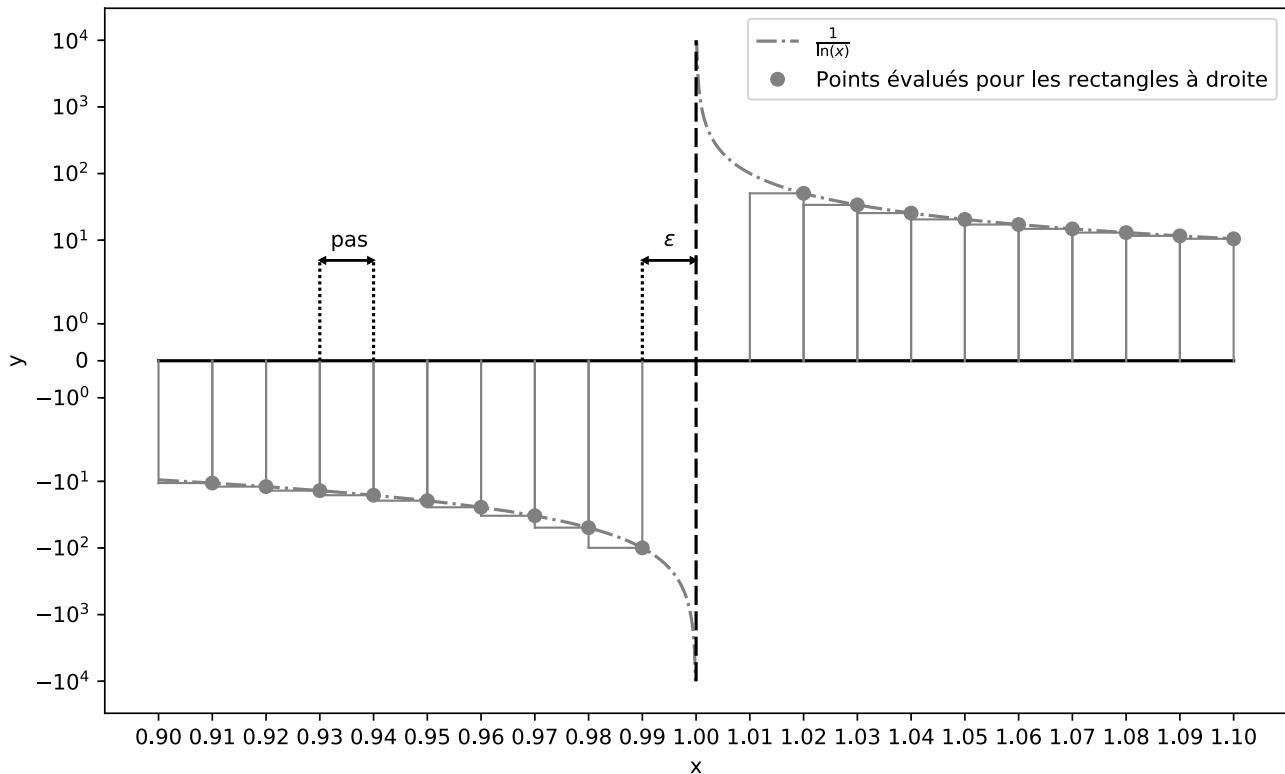


FIGURE 4 – Rectangles utilisés par la méthode des rectangles à droite au voisinage de 1 afin de calculer la valeur principale de Cauchy introduite dans l'équation (4). Les paramètres sont $pas = \varepsilon = 10^{-2}$

Estimation de li via Ei

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

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

Pour le cas $x > 0$ on utilise la valeur principale de Cauchy telle que vue pour li.

Le lien entre li et Ei est :

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

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

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

Avec $\gamma \simeq 0.577215664901$ la constante d'Euler-Mascheroni.

Comme l'évaluation de la somme jusqu'à l'infini est impossible on utilise en pratique la somme suivante :

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

Le choix de n se fait en comparant Ei_{n-1} à Ei_n jusqu'à ce qu'ils soient considérés comme suffisamment proches.

L'évaluation via un ordinateur de ce développement est numériquement stable jusqu'à $x = 40$. Au delà les résultats sont entachés d'erreurs de calcul et d'autres méthodes doivent être utilisées.

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

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

Prendre `MAXIT = 100` se révèle largement suffisant à l'usage.

On demande à ce que la complexité dans le pire des cas soit $O(\text{MAXIT})$. Le paramètre `x` est un flottant quelconque.

Partie IV. Analyse de performance de code

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

La première table est `ordinateurs` et permet de stocker des informations sur les ordinateurs utilisés pour les tests. Ses attributs sont :

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

Exemple du contenu de cette table :

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

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

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

Exemple du contenu de cette table :

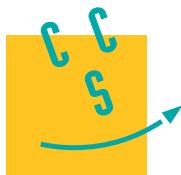
id	nom	algorithme	teste_sur	temps_exec
1	li	rectangles	nyarlathotep165	2638
2	li	rectangles	shubniggurath28	736
3	li	trapezes	nyarlathotep165	4842
...				
2154	Ei	puiseux	nyarlathotep145	2766
2155	aleatoire	BBS	azathoth145	524

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

□ **Q26** – Écrire des requêtes SQL permettant de :

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

Fin de l'épreuve.



CONCOURS CENTRALE-SUPÉLEC

Informatique

MP, PC, PSI, TSI**2019**

3 heures

Calculatrice autorisée

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 oeuvre 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 chaîne 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 » appliquée 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 indégradable, 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`

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, \dots, x_n est la moyenne des carrés des écarts à la moyenne, c'est-à-dire

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

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 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.

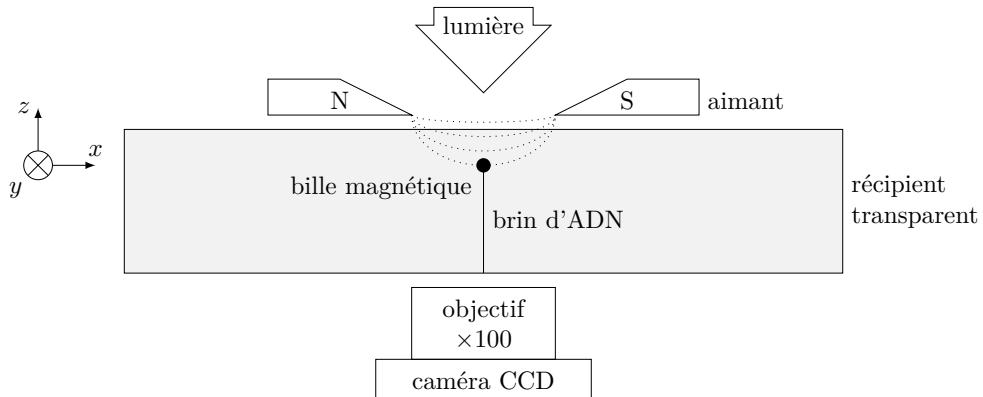


Figure 1 Schéma du dispositif expérimental (échelle non respectée)

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).

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.

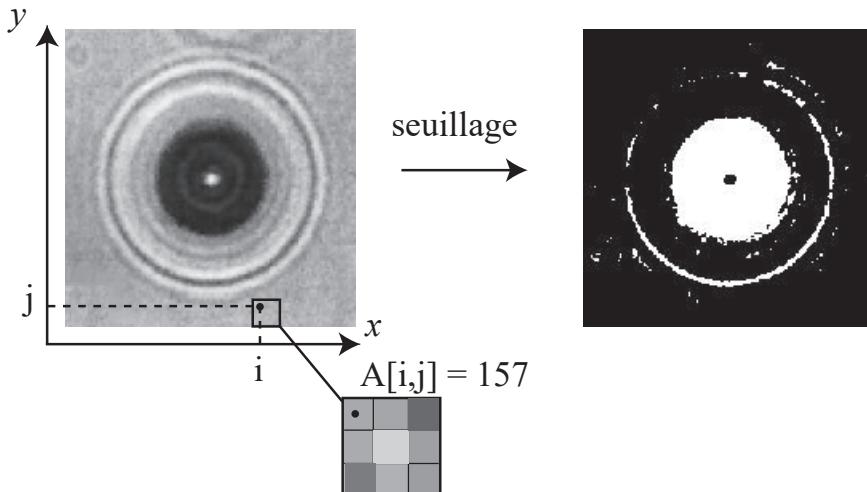


Figure 2 Figure de diffraction d'une bille et opération de seuillage

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).

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 découplant 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.

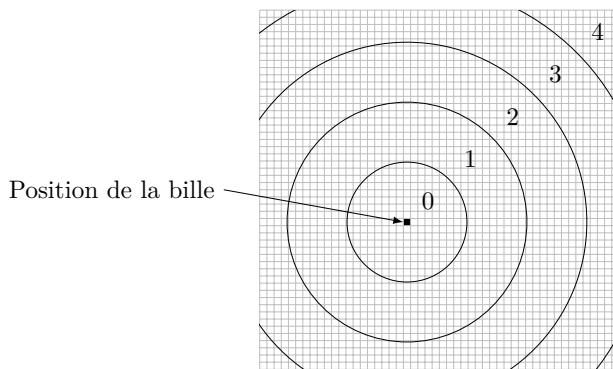


Figure 3 Exemple de découpage d'une image en cinq anneaux concentriques

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 découplant 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 :

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

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.

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

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

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

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'élargissement z_i et les valeurs de la fonction **force** est défini par

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

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 $g_x = \frac{\partial G}{\partial x}$ et $g_y = \frac{\partial G}{\partial y}$ les composantes du gradient de la fonction G . On rappelle que

$$\begin{aligned} g_x(x, y) &= g_x(x_0, y_0) + \frac{\partial g_x}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial g_x}{\partial y}(x_0, y_0)(y - y_0) + o(x - x_0, y - y_0) \\ g_y(x, y) &= g_y(x_0, y_0) + \frac{\partial g_y}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial g_y}{\partial y}(x_0, y_0)(y - y_0) + o(x - x_0, y - y_0) \end{aligned}$$

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 rechercher 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:
```

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 chaîne librement jointe

La molécule d'ADN peut également être représentée par le modèle de la « chaîne 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 recherchés.

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 $\theta_i \in [-\pi, \pi[$ ($i \in \llbracket 0, n-1 \rrbracket$) 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).

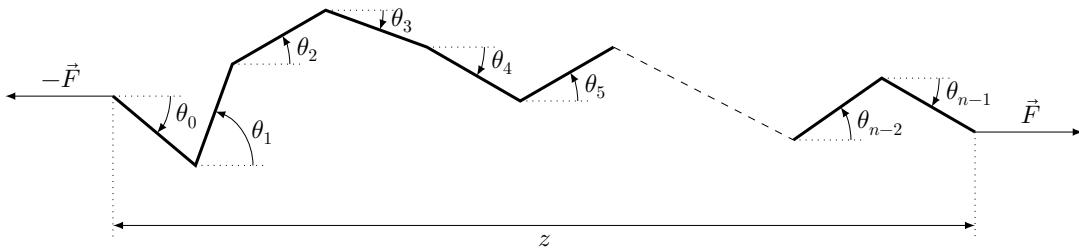


Figure 5 Représentation d'un brin d'ADN sous forme de chaîne librement jointe

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

$$E = -zF \quad (\text{IV.1})$$

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 chaîne dans la conformation θ pour une longueur de segment 1.

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 créée 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 $E_2 \geq E_1$, la nouvelle conformation est conservée, avec la probabilité

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

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.

Opérations et fonctions Python disponibles

Fonctions

- `range(n)` renvoie la séquence des `n` premiers entiers ($0 \rightarrow n - 1$)
`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 $O(\text{len}(u))$
`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 $O(\text{len}(u))$
- `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`

- `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 • • •

SESSION 2019

PSIIN07



ÉPREUVE SPÉCIFIQUE - FILIÈRE PSI

INFORMATIQUE

Vendredi 3 mai : 8 h - 11 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet est composé de trois parties indépendantes.

L'épreuve est à traiter en langage **Python**, sauf les questions sur les bases de données qui seront traitées en langage **SQL**. La syntaxe de Python est rappelée en **Annexe**, page 12.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).

Il est demandé au candidat de bien vouloir rédiger ses réponses **en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions**. Bien que largement indépendante, la **partie III** fait appel aux données et à la définition de fonctions définies dans la **partie II**.

La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

Sujet : page 1 à page 11

Annexe : page 12

Intelligence Artificielle - Application en médecine

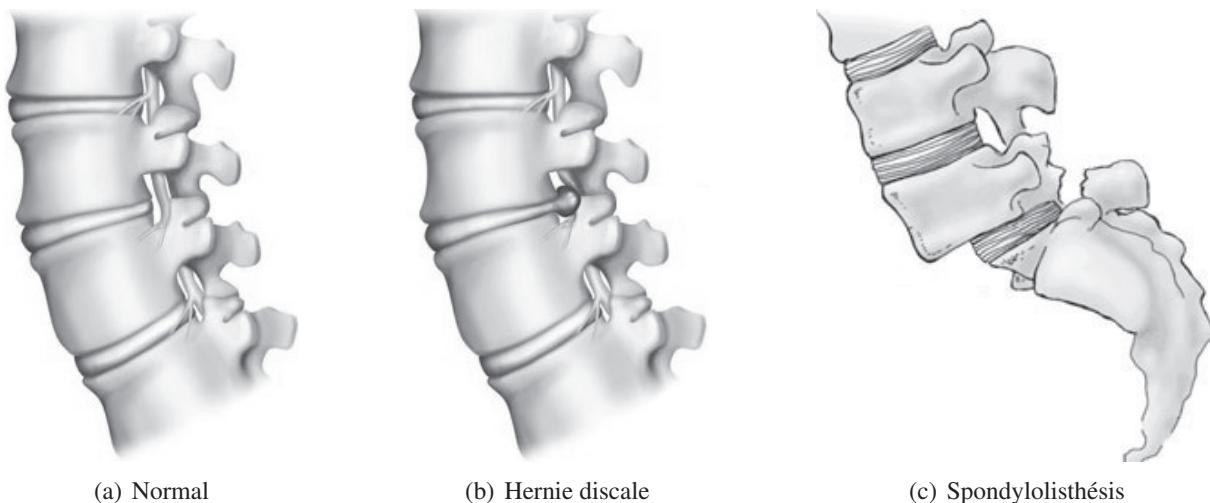
Partie I - Présentation

L'Intelligence Artificielle est de plus en plus utilisée dans de nombreux domaines divers et variés. Les techniques d'apprentissage machine, qui permettent à un logiciel d'apprendre automatiquement à partir de données au lieu d'être explicitement programmé, fournissent des résultats impressionnantes. Les applications sont nombreuses dans la reconnaissance d'image, la compréhension de la parole ou de textes, dans l'assistance à la décision, dans la classification de données, dans la robotique...

L'Intelligence Artificielle progresse également dans le domaine de la santé. Le logiciel Watson développé par IBM peut analyser les données d'un patient : ses symptômes, ses consultations médicales, ses antécédents familiaux, ses données comportementales, ses résultats d'examen, etc. Il établit alors une prévision de diagnostic le plus vraisemblable et propose des options de traitement en s'appuyant sur une base de données établie sur un grand nombre de patients.

Objectif

L'objectif du travail proposé est de découvrir quelques techniques d'Intelligence Artificielle en s'appuyant sur un problème médical. À partir d'une base de données comportant des propriétés sur le bassin et le rachis lombaire (figures 1), on cherche à déterminer si un patient peut être considéré comme « normal » ou bien si ces données impliquent un développement anormal de type hernie discale (saillie d'un disque intervertébral) ou spondylolisthésis (glissement du corps vertébral par rapport à la vertèbre sous-jacente).



Figures 1 – Différentes configurations des vertèbres

Le sujet abordera les points suivants :

- analyse et représentation des données,
- prédiction à l'aide de la méthode KNN,
- apprentissage et prédiction à l'aide de la méthode dite « Naïve Bayes ».

Dans tout le sujet, il sera supposé que les modules python `numpy`, `matplotlib.pyplot` sont déjà importés dans le programme.

Partie II - Analyse des données

La base de données médicale contient des informations administratives sur les patients et des informations médicales. Pour simplifier le problème, on considère deux tables : PATIENT et MEDICAL.

La table PATIENT contient les attributs suivants :

- id : identifiant d'un individu (entier), clé primaire ;
- nom : nom du patient (chaîne de caractères) ;
- prenom : prénom du patient (chaîne de caractères) ;
- adresse : adresse du patient (chaîne de caractères) ;
- email : (chaîne de caractères) ;
- naissance : année de naissance (entier).

PATIENT	MEDICAL
id nom prenom adresse email naissance	id data1 data2 ... idpatient etat

La table MEDICAL contient les attributs suivants :

- id : identifiant d'un ensemble de propriétés médicales (entier), clé primaire ;
- data1 : donnée (flottant) ;
- data2 : donnée (flottant) ;
- ... ;
- idpatient : identifiant du patient représenté par l'attribut id de la table PATIENT (entier) ;
- etat : description de l'état du patient (chaîne de caractères).

Les attributs data1, data2... sont des données relatives à l'analyse médicale souhaitée (dans notre cas des données biomécaniques). L'attribut « etat » permet d'affecter un label à un ensemble de données médicales : « normal », « hernie discale », « spondylolisthésis ».

- Q1.** Écrire une requête SQL permettant d'extraire les identifiants des patients ayant une « hernie discale ».
- Q2.** Écrire une requête SQL permettant d'extraire les noms et prénoms des patients atteints de « spondylolisthésis ».
- Q3.** Écrire une requête SQL permettant d'extraire chaque état et le nombre de patients pour chaque état.

Une telle base de données permet donc de faire de nombreuses recherches intéressantes pour essayer de trouver des liens entre les données des patients et une maladie. Cependant, compte-tenu du nombre d'informations disponibles, il est nécessaire de mettre en place des outils pour aider à la classification de nouveaux patients. C'est l'objet des algorithmes d'Intelligence Artificielle développés dans la suite. Pour l'étude qui va suivre, on extrait de la base de données les attributs biomécaniques de chaque patient que l'on stocke dans un tableau de réels (codés sur 32 bits) à N lignes (nombre de patients) et n colonnes (nombre d'attributs égal à 6 dans notre exemple). On nomme `data` ce tableau. L'état de santé est stocké dans un vecteur de taille N contenant des valeurs entières (codées sur 8 bits) correspondant aux différents états pris en compte (0 : normal, 1 : hernie discale, 2 : spondylolisthésis). On le nomme : `etat`.

Les variables `data` et `etat` sont stockées dans des objets de type `array` de la bibliothèque Numpy. Des rappels quant à l'utilisation de ce module Python sont donnés dans l'**Annexe**, page 12.

- Q4.** Citer un intérêt d'utiliser la bibliothèque de calcul numérique Numpy quand les tableaux sont de grande taille.
- Q5.** Déterminer la quantité de mémoire totale en Mo (1 Mo = 1 000 000 octets) nécessaire pour stocker le tableau et le vecteur des données si $N = 100\,000$. On supposera que les données sont représentées en suivant la norme usuelle IEEE 754.

Les 6 attributs considérés dans notre exemple (les n colonnes du tableau) sont définis ci-après :

- angle d'incidence du bassin en ° ;
- angle d'orientation du bassin en ° ;
- angle de lordose lombaire en ° ;
- pente du sacrum en ° ;
- rayon du bassin en mm ;
- distance algébrique de glissement de spondylolisthésis en mm.

Les labels de ces attributs sont stockés dans une liste nommée :

```
label_attributs = ['incidence_bassin', 'orientation_bassin', 'angle_lordose',
'pente_sacrum', 'rayon_bassin', 'glissement_spon'].
```

Le tableau suivant montre les premières valeurs du tableau data.

incidence_bassin	orientation_bassin	angle_lordose	pente_sacrum	rayon_bassin	glissement_spon
63,03	22,55	39,61	40,48	98,67	- 0,25
39,06	10,06	25,02	29,0	114,41	4,56
68,83	22,22	50,09	46,61	105,99	- 3,53
69,3	24,65	44,31	44,64	101,87	11,21
49,71	9,65	28,32	40,06	108,17	7,92
40,25	13,92	25,12	26,33	130,33	2,23
48,26	16,42	36,33	31,84	94,88	28,34

Tableau 1 – Données (partielles) des patients à diagnostiquer

Avant de traiter les données pour réaliser une prédiction (ou diagnostic), il peut être intéressant de les visualiser en traçant un attribut en fonction d'un autre attribut et en utilisant des motifs différents selon les valeurs du vecteur etat.

On obtient, à partir des données exploitées, les courbes de la **figure 2** (zoom sur la **figure 3**).

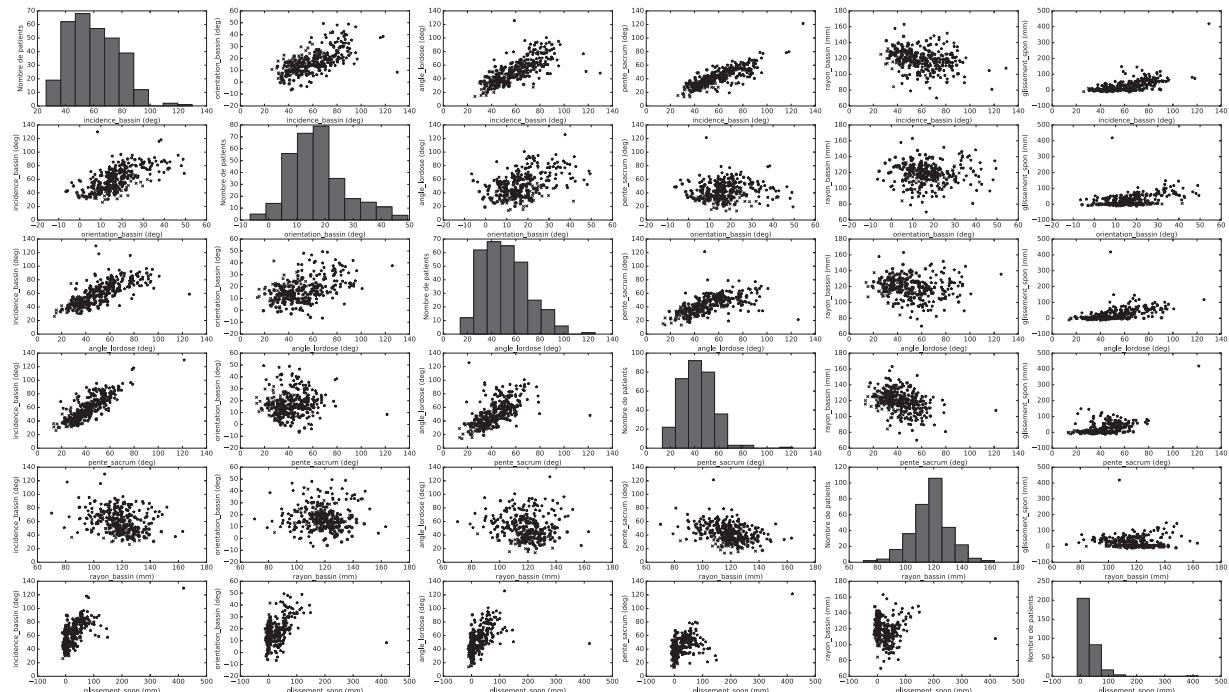


Figure 2 – Répartition des données

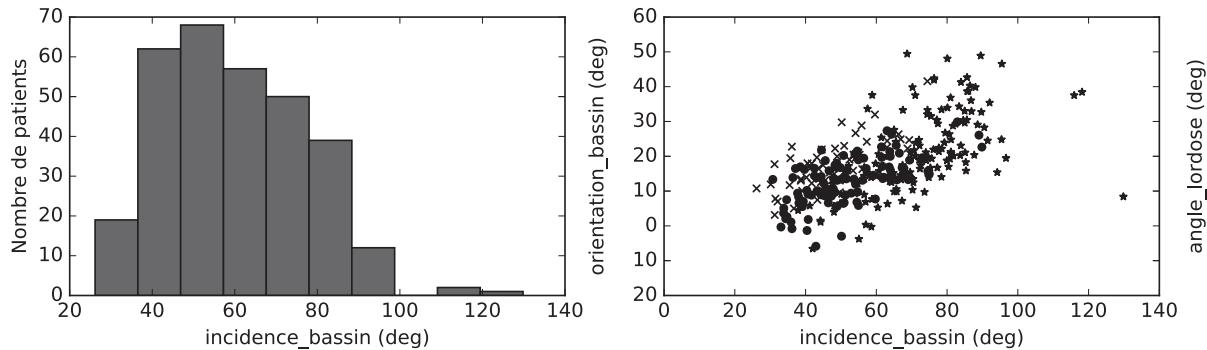


Figure 3 – Zoom sur les figures de coordonnées (1,1) et (1,2) de la matrice de répartition des données (**figure 2**)

Cette figure est une matrice dont le terme (i, i) de la diagonale est l'histogramme des fréquences de l'attribut i et les termes extra-diagonaux (i, j) représentent l'attribut i en fonction de l'attribut j pour chaque ligne du tableau `data`.

Sur les figures hors diagonale, on représente chaque donnée d'attribut i en fonction de l'attribut j à l'aide d'un symbole dépendant de l'état du patient (un rond 'o' si l'état du patient est normal, une croix 'x' si l'état est "hernie discale" et une étoile '*' si l'état est "spondylolisthésis").

Pour réaliser cette figure, il faut séparer les données en fonction de l'état du patient afin d'affecter un symbole par état. Ceci revient à classer les patients en plusieurs groupes.

- Q6.** Écrire une fonction `separationParGroupe(data, etat)` qui sépare le tableau `data` en 3 sous-tableaux en fonction des valeurs du vecteur `etat` correspondantes (on rappelle que celui-ci contient les valeurs 0, 1 et 2 uniquement). La fonction doit renvoyer une liste de taille 3 de sous-tableaux. Les sous-tableaux ne seront pas nécessairement représentés par un type 'array'; 'liste de listes', 'liste d'array' conviennent également.

Pour définir la figure, les instructions suivantes sont exécutées :

```

1 fig = plt.figure()
2 mark = [ 'o', 'x', '*' ]
3 label_attributs = [ 'incidence_bassin (deg)', 'orientation_bassin (deg)', 
4                     'angle_lordose (deg)', 'pente_sacrum (deg)', 
5                     'rayon_bassin (mm)', 'glissement_spon (mm)' ]
6 groupes = separationParGroupe(data , etat)
7 for i in range (len(groupes)):
8     groupes[i] = array(groupes[i])
9
10 n=len(data[0])
11 for i in range(n):
12     for j in range(n):
13         ax1 = plt.subplot(ARGS1)
14         plt.ylabel(label_attributs[j]) #mettre un label à l'axe y
15         if TEST :
16             for k in range(len(groupes)):
17                 plt.xlabel(label_attributs[i]) #mettre un label à l'axe x
18                 ax1.scatter(ARGS2)
19         else :
20             plt.xlabel("Nombre de patients") #mettre un label à l'axe x
21             ax1.hist(ARGS3)
22 plt.show()

```

Les instructions précédentes utilisent la fonction `separationParGroupe` définie à la question **Q6**, puis les éléments de la liste `groupes` sont convertis en `array` afin de pouvoir extraire les colonnes plus facilement.

La documentation du module `matplotlib(plt)` renseigne sur les arguments des fonctions `subplot`, `scatter` et `hist`.

```
| ax1 = plt.subplot(a,b,k)
```

Cette instruction permet de sélectionner parmi un tableau de figures de taille `a` (nombre de lignes), `b` (nombre de colonnes), la k^{e} en numérotant les sous-figures de 1 à $m \times n$ en partant du haut gauche vers le bas droite (en allant de gauche à droite et de haut en bas).

```
| ax1.scatter(datax , datay , marker=mark[k])
```

Cette commande permet de tracer sur la sous-figure `ax1` un nuage de points d'abscisses un vecteur `datax` et d'ordonnées un vecteur `datay` avec un symbole à choisir parmi ceux de la liste `mark`.

```
| ax1.hist(datax)
```

Cette commande permet de tracer un histogramme des données `datax` sur la sous-figure `ax1`.

Q7. Définir les arguments `ARGS1`, `ARGS2`, `ARGS3` ainsi que la condition `TEST` définis dans le script précédent permettant d'obtenir la **figure 2**.

Q8. Préciser l'utilité des diagrammes de la diagonale ainsi que celle des diagrammes hors diagonale.

Partie III - Apprentissage et prédiction

III.1 - Méthode KNN

La méthode KNN est une méthode d'apprentissage dite supervisée ; les données sont déjà classées par groupes clairement identifiés et on cherche dans quels groupes appartiennent de nouvelles données.

Le principe de la méthode est simple. Après avoir calculé la distance euclidienne entre toutes les données connues des patients et les données d'un nouveau patient à classer, on extrait les K données connues les plus proches. L'appartenance du nouveau patient à un groupe est obtenue en cherchant le groupe majoritaire, c'est-à-dire, le groupe qui apparaît être le plus représentatif parmi les K données connues.

On note X_i , $i \in \llbracket 0, n - 1 \rrbracket$, le vecteur colonne i du tableau de données `data`, c'est-à-dire les valeurs prises par l'attribut i des données des patients déjà classées. On cherche à déterminer à quel groupe appartient un nouveau patient dont les attributs sont représentés par un n -uplet z de valeurs notées $(z_0, z_1, \dots, z_{n-1})$.

Préparation des données

Avant de calculer la distance euclidienne, il est préférable de normaliser les attributs pour éviter qu'un attribut ait plus de poids par rapport à un autre. On choisit de ramener toutes les valeurs des attributs entre 0 et 1.

Une technique de normalisation consiste à rechercher pour chaque attribut X le minimum ($\min(X)$) qui est placé à 0 et le maximum ($\max(X)$) qui est placé à 1. On note alors X_{norm} un des vecteurs colonne X après normalisation (toutes les valeurs de X_{norm} sont donc comprises entre 0 et 1).

Q9. Proposer une expression de x_{norm_j} un élément du vecteur X_{norm} en fonction de l'élément x_j du vecteur X correspondant et de $\min(X)$ et $\max(X)$.

Q10. Écrire une fonction $\text{min_max}(X)$ qui retourne les valeurs du minimum et du maximum d'un vecteur X passé en argument. La fonction devra être de complexité linéaire.

Q11. Écrire une fonction $\text{distance}(z, \text{data})$ qui parcourt les N lignes du tableau data et calcule les distances euclidiennes entre le n -uplet z et chaque n -uplet x du tableau de données connues (x représente une ligne du tableau). La fonction doit renvoyer une liste de taille N contenant les distances entre chaque n -uplet x et le n -uplet z .

Détermination des K plus proches voisins

Pour déterminer les K plus proches voisins avec K un entier choisi arbitrairement, il suffit d'utiliser un algorithme de tri efficace. La liste T à trier est une liste de listes à 2 éléments contenant :

- la distance entre le n -uplet à classer et un n -uplet connu (on trie par ordre croissant sur ces valeurs);
- la valeur de l'état correspondant au n -uplet connu.

On retient l'algorithme suivant :

```

1 | def tri(T):
2 |     if len(T) <= 1:
3 |         return T
4 |     else:
5 |         m = len(T)//2
6 |         tmp1 = []
7 |         for x in range(m):
8 |             tmp1.append(T[x])
9 |         tmp2 = []
10 |        for x in range(m, len(T)):
11 |            tmp2.append(T[x])
12 |        return fct(tri(tmp1), tri(tmp2))
13 |
14 | def fct(T1, T2):
15 |     if T1 == []:
16 |         ..... #ligne 1 à compléter
17 |     if T2 == []:
18 |         ..... #ligne 2 à compléter
19 |     if T1[0][0] < T2[0][0]:
20 |         return [T1[0]]+fct(T1[1:], T2)
21 |     else:
22 |         ..... #ligne 3 à compléter

```

Q12. Donner le nom de ce tri ainsi que son intérêt par rapport à un tri par insertion. Préciser un inconvénient du programme proposé par rapport à ce tri.

Q13. Préciser les lignes 1 à 3 de la fonction fct pour que l'algorithme de tri soit fonctionnel.

L'algorithme de la méthode KNN est décrit par la fonction python suivante :

```

1 | def KNN( data , etat ,z ,K, nb ) :
2 |     #partie 1
3 |     T = []
4 |     dist = distance(z , data )
5 |     for i in range(len(dist)):
6 |         T.append([ dist[i] ,i ])
7 |     tri(T)
8 |
9 |     #partie 2
10 |    select = [0]*nb
11 |    for i in range(K):
12 |        select[ etat[T[i][1]] ]+=1
13 |
14 |    #partie 3
15 |    ind = 0
16 |    res = select[0]
17 |    for k in range(1,nb):
18 |        if select[k] > res:
19 |            res = select[k]
20 |            ind = k
21 |    return ind

```

K représente le nombre de voisins proches retenus, nb correspond au nombre d'état (3 dans notre exemple) et z correspond aux attributs du patient à classer.

Q14. Expliquer ce que font globalement les parties 1 (lignes 3 à 7), 2 (lignes 10 à 12) et 3 (lignes 15 à 21) de l'algorithme. Préciser ce que représentent les variables locales T , $dist$, $select$, ind .

Validation de l'algorithme

Pour tester l'algorithme, on utilise un jeu de données supplémentaires normalisées dont l'état des patients est connu (100 patients). On note $datatest$ ces données et $etattest$ le vecteur d'état connu pour ces patients. On applique ensuite l'algorithme sur chaque élément de ce jeu de données pour une valeur de K fixée.

On définit la fonction suivante qui renvoie une matrice appelée "matrice de confusion".

```

1 | def test_KNN( datatest , etattest ,data , etat ,K, nb ) :
2 |     etatpredit = []
3 |     for i in range(len(datatest)):
4 |         res = KNN( data , etat ,datatest[i] ,K, nb )
5 |         etatpredit.append(res)
6 |
7 |     mat = np.zeros((nb ,nb ))
8 |     for i in range(len(etattest)):
9 |         mat[ etattest[i] , etatpredit[i] ] += 1
10 |    return mat

```

On obtient pour $K = 8$ la matrice suivante :
$$\begin{pmatrix} 23 & 4 & 7 \\ 7 & 11 & 1 \\ 5 & 2 & 40 \end{pmatrix}.$$

- Q15.** Indiquer l'information apportée par la diagonale de la matrice. Exploiter les valeurs de la première ligne de cette matrice en expliquant les informations que l'on peut en tirer. Faire de même avec la première colonne. En déduire à quoi sert cette matrice.

On peut également tracer l'efficacité de l'algorithme pour différentes valeurs de K sur le jeu de données test (100 éléments sur un échantillon de 310 données). On trace le pourcentage de réussite en fonction de la valeur de K sur la **figure 4**.

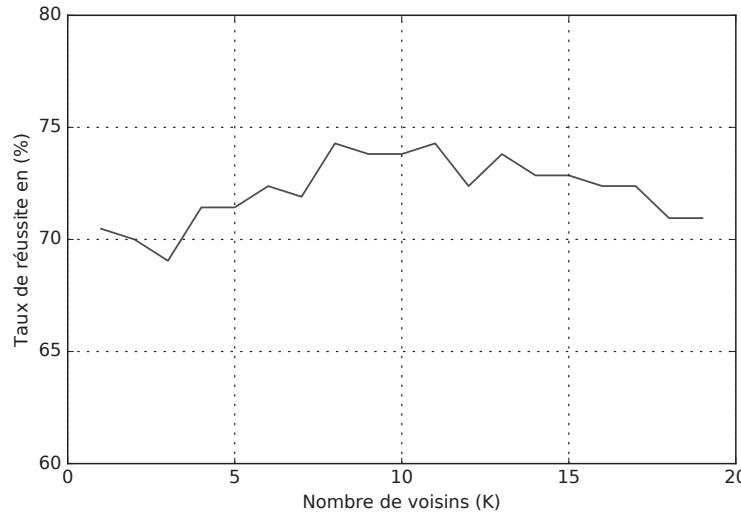


Figure 4 – Pourcentage de réussite de l'algorithme en fonction de K

- Q16.** Commenter la courbe obtenue et critiquer l'efficacité de l'algorithme.

III.2 - Méthode de classification naïve bayésienne

Une autre méthode simple à mettre en oeuvre et qui fournit de bons résultats, malgré l'hypothèse forte utilisée, est la classification naïve bayésienne. Cette méthode d'apprentissage est supervisée également. La méthode repose sur le théorème de Bayes qui suppose l'indépendance probabilistique des caractéristiques d'un groupe donné.

On suppose que chaque attribut i (colonne de la matrice de données `data`) est modélisable par une variable aléatoire notée X_i pour $i \in \llbracket 0, n-1 \rrbracket$. L'appartenance d'une donnée à un groupe est modélisée par la variable aléatoire Y , dont les valeurs discrètes sont $y_0 = 0$ pour un sujet sain, $y_1 = 1$ pour un sujet atteint d'une hernie discale et $y_2 = 2$ pour un sujet atteint de spondylolisthésis.

L'appartenance de la donnée k (k^{e} ligne de la matrice de données) est connue et indiquée dans le vecteur `etat`.

Le théorème de Bayes permet de déterminer la probabilité qu'une donnée appartienne à un groupe y_j connaissant ses attributs x_i

$$P(Y = y_j | X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}) = \frac{P(Y = y_j) \times P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1} | Y = y_j)}{P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1})}$$

où :

- $P(Y = y_j|X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1})$ est la probabilité d'appartenir au groupe y_j sachant que les différentes variables aléatoires X_i prennent respectivement les valeurs x_i ,
- $P(Y = y_j)$ est la probabilité d'appartenir au groupe y_j ,
- $P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}|Y = y_j)$ est la probabilité que les différentes variables aléatoires X_i prennent respectivement les valeurs x_i sachant que la donnée appartient au groupe y_j ,
- $P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1})$ est la probabilité que les différentes variables aléatoires X_i prennent respectivement les valeurs x_i .

L'hypothèse naïve bayésienne suppose que tous les attributs sont indépendants et donc que :

$$P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}|Y = y_j) = \prod_i P(X_i = x_i|Y = y_j).$$

Comme le dénominateur $P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1})$ est indépendant du groupe considéré et donc constant, on ne considère que le numérateur :

$$P(Y = y_j) \times P(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}|Y = y_j).$$

Pour déterminer le groupe le plus probable auquel appartient une donnée z à classer, représentée par le n -uplet $(z_0, z_1, \dots, z_{n-1})$, on choisit la probabilité maximale $P(Y = y_j) \times \prod_i P(X_i = z_i|Y = y_j)$ parmi les j groupes.

Des lois de probabilités diverses sont utilisées pour estimer $P(X_i = z_i|Y = y_j)$; nous utiliserons une loi de distribution gaussienne.

Apprentissage

La première étape consiste à séparer les données selon les groupes auxquels elles appartiennent. On réutilise pour cela la fonction `separationParGroupe(data, etat)` définie à la question **Q6**.

Pour calculer la probabilité conditionnelle $P(X_i = z_i|Y = y_j)$ d'une donnée z représentée par le n -uplet $(z_0, z_1, \dots, z_{n-1})$, on utilise une distribution gaussienne de la forme

$$P(X_i = z_i|Y = y_j) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_i - \mu_{x_i,y_j})^2}{2\sigma_{x_i,y_j}^2}\right)$$

où μ_{x_i,y_j} et σ_{x_i,y_j}^2 sont la moyenne et la variance de la variable aléatoire X_i , estimées à partir des valeurs d'attribut des données de la i -ième colonne du tableau `data` pour le groupe correspondant à y_j .

On rappelle que, pour un vecteur x de dimension n , $\mu = \frac{\sum_{i=0}^{n-1} x_i}{n}$ et $\sigma^2 = \frac{\sum_{i=0}^{n-1} (x_i - \mu)^2}{n}$.

Q17. Écrire deux fonctions de complexité linéaire `moyenne(x)` et `variance(x)` permettant de renvoyer la moyenne et la variance d'un vecteur x de dimension quelconque.

Q18. Proposer une fonction `synthese(data, etat)` qui renvoie une liste composée de doublets [`moyenne`, `variance`] pour chaque attribut de la matrice `data` en les regroupant selon les valeurs du vecteur `etat` :

$$\begin{aligned} & [[[[\mu_{x_0,y_0}, \sigma_{x_0,y_0}], [\mu_{x_1,y_0}, \sigma_{x_1,y_0}], \dots, [\mu_{x_5,y_0}, \sigma_{x_5,y_0}]], \\ & [[[\mu_{x_0,y_1}, \sigma_{x_0,y_1}], [\mu_{x_1,y_1}, \sigma_{x_1,y_1}], \dots, [\mu_{x_5,y_1}, \sigma_{x_5,y_1}]], \\ & [[[\mu_{x_0,y_2}, \sigma_{x_0,y_2}], [\mu_{x_1,y_2}, \sigma_{x_1,y_2}], \dots, [\mu_{x_5,y_2}, \sigma_{x_5,y_2}]]]. \end{aligned}$$

Cette fonction appliquée au tableau `data` complet (non normalisé) renvoie une liste qui contient les valeurs suivantes :

	x_1	x_2	x_3	x_4	x_5	x_6
etat=0	[51.68, 12.3]	[12.82, 6.74]	[43.54, 12.29]	[38.86, 9.57]	[123.89, 8.96]	[2.18, 6.27]
etat=1	[47.63, 10.6]	[17.39, 6.95]	[35.46, 9.68]	[30.23, 7.49]	[116.47, 9.27]	[2.48, 5.48]
etat=2	[71.51, 15.05]	[20.74, 11.46]	[64.11, 16.34]	[50.76, 12.27]	[114.51, 15.52]	[51.89, 39.97]

Prédiction

On considère pour la deuxième étape une donnée z à classer représentée par le n -uplet $(z_0, z_1, \dots, z_{n-1})$. Pour réaliser la prédiction, il faut tout d'abord calculer la probabilité d'appartenance à un groupe selon la loi gaussienne choisie $P(X_i = z_i | Y = y_j)$ en fonction de l'attribut considéré, puis multiplier ces probabilités pour un même groupe y_j pour obtenir $\prod_i P(X_i = z_i | Y = y_j)$.

Q19. Écrire une fonction `gaussienne(a, moy, v)` qui calcule la probabilité selon une loi gaussienne de moyenne moy et de variance v pour un élément a de \mathbf{R} .

Q20. Déduire de la description de l'algorithme de classification naïve bayésienne une fonction `probabiliteGroupe(z, data, etat)` prenant en argument le vecteur z qui contient le n -uplet de la donnée z , le tableau des données connues `data` et le vecteur d'appartenance à un groupe de chacune de ces données `etat`. Cette fonction renvoie la probabilité d'appartenance à chacun des $nb = 3$ groupes sous la forme d'une liste de trois valeurs. On utilisera la fonction `synthese(data, etat)` et la fonction `gaussienne(a, moy, v)`.

Sur le patient dont les caractéristiques sont les suivantes : 48,26, 16,42, 36,33, 31,84, 94,88, 28,34, on obtient les probabilités suivantes : appartenance au groupe 0 : $1,1 \cdot 10^{-15}$, appartenance au groupe 1 : $7,4 \cdot 10^{-16}$, appartenance au groupe 2 : $5,4 \cdot 10^{-13}$.

La décision de l'appartenance à un groupe particulier est prise en déterminant le maximum parmi les probabilités de groupes déterminées.

Q21. Écrire une fonction `prediction`, dont vous préciserez les arguments, qui renvoit le numéro du groupe auquel appartient un élément z .

Usuellement la méthode naïve bayésienne est utilisée dans un espace logarithmique ; c'est-à-dire qu'au lieu de calculer la probabilité d'appartenance à un groupe comme vu précédemment, on calcule le logarithme de cette quantité.

Q22. Proposer une explication qui justifie cette utilisation du logarithme.

L'algorithme mis en place peut être testé sur le jeu de 100 données test utilisé pour l'algorithme KNN dont on connaît déjà l'appartenance à chacun des groupes. On obtient alors la matrice de confusion

suivante :
$$\begin{pmatrix} 23 & 9 & 8 \\ 9 & 10 & 1 \\ 10 & 1 & 49 \end{pmatrix}$$

Q23. Calculer le pourcentage de réussite de la méthode KNN, à partir de la matrice de confusion pour $K = 8$ (page 8), ainsi que celui de la méthode naïve bayésienne à partir de la matrice ci-dessus. Discuter de la pertinence de chacune des deux méthodes sur l'exemple traité.

FIN

ANNEXE

Rappels des syntaxes en Python

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	L=[1,2,3] (liste) v=array([1,2,3]) (vecteur)
accéder à un élément	v[0] renvoie 1 (L[0] également)
ajouter un élément	L.append(5) uniquement sur les listes
tableau à deux dimensions (matrice)	M=array([[1,2,3],[3,4,5]])
accéder à un élément	M[1,2] donne 5
extraire une portion de tableau (2 premières colonnes)	M[:,0:2]
extraire la colonne i	M[:,i]
extraire la ligne i	M[i,:]
tableau de 0 (2 lignes, 3 colonnes)	zeros((2,3))
dimension d'un tableau T de taille (i, j)	T.shape donne [i,j]
produit matrice-vecteur	a = array([[1,2,3],[4,5,6],[7,8,9]]) b = array([1,2,3]) print(a.dot(b)) >>> array([14,32,50])
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	arange(0,10.1,0.1)
définir une chaîne de caractères	mot='Python'
taille d'une chaîne	len(mot)
extraire des caractères	mot[2:7]
boucle For	for i in range(10): print(i)
condition If	if (i>3): print(i) else: print('hello')
définir une fonction qui possède un argument et renvoie 2 résultats	def f(param): a = param b = param*param return a,b
tracé d'une courbe de deux listes de points x et y	plot(x,y)
tracé d'une courbe de trois listes de points x, y et z	gca(projection='3d').plot(x,y,z)
ajout d'un titre sur les axes d'une figure	xlabel(texte) ylabel(texte)
ajout d'un titre principe sur une figure	title(texte)

SESSION 2019

TSIIN07



ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Jeudi 2 mai : 14 h - 17 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet comporte 12 pages. Seul le **Document Réponse** de 11 pages est à rendre en fin d'épreuve.

Remarques générales

Important : nous informons les candidats que chaque partie de ce sujet peut être traitée séparément. Vous pouvez utiliser toutes les fonctions des questions précédentes même si vous ne les avez pas implémentées.

Les réponses aux questions sont à rédiger sur le **Document Réponse (DR)** et ne doivent pas dépasser les dimensions des cadres proposés.

Si la réponse attendue est spécifique à un langage de programmation, seul le **langage Python** est permis.

Les structures algorithmiques doivent être clairement identifiables par des indentations visibles ou par des barres droites entre le début et la fin de la structure comme l'exemple ci-dessous :

```
si (Condition)
    alors
        | Instructions
    sinon
        | Instructions
fin si
```

Dans les questions où l'on demandera d'écrire une fonction, on donnera systématiquement sa signature, c'est-à-dire le nom de la fonction avec les types des paramètres ainsi que le type du résultat retourné par cette fonction.

Par exemple, la fonction **foo** qui prend en entrée deux paramètres de type entier et retourne une liste, aura comme description :

Signature de la fonction *foo*:

`foo(int, int) -> list`

SÉCURISATION DE L'ENTRÉE DU PERSONNEL D'UNE ENTREPRISE

Partie I - Présentation

Une entreprise possède 5 sites de production. Le Directeur Général veut améliorer la sécurité. Il souhaite attribuer à chaque employé une carte personnelle et infalsifiable lui permettant l'accès à son site de production et peut-être à d'autres sites de l'entreprise. Chaque employé est affecté uniquement à un site que l'on appellera son site d'origine.

Chaque site de production ne pourra compter plus de 100 employés. La carte attribuée à l'employé comportera sa photo d'identité ainsi qu'une puce. Le code représentant le nom de la personne sera intégré dans la photo mais pas dans la puce.

Un tel code a été placé dans la photo de droite de la **figure 1**. Comme on peut le remarquer, il y a très peu de différences entre les deux photos, l'une sans code, l'autre avec un code intégré.

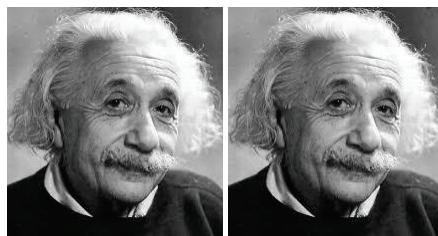


Figure 1 – Photos d'Einstein

Chaque employé sera référencé par un code composé d'un entier compris entre 1 et 5 (indiquant le site de référence où il travaille) ainsi qu'une séquence de 7 caractères (pris parmi les 7 premiers caractères de l'alphabet en majuscules : 'A' ... 'G'). Le site sera codé en machine sur 3 bits. Par exemple, pour un employé travaillant dans le site n° 3, le code pourrait être 3DABGEFC. Ce code n'affiche pas le nom de l'employé mais un automate permet, à partir des 7 caractères, de le retrouver.

Q1. Donner le nombre maximal d'employés par site ainsi que le nombre maximal de sites que pourrait gérer cette entreprise avec ce type de codage. La réponse peut être donnée par une expression numérique sans chercher à la calculer.

Indiquer si la technique de codage est suffisante pour gérer le personnel de cette entreprise.

Partie II - Distance de *Levenshtein*

La problématique est de savoir si les codes créés pour gérer les employés sont suffisamment disjoints les uns des autres, c'est-à-dire si la mesure de la différence entre deux codes est suffisamment importante. Ceci est possible grâce à la distance de *Levenshtein* qui peut être utilisée pour analyser tous les codes. L'entreprise veut tester tous les codes associés aux employés et changer les codes de tous ceux dont la distance de *Levenshtein* est inférieure à la valeur 3.

Dans cette partie, nous allons développer les fonctions permettant de répondre à cette problématique.

La distance de *Levenshtein* est une valeur donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne A à une chaîne B.

On considère que chaque opération élémentaire mise en oeuvre (supprimer, insérer ou remplacer) a un coût de 1. On précise que chaque opération élémentaire ne porte que sur un seul caractère. La distance de *Levenshtein*, au sens mathématiques du terme, est la somme de ces coûts.

Par exemple, la distance de *Levenshtein* entre les chaînes *Soveil* et *Soleil* est de coût égal à 1 car il a fallu réaliser une seule transformation, une substitution pour remplacer 'v' par 'l' dans la chaîne *Soveil*.

On va s'intéresser à déterminer quelle est la distance de *Levenshtein* entre les chaînes de caractères *Auttame* et *Automne*.

Q2. Écrire, dans le langage Python, les transformations (insertion, substitution, suppression) appliquées à la chaîne *Auttame* afin d'obtenir la chaîne *Automne*. On déterminera systématiquement la valeur de la chaîne, notée *s*, avant et après chaque transformation ainsi que le nom de la transformation appliquée.

Vous devez compléter le **Document Réponse**, à l'aide de *s=Auttame*.

Calculer le coût de la distance de *Levenshtein*.

L'algorithme qui vous est proposé ci-après décrit le calcul de la distance de *Levenshtein* entre deux chaînes de caractères. Cet algorithme retourne un entier (éventuellement nul) donnant la distance entre deux chaînes de caractères au sens de *Levenshtein*.

```

1 levenshtein(chaine1, chaine2) =
2 entree :
3     chaine1[], chaine de caracteres d'indice 1 a n avec n egal a longueurChaine1 ;
4     chaine2[], chaine de caracteres d'indice 1 a m avec m egal a longueurChaine2 ;
5 sortie :
6     entier : la valeur de la distance ;
7 declarer :
8     d[] : matrice de nombres entiers initialisée avec des 0, indice de 0 a n et de 0 a m ;
9     i, j, coutSubstitution : entier ;
10 debut :
11     pour i de 0 a n
12         d[i,0] := i
13     pour j de 0 a m
14         d[0,j] := j
15     pour i de 1 a n
16         pour j de 1 a m
17             si (chaine1[i] = chaine2[j])
18                 alors
19                     coutSubstitution := 0
20                 sinon
21                     coutSubstitution := 1
22             fin si
23             d[i,j] := minimum(
24                 d[i-1,j] + 1,                                # pour une suppression
25                 d[i,j-1] + 1,                                # pour une insertion
26                 d[i-1,j-1] + coutSubstitution)            # pour une substitution
27     retourner d[n, m]
28 fin
```

Q3. Déterminer la matrice d de l'algorithme de *Levenshtein* après l'instruction de la ligne 14 et après celle de la ligne 26, ainsi que la distance de *Levenshtein* lors de son exécution entre les chaînes 'GAZ' et 'LA'.

Q4. Déterminer la complexité de l'algorithme précédent.

Tous les codes ont été rassemblés dans une liste nommée *table_code*.

Q5. Écrire une fonction *code_bon_lvs* qui supprime tous les codes de la liste *table_code* dont la mesure de *Levenshtein* est inférieure ou égale à la valeur 3. Les codes à supprimer seront remplacés dans la liste *table_code* par le code nul, c'est-à-dire la chaîne de caractères '0000000'.

Signature de la fonction *code_bon_lvs*:

code_bon_lvs(list) -> *NoneType*

Q6. Écrire une fonction *pourcentage_lvs* qui détermine le pourcentage de codes nuls dans la liste *table_code*. Le résultat sera une chaîne de caractères constituée d'un nombre entier (pas un nombre flottant) et du caractère %. On permet l'arrondi d'un calcul à l'entier inférieur ou à l'entier supérieur.

Signature de la fonction *pourcentage_lvs*:

pourcentage_lvs(list) -> *str*

Exemple :

```
>>> pourcentage_lvs(table_code)
'17%'
```

Partie III - Gestion des données

Nous allons nous intéresser à la gestion des données qui sont stockées dans la base de données nommée *Personnel* et sera constituée de deux tables intitulées *Employes* et *ListeCategories*. Les contenus de ces tables se trouvent en **Annexe**, page 12.

La table *Employes* est constituée de 8 champs :

- *id*: de type INTEGER ;
- *nom*: de type TEXT ;
- *prenom*: de type TEXT ;
- *email*: de type TEXT – clé primaire (l'adresse email est définie sans son extension @cpp.com dans la table) ;
- *age*: de type INTEGER ;
- *code*: de type TEXT (voir descriptif plus loin) ;
- *site*: de type INTEGER – liste des sites où l'employé est autorisé à entrer en plus de son site d'origine (voir descriptif plus loin) ;
- *code_categorie*: de type INTEGER – indice où la catégorie est référencée dans la table nommée *ListeCategories* (clé étrangère).

La table *ListeCategories* est constituée de 2 champs :

- *id*: de type INTEGER – clé primaire ;
- *categorie*: de type TEXT – liste des métiers de l’entreprise.

Des lecteurs de codes sont installés aux portes des différents sites afin de limiter les accès aux seules personnes habilitées à y entrer.

Rappelons que le code est défini sous la forme d’une chaîne de 8 caractères, le premier caractère indiquant le numéro du site auquel appartient une personne. Par défaut, chaque personne est attribuée à un et un seul site, son site d’origine. Par contre, il est possible à toute personne de cette entreprise de se déplacer dans d’autres sites si l’autorisation en a été donnée (champ *site* de la table *Employes*). Le reste des caractères correspond à la clé de sécurité associée à chaque employé.

La gestion du champ *site* est particulière. On définit un entier qui permet de connaître les sites où l’employé est autorisé à entrer. On attribue les valeurs suivantes :

- 1 : pour le site numéroté 1 ;
- 2 : pour le site numéroté 2 ;
- 4 : pour le site numéroté 3 ;
- 8 : pour le site numéroté 4 ;
- 16 : pour le site numéroté 5.

Ainsi, si un employé appartenant au site 1 est autorisé à entrer dans les sites 2 et 3, la valeur du champ *site* aura comme valeur $2 + 4$, soit 6. Pour un autre employé appartenant au site 4 autorisé à entrer dans les sites 1, 2 et 5, la valeur du champ *site* sera $1 + 2 + 16$, soit 19.

Attention : comme on peut le remarquer sur les deux exemples précédents, le numéro du site d’origine (le site où l’employé travaille par défaut) n’est jamais pris en compte dans le calcul donnant la valeur du champ *site*.

Q7. Écrire une fonction *liste_site* donnant la liste des sites où un employé peut se rendre en plus de son site d’origine dès que l’on donne une valeur (un entier) représentant la valeur du champ *site* de la table *Employes*. Si la valeur donnée est incorrecte, la fonction *liste_site* retournera une chaîne vide.

L’ordre des sites dans le résultat n’a pas d’importance.

Signature de la fonction *liste_site*:

`liste_site(int) -> list`

Exemple :

```
>>> liste_site(21)
[1, 3, 5]

>>> liste_site(50)
[]
```

L’équipe de direction souhaite avoir certaines informations au sujet des employés de l’entreprise.

Q8. Écrire en SQL la requête 1 suivante donnée en algèbre relationnel :

$$\pi_{nom,prenom,age}(\sigma_{age > 50}(Employes))$$

Q9. Écrire en SQL la requête 2 donnant comme résultat l'adresse email (sans le nom de domaine) des employés pouvant accéder uniquement aux sites 3 et 4 en plus de leur site d'origine.

Q10. Écrire en SQL la requête 3 donnant comme résultat le nom et la catégorie des personnes de l'entreprise ayant au moins 20 ans. Les noms seront classés par ordre alphabétique.

Q11. Écrire en SQL la requête 4 donnant comme résultat la liste des âges des employés avec comme information associée le nombre d'employés ayant le même âge. On demande que cette liste soit décrémentale par rapport au nombre de personnes ayant le même âge.

Par exemple, dans la table *Employes*, il y a 2 personnes qui ont 22 ans.

Partie IV - Codage des couleurs

La photo, une image couleur, est décrite informatiquement comme un tableau (une matrice) de pixels. Chaque pixel sera représenté par une couleur au format RGB¹.

Par exemple, la couleur d'un pixel pourrait être en format RGB (64,78,191). Si on écrit les trois composantes RGB en code binaire, on aura le triplet (01000000, 01001110, 10111111). Le bit de poids faible de chacune des composantes RGB est représenté en gras. On va se servir de ce triplet de valeurs définies en gras pour coder une information dans l'image.

La modification du bit de poids faible a très peu de conséquences sur la représentation de l'image.

Q12. Donner le code RGB en décimal et en hexadécimal des éléments suivants :

- un pixel de couleur bleu ;
- un pixel de couleur blanc.

Un pixel est codé dans le format RGB en (10,10,10). Indiquez la couleur de ce pixel.

Q13. Donner pour le codage RGB le nombre de couleurs possibles. La réponse peut être donnée par une expression numérique sans chercher à la calculer.

1.

Le système RGB (Red, Green, Blue) ou en français (Rouge, Vert, Bleu), permet de coder les couleurs en informatique. Un écran informatique est composé de pixels représentant une couleur au format RGB.

La composante R est codée sur 8 bits de 0 à 255 en décimal et de 00 à FF en hexadécimal. Il en va de même pour les autres composantes. Le codage des couleurs va du plus foncé au plus clair.

Par exemple, la couleur d'un pixel orange pourrait avoir comme valeur (255,100,100). Un pixel sera de couleur gris si les composantes R, G et B sont identiques.

Partie V - Codage de l'information

Cette partie sera consacrée à l'implémentation de fonctions qui serviront notamment dans la partie VI, réservée au décodage de l'information se trouvant dans la photo de l'employé(e).

Les informations constituant le code sont définies dans une trame composée de blocs non consécutifs. Chaque bloc sera associé à un pixel. Un premier bloc est constitué d'un pixel pour représenter le numéro du site et est suivi d'une séquence de 7 blocs représentant chacun un caractère. Le premier bloc sera considéré comme le bloc de référence.

Pour un pixel codant le numéro du site de l'employé, les bits de poids faible indiquent directement le numéro au format binaire. Ainsi, pour l'exemple précédent, le codage correspondant au site n°3 sera (01000000, 01001111, 10111111) puisque le triplet 011 correspond à la valeur 3 en code décimal. Par conséquent, on met une information dans l'image en modifiant uniquement les bits de poids faible, ce qui a peu de conséquences sur la représentation de cette image.

Pour un pixel codant une des lettres de l'identifiant de l'employé, on valide la convention suivante : le caractère 'A' sera associé à 001, le caractère 'B' à 010, jusqu'au caractère 'G' qui aura comme valeur 111.

Par exemple, un pixel codant le caractère 'B' (010) pourrait avoir pour format RGB (...0, ...1, ...0). On ne traitera pas le caractère associé à 000 qui aura pour lettre 'W', réservée pour la gestion du personnel de service (gardiens, personnel de nettoyage, etc.) ayant l'autorisation d'entrer dans les sites de l'entreprise.

Chaque trame sera constituée de 8 blocs, c'est-à-dire 8 pixels (voir la **figure 2**).

- Un pixel de référence est défini par ses coordonnées que l'on nomme *posi* dans la suite.
- Le paramètre ou la variable *posi* est un couple de valeurs (colonne,ligne) qui est obtenu à l'aide d'une clé se trouvant dans la puce. Les valeurs des coordonnées nommées *posi* sont supposées données.
- Les caractères définis dans le code sont, dans l'image, décalés d'une valeur *delta* qui est déterminée à partir du pixel de référence en calculant la somme des valeurs de ses composantes R,G,B. Autrement dit, si le pixel de référence est à la position (*x,y*) et si la valeur *x+delta* n'est pas supérieure à la largeur (*imx*) de l'image, le premier bloc sera à la position (*x+delta,y*), sinon le bloc se trouvera sur la ligne suivante et ceci jusqu'au 7^e caractère défini dans le code de l'employé.

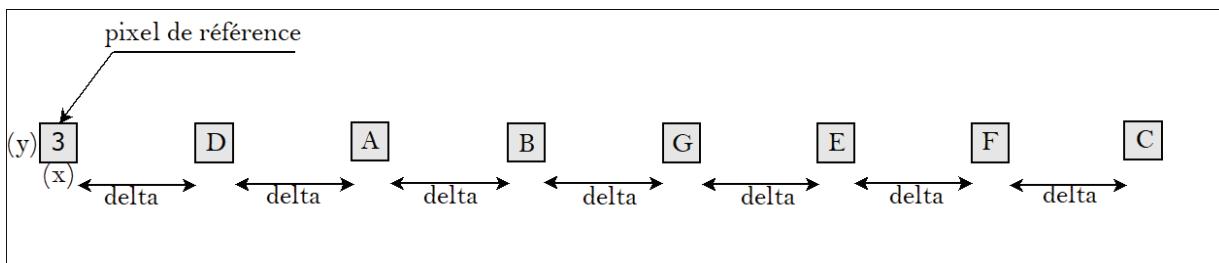


Figure 2 – Trame

Q14. Écrire une fonction *bin20* qui, à partir d'un nombre entier, retourne un nombre en code binaire. Cette fonction convertit un nombre entier en une chaîne de chiffres binaires sans le *0b* devant, comme le ferait la fonction *bin()* de Python. Le résultat sera toujours écrit sous la forme d'une séquence d'au moins 3 digits de type *string*.

Remarque : pour l'implémentation de cette fonction, l'utilisation de la fonction `bin()` est possible. On rappelle que la fonction `bin()` convertit et retourne la chaîne de caractères en format binaire d'un entier donné. Par exemple, `bin(12)` retourne comme valeur la chaîne de caractères '`0b1100`'.

Signature de la fonction `bin2`:

`bin2(int) -> str`

Exemples :

```
>>> bin2(45)
'101101'
>>> bin2(1)
'001'
```

Q15. Écrire la fonction `num()` qui calcule à partir d'une chaîne en chiffres binaires, par exemple '`010`', sa valeur en décimal.

Si la chaîne est vide, la fonction retournera comme valeur `-1`.

Signature de la fonction `num`:

`num(str) -> int`

Exemples :

```
>>> num('101')
5
>>> num('')
-1
```

Partie VI - Décodage de l'information

Cette partie sera consacrée à l'implémentation de fonctions pour le décodage de l'information.

Remarque : vous pouvez utiliser les fonctions des questions précédentes même si elles n'ont pas été traitées.

On donne le code suivant :

```
1  from PIL import Image
2  import os
3  os.chdir("C:\\photos")      # Positionnement sur le bon répertoire.
4  im = Image.open("a123.bmp")   # Lecture de l'image de l'employé dont la
                                # photo est a123.
5  imx, imy = im.size          # La taille de l'image (largeur, hauteur).
                                # Résultat : (277, 250).
6  posi = (30,20)              # Position du pixel (x,y).
                                # Récupère la valeur au format RGB du pixel
                                # situé aux coordonnées (30,20).
7                                # Résultat : (2, 57, 139).
8                                # Les indices lignes et colonnes débutent à 0.
```

Lorsque l'employé passe sa carte sur le lecteur pour ouvrir une porte, les données de la puce et le code se trouvant dans la photo sont lus par le lecteur qui génère 2 valeurs envoyées au processus d'analyse (dont vous allez implémenter certaines fonctions). Ces 2 valeurs sont le pixel de référence (avec sa valeur *posi*) et le code associé à l'employé (un entier suivi de 7 caractères). Le pixel de référence est différent pour chacun des employés.

Les caractères du code dans l'image ne sont pas placés d'une manière consécutive mais séparés les uns des autres par une même valeur (*delta*) calculée à l'aide des valeurs RGB du pixel de référence (voir la **figure 2**) en réalisant la somme des 3 composantes R, G et B.

Q16. Écrire la fonction *lettre()* qui, à partir d'une chaîne de 3 bits, détermine le caractère. La codification des 7 caractères a été réalisée sur seulement 3 bits. On rappelle que le caractère 'A' est codé '001', le caractère 'B' est codé '010' et ainsi de suite jusqu'au caractère 'G' qui est codé '111', ceci en suivant l'ordre des 7 premières lettres de l'alphabet.

Signature de la fonction *lettre*:

```
lettre(str) -> str
```

Exemples :

```
>>> lettre('001')
'A'
>>> lettre('101')
'E'
```

Q17. Écrire la fonction *bpf* qui récupère les valeurs des bits de poids faible d'un pixel.

Cette fonction a 2 paramètres, l'image et la position du pixel. Elle retourne une chaîne de caractères composée de 3 bits.

Signature de la fonction *bpf*:

```
bpf(BmpImageFile, (int,int)) -> str
```

Exemple :

```
>>> bpf(im, pos)
'011'
```

Q18. Soit la fonction *valeur_delta()* dont le code est :

```
1 def valeur_delta(im,posi):
2     px = im.getpixel(posi)
3     inter = (px[0] + px[1] + px[2]) % (128-41) + 40
4     if inter % 2 == 1 :
5         return inter + 1
6     else :
7         return inter
```

Cette fonction a 2 paramètres : l'image et les coordonnées du point de référence (le pixel du point de référence).

Signature de la fonction *valeur_delta*:

```
valeur_delta(BmpImageFile, (int,int)) -> int
```

Indiquer ce que fait cette fonction en précisant les valeurs retournées. Le résultat sera décrit sous la forme d'un intervalle ou d'une union d'intervalles de nombres entiers.

Les deux fonctions suivantes *position_bloc()* et *num_site()* ne sont pas à implémenter.

La fonction *position_bloc()* détermine les coordonnées d'un bloc de la trame se trouvant à un indice compris entre 0 et 7, l'indice 0 représentant le pixel de référence. Le premier bloc des caractères du code se trouve à l'indice 1 et le dernier à l'indice 7. Rappelons que l'indice 0 (le pixel de référence) est le bloc constitué du numéro du site où l'employé travaille.

La fonction *position_bloc()* a 4 paramètres : la position initiale du pixel de référence, la valeur de l'intervalle, la largeur de l'image et l'indice du bloc. Elle retourne les coordonnées du pixel du bloc recherché.

Signature de la fonction *position_bloc*:

```
position_bloc((int,int),int,int,int) -> (int,int)
```

Exemple:

```
>>> position_bloc(posi,70,imx,1)  
(100, 20)
```

La fonction *num_site()* retourne comme résultat le numéro du site où la personne travaille. Cette fonction a 2 paramètres : l'image et la position du pixel de référence.

Signature de la fonction *num_site*:

```
num_site(BmpImageFile,(int,int)) -> int
```

Exemple:

```
>>> num_site(im, posi)  
3
```

Q19. Écrire la fonction *lecture_lettres()* qui retourne la séquence des lettres codées dans l'image, c'est-à-dire les 7 lettres constituant une partie du code de l'employé.

Cette fonction a 2 paramètres, l'image et la position du pixel de référence. Elle retourne une chaîne de caractères.

Signature de la fonction *lecture_lettres*:

```
lecture_lettres(BmpImageFile,(int,int)) -> str
```

Exemple:

```
>>> lecture_lettres(im,posi)  
'DABGEFC'
```

Q20. Indiquer ce que fait la fonction suivante :

```
1 def lecture_code(im, posi):  
2     num = num_site(im, posi)  
3     msg = lecture_lettres(im, posi)  
4     return str(num) + msg
```

Annexe

Base de données “Personnel”

Le contenu des tables *Employes* et *ListeCategories* de la base de données *Personnel* est donnée ci-après.

Table “*Employes*”

id	nom	prenom	email	age	code	site	code_categorie
1	Genereux	Alain	alain.genereux	47	1AACDEF	0	1
2	Tanguy	Alain	alain.tanguy	22	1BBACABE	10	8
3	Smith	Alan	alan.smith	47	5BCABCAC	5	2
4	Lefoll	Claude	claude.lefoll	28	2EEABECC	24	1
5	Herberts	Dany	dany.herberts	58	1EEEEABC	0	1
6	Korbs	Eva	eva.korbs	33	2FEAFEAB	9	5
7	Niels	Edwin	edwin.niels	24	2EFDDACA	28	4
8	Joly	Emilie	emilie.joly	25	3FAFABEF	19	4
9	Esteban	Flore	flore.esteban	20	4BECEBAA	12	2
10	Serin	Jacques	jacques.serin	22	2GBBCEAE	21	6
11	Clerc	Jerome	jerome.clerc	29	3DDAABBC	3	1
12	Brown	Katia	katia.brown	46	5AFBECCA	3	1
13	Forbs	Laura	laura.forbs	18	2CBAEAEC	0	1
14	Phil	Marc	marc.phil	33	4BCDABCD	17	7
15	Auzas	Michel	michel.auzas	32	5FECDAEA	15	7
16	Kotta	Michelle	micelle.kotta	27	2ABEEABC	9	7
17	Lambert	Pierre	pierre.lambert	35	3DABGEFC	10	2
18	Klader	Sylvie	sylvie.klader	32	2EAEAEAB	20	1
19	Durand	Sylvie	sylvie.durand	31	1CBBCAFA	16	6
20	Olm	Tatiana	tatiana.olm	25	4EFEFACB	19	3

Table “*ListeCategories*”

id	categorie
1	ingénieur
2	secrétaire
3	comptable
4	technicien
5	opérateur
6	administratif
7	chef de projet
8	responsable de département

FIN

	Numéro d'inscription	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Nom : _____
	Numéro de table	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Prénom : _____
	Né(e) le	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
Emplacement QR Code	Filière : TSI Session : 2019		
Épreuve de : Informatique			
Consignes <ul style="list-style-type: none"> • Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer • Rédiger avec un stylo non effaçable bleu ou noir • Ne rien écrire dans les marges (gauche et droite) • Numérotter chaque page (cadre en bas à droite) • Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre 			

TSIIN07

DOCUMENT RÉPONSE

SÉCURISATION D'UNE ENTREPRISE

Q1

Nombre maximal d'employés par site :

Nombre maximal de sites :

Réponse justifiée :

B

NE RIEN ÉCRIRE DANS CE CADRE

— Q2 —

1) Première séquence

Valeur de s avant la transformation :

Nom de la transformation :

Code Python :

Valeur de s après la transformation :

2) Deuxième séquence

Valeur de s avant la transformation :

Nom de la transformation :

Code Python :

Valeur de s après la transformation :

.../...

Q2 suite3) Troisième séquence

Valeur de s avant la transformation :

Nom de la transformation :

Code Python :

Valeur de s après la transformation :

Coût de cette distance :

Q3

Après la ligne 14, matrice d :

Après la ligne 26, matrice d :

Distance de Levenshtein =

Q4

Complexité :

Q5

```
def code_bon_lvs(t):
```

Q6

```
def pourcentage_lvs(t):
```

 CONCOURS COMMUN INP	Numéro d'inscription	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Nom :	
	Numéro de table	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Prénom :	
	Né(e) le	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>		
Emplacement QR Code	Filière : TSI		Session : 2019	
	Épreuve de : Informatique			
Consignes	<ul style="list-style-type: none">• Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer• Rédiger avec un stylo non effaçable bleu ou noir• Ne rien écrire dans les marges (gauche et droite)• Numérotter chaque page (cadre en bas à droite)• Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre			

TSIIN07

Q7

```
def liste_site(valeur):
```

(C)

5/11

NE RIEN ÉCRIRE DANS CE CADRE

Q8

Requête 1 =

Q9

Requête 2 =

Q10

Requête 3 =

Q11

Requête 4 =

Q12

Pixel bleu :

Pixel blanc :

Couleur du pixel au format RGB (10,10,10) :

Q13

Nombre de possibilités pour le code RGB :

Q14

```
def bin2(n):
```

Q15

```
def num(ch):
```

 CONCOURS COMMUN INP	Numéro d'inscription	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Nom :	
	Numéro de table	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Prénom :	
	Né(e) le	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>		
Emplacement QR Code	Filière : TSI		Session : 2019	
	Épreuve de : Informatique			
Consignes	<ul style="list-style-type: none">• Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer• Rédiger avec un stylo non effaçable bleu ou noir• Ne rien écrire dans les marges (gauche et droite)• Numérotter chaque page (cadre en bas à droite)• Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre			

TSIIN07

Q16 _____

```
def lettre(ch_bin):
```

Q17 _____

```
def bpf(im, pos):
```

(D)

9/11

NE RIEN ÉCRIRE DANS CE CADRE

Q18

Explication :

.....
.....
.....
.....
.....

Résultat :

Q19

```
def lecture_lettres(im,posi):
```

Q20

Réponse :

FIN



ÉPREUVE SPÉCIFIQUE - FILIÈRE PC

MODÉLISATION DE SYSTÈMES PHYSIQUES OU CHIMIQUES

Jeudi 2 mai : 8 h - 12 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont autorisées

Le sujet est composé de trois parties partiellement indépendantes.

Modélisation du mouvement d'une plateforme en mer

On s'intéresse la résolution d'équations du mouvement dans une approche classique de la mécanique afin d'étudier le mouvement d'une plateforme en mer. Le modèle envisagé est un système un degré de liberté considéré comme oscillateur harmonique : une masse est reliée un ressort, avec ou sans amortissement, et peut être soumise une excitation externe.

La résolution est tout d'abord abordée de façon analytique puis de façon numérique, avant enfin de comparer les résultats obtenus.

Les résolutions analytiques et numériques sont largement indépendantes.

Dans la suite de l'énoncé, toutes les grandeurs vectorielles sont indiquées en gras.

Un aide-mémoire sur `numpy` est donné en Annexe.

On considère le mouvement d'une plateforme en mer soumise à un courant marin. Sa partie supérieure de masse $m = 110$ tonnes est considérée comme rigide et le mouvement principal de la plateforme a lieu suivant x (**figure 1(a)**).

Afin d'étudier le mouvement de cette plateforme, on la représente par une masse m , liée à un ressort de constante de raideur k et à un amortisseur de constante d'amortissement γ , pouvant subir une excitation externe de force \mathbf{F}_{exc} , et se déplaçant sur un support (**figure 1(b)**). Le ressort représente la rigidité de l'ensemble du support de la plateforme. L'amortisseur permet de prendre en compte l'effet de l'eau environnante et la force d'excitation externe celui des vagues qui frappent périodiquement la plateforme. La masse est supposée se déplacer selon une seule direction parallèle à l'axe Ox en fonction du temps t .

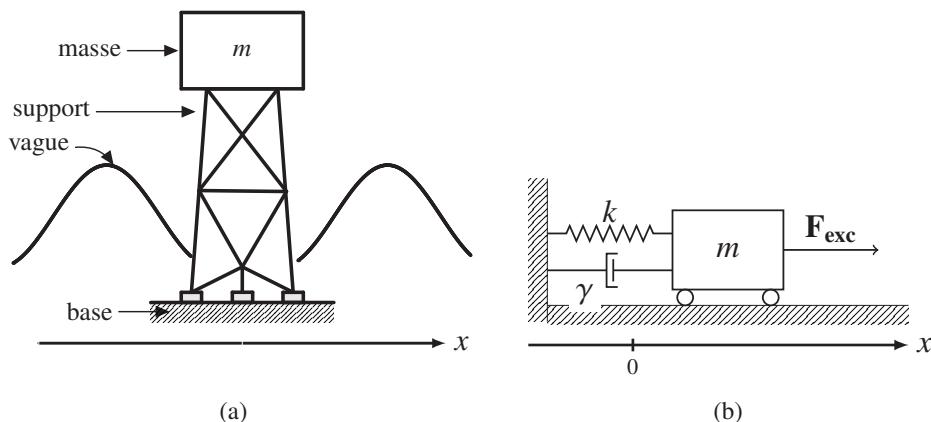


Figure 1 – (a) Plateforme en mer soumise aux vagues marines, (b) système masse (m), ressort (k), amortisseur (γ) et excitation externe (\mathbf{F}_{exc})

Les projections sur l'axe Ox de la position, de la vitesse et de l'accélération de la masse en fonction du temps sont notées respectivement $x(t)$, $\dot{x}(t)$ et $\ddot{x}(t)$. La force totale \mathbf{F}_{tot} agissant sur la masse correspond à la réaction normale \mathbf{R}_N de la base horizontale, à la force de frottement \mathbf{F}_d , à la force de rappel \mathbf{F}_k du ressort, au poids \mathbf{P} de la masse et à la force \mathbf{F}_{exc} d'excitation externe. La position d'équilibre de la masse sera choisie à $x = 0$. En l'absence d'action de l'amortisseur, la masse se déplace sur la base horizontale sans frottements.

Partie I - Résolution analytique et détermination des paramètres pour la modélisation

- Q1.** En effectuant une projection sur l'axe Ox, montrer que \mathbf{P} et \mathbf{R}_N n'interviennent pas dans le bilan des forces.

I.1 - Ressort sans amortissement et sans excitation

- Q2.** Démontrer que l'équation du mouvement de la masse correspond à l'équation différentielle du second ordre suivante :

$$m\ddot{x} + kx = 0. \quad (1)$$

- Q3.** La solution de cette équation prend la forme générale suivante

$$x(t) = A_0 \sin(\omega_0 t) + B_0 \cos(\omega_0 t) \quad (2)$$

avec A_0 et B_0 deux coefficients réels. Exprimer ω_0 en fonction des grandeurs caractéristiques du système et donner sa signification physique. De plus, en remarquant qu'à $t = 0$: $x(t) = x_0$ et $\dot{x}(t) = \dot{x}_0$, déterminer les expressions de A_0 et de B_0 en fonction de x_0 , \dot{x}_0 et de ω_0 .

- Q4.** On cherche à reformuler l'équation précédente sous une forme plus compacte du type :

$$x(t) = R_0 \cos(\omega_0 t - \phi_0). \quad (3)$$

Donner les expressions de R_0 et de ϕ_0 en fonction de x_0 , \dot{x}_0 et de ω_0 .

- Q5.** Représenter qualitativement $x(t)$ en fonction de t et indiquer sur le tracé R_0 , x_0 et $2\pi/\omega_0$.

- Q6.** En utilisant les expressions des énergies cinétique $K(t)$ et potentielle $U(t)$ du système, montrer que l'énergie totale $E(t)$ du système est alors :

$$E(t) = \frac{kR_0^2}{2}. \quad (4)$$

Justifier le résultat obtenu.

- Q7.** Représenter qualitativement $E(t)$, $K(t)$ et $U(t)$ en fonction de t .

I.2 - Ressort avec amortissement et sans excitation

- Q8.** La force de frottement que l'amortisseur exerce sur la masse est considérée comme linéaire, c'est-à-dire proportionnelle au vecteur vitesse \mathbf{v} de celle-ci : $\mathbf{F}_d = -\gamma \mathbf{v}$, avec γ constante d'amortissement (> 0). En considérant une projection sur l'axe Ox, démontrer que la position de la masse en fonction du temps suit l'équation du mouvement ci-après

$$\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = 0 \quad (5)$$

avec ω_0 défini en question **Q3** et ζ à exprimer en fonction de γ , k et m .

Q9. Dans le cas où $\zeta < 1$, $x(t)$ prend la forme suivante :

$$x(t) = e^{-\zeta\omega_0 t} (A_d \cos(\omega_d t) + B_d \sin(\omega_d t)). \quad (6)$$

Déterminer les deux coefficients réels A_d et B_d en fonction de x_0 , \dot{x}_0 , ζ , ω_0 et $\omega_d = \omega_0 \cdot \sqrt{1 - \zeta^2}$. On utilisera pour cela les mêmes conditions initiales que celles utilisées en question **Q3**.

Q10. Montrer alors que l'on peut obtenir une forme du type

$$x(t) = R_d e^{-\zeta\omega_0 t} \cos(\omega_d t - \phi_d) \quad (7)$$

avec R_d et ϕ_d à préciser.

Q11. Représenter qualitativement $x(t)$ en fonction de t et indiquer sur le tracé $R_d e^{-\zeta\omega_0 t}$, x_0 et $2\pi/\omega_d$.

Q12. Donner l'expression de $E(t)$ et commenter les cas où $\zeta = 0$ et $\zeta = 1$.

Q13. Montrer de façon simple que E est une fonction décroissante de t . quoi cela est-il dû ?

Q14. On envisage deux temps successifs t_1 et t_2 pour lesquels les déplacements sont x_1 et x_2 , tels que $t_2 > t_1$ et $t_2 - t_1 = \tau_d$, avec τ_d : période des oscillations amorties. En utilisant l'équation (7) et en considérant que $\zeta \ll 1$, montrer que :

$$\ln(x_1/x_2) \approx 2\pi\zeta. \quad (8)$$

Q15. Le relevé du déplacement horizontal de la plateforme en fonction du temps est représenté en **figure 2**.

En utilisant les deux points qui sont indiqués sur la figure, déterminer k , ζ et γ . Comment ce tracé serait modifié en fonction de la valeur de ζ ?

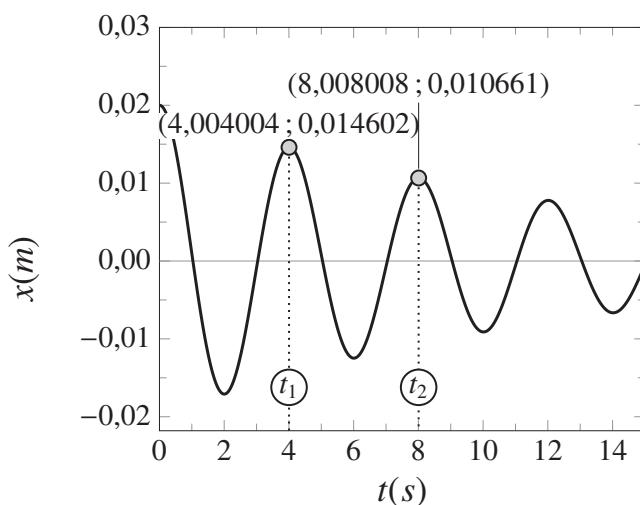


Figure 2 – Relevé du déplacement horizontal x (en m) de la plateforme de masse $m = 110$ tonnes en fonction du temps t (en s). Les deux temps t_1 et t_2 mentionnés en question **Q14** sont indiqués

I.3 - Ressort avec amortissement et avec excitation

On envisage enfin le cas où le système est soumis à la fois aux effets d'amortissement et d'excitation.

On se limite ici à la réponse à une excitation harmonique sinusoïdale de fréquence ω produite par une force extérieure au système

$$\mathbf{F}_{\text{exc}}(t) = F_0 \cos(\omega t) \mathbf{e}_x \quad (9)$$

avec \mathbf{e}_x vecteur unitaire sur l'axe Ox et on se place dans le cas traité précédemment pour l'étude de l'amortisseur, c'est-à-dire $\zeta < 1$ (**I.2**).

On admet de plus dans ce qui suit que la réponse du système dans le cas où amortisseur et excitation sont pris en compte peut s'écrire comme somme de la solution donnée par l'équation (6) et de la contribution due à l'excitation :

$$x_{\text{exc}}(t) = X \cos(\omega t - \phi). \quad (10)$$

Q16. Montrer que l'équation différentielle caractérisant le système devient alors :

$$\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = \frac{F_0}{m} \cos(\omega t). \quad (11)$$

Q17. En utilisant l'équation (10) et en privilégiant une représentation complexe, vérifier que :

$$\left\{ \begin{array}{l} X = \frac{F_0}{m} \cdot \frac{1}{\sqrt{(\omega_0^2 - \omega^2)^2 + (2\zeta\omega_0\omega)^2}} \\ \tan \phi = \frac{2\zeta\omega_0\omega}{\omega_0^2 - \omega^2} \end{array} \right. . \quad (12)$$

Q18. Exprimer la grandeur $M = \frac{X}{F_0/k}$ en fonction de $r = \omega/\omega_0$ et expliciter le sens physique de M .

Q19. Trouver la condition sur r puis sur ω pour laquelle M est maximale.

Q20. Si l'on considère une période moyenne des vagues en mer de 8 s, que peut-on conclure sur le mouvement de la plateforme ?

Partie II - Modélisation : codage

On souhaite maintenant obtenir $x(t)$ et $E(t)$ de façon numérique et comparer les résultats obtenus à ceux fournis par les solutions analytiques précédentes pour $x(t)$. On rappelle que $x(t)$ et $E(t)$ représentent respectivement la position de la masse et l'énergie mécanique totale en fonction du temps.

Pour cela, le temps est discrétisé en N points $t = 0, t, 2t, \dots, (N-1)t$ avec un pas de temps constant t . Les $N-1$ pas sont effectués pendant la simulation de durée totale t_{\max} . On note respectivement x_n, v_n, a_n, E_n et F_n les valeurs de $x(t), \dot{x}(t), \ddot{x}(t), E(t)$ et $F_{\text{exc}}(t)$ à $t = n t$.

Chaque pas, les équations du mouvement reliant x_{n+1} et v_{n+1} à x_n et v_n sont utilisées afin d'obtenir les valeurs de x, v et E . Les conditions initiales x_0 et v_0 sont connues et permettent de démarrer le processus d'intégration numérique. Deux algorithmes distincts (Euler et Leapfrog) vont être utilisés dans la suite.

Pour l'écriture du code, on se place dans le cas le plus général, c'est-à-dire avec amortissement et excitation harmonique externe. Les variables et tableaux suivants sont notamment choisis :

N	nombre de points sur l'axe des temps utilisés pendant toute la simulation
$t[]$	tableau des temps (s), de dimension N
$x[]$	tableau des positions (m), de dimension N
$v[]$	tableau des vitesses (m.s^{-1}), de dimension N
$E[]$	tableau des énergies totales (J), de dimension N
$F[]$	tableau des forces d'excitation (N), de dimension N
dt	pas de temps (s)
t_{\max}	temps total de la simulation (s)
k	constante de raideur du ressort (N.m^{-1})
m	masse du système (kg)
ω_0	$\omega_0 (\text{s}^{-1})$
ζ	ζ (sans unité)

Tableau 1 – Principaux tableaux et principales variables utilisés pour la résolution numérique

On rappelle qu'un aide-mémoire sur `numpy` est fourni en **Annexe**, page 10.

Q21. Écrire les lignes de code permettant de définir l'entier N . On suppose t_{\max} et t connus et fixés par l'utilisateur en début de code.

Q22. Écrire alors l'instruction permettant de définir le tableau t qui contient toutes les valeurs de t telles que : $0 \leq t \leq t_{\max}$. On rappelle que le temps est discrétisé en N points $t = 0, t, 2t, \dots, (N-1)t$ avec un pas de temps constant t et que $N-1$ pas sont effectués.

Q23. En effectuant des développements de Taylor de x et v tronqués à l'ordre 1, on obtient l'algorithme d'Euler, où x et v sont évalués au même temps t selon le schéma donné **figure 3**, page 7 :

$$\begin{cases} x_{n+1} \approx x_n + v_n \cdot t \\ v_{n+1} \approx v_n + a_n \cdot t \end{cases}. \quad (13)$$

Donner les expressions de x_{n+1} et v_{n+1} en fonction de x_n, v_n et F_n .

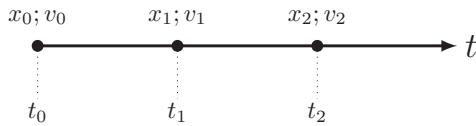


Figure 3 – Discréétisation en temps utilisée dans le cas de l'algorithme d'Euler. Les conditions initiales correspondent à $(x_0; v_0)$

Q24. Écrire la boucle en i permettant d'obtenir toutes les valeurs de $x[i+1]$, $v[i+1]$ et $E[i+1]$ où i correspond à un point sur l'axe des temps. On précisera les variables éventuellement introduites en supposant qu'elles ont été définies dans le code.

Q25. Dans l'algorithme de Leapfrog, les x sont évalués aux temps entiers, c'est-à-dire à $t = 0, t, 2t, \dots, (N-1)t$, alors que les v sont évalués à $t = -t/2, t/2, 3t/2, \dots, (N-1)t - t/2$ selon le schéma donné **figure 4**. Ainsi, pour cet algorithme, $x[i]$ représente de façon approchée la position à l'instant i t , et $v[i]$ représente de façon approchée la vitesse à l'instant $(i - 1/2)$ t .

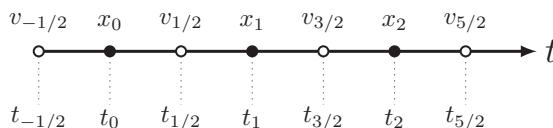


Figure 4 – Discréétisation en temps utilisée dans le cas de l'algorithme de Leapfrog. Les conditions initiales correspondent à $(x_0; v_{-1/2})$

Pour le système considéré, montrer alors que x_{n+1} et $v_{n+1/2}$ prennent les formes suivantes :

$$\begin{cases} x_{n+1} \approx x_n + v_{n+1/2} \cdot t \\ v_{n+1/2} \approx -\frac{\omega_0^2}{1 + \zeta\omega_0} \frac{t}{t} x_n + \frac{1 - \zeta\omega_0}{1 + \zeta\omega_0} \frac{t}{t} v_{n-1/2} + \frac{F_n}{m(1 + \zeta\omega_0)} t. \end{cases} \quad (14)$$

Q26. Quel problème pose l'évaluation de $E[i+1]$?

Dans la suite, on préfèrera ainsi évaluer $E[i]$.

Q27. Comment obtenir $E[i]$ à partir de $x[i]$ et $v[i]$?

Q28. En introduisant deux variables `fac1` et `fac2` correspondant respectivement à $1 - \zeta\omega_0 t$ et $\frac{1}{1 + \zeta\omega_0 t}$, écrire la boucle en i permettant d'obtenir toutes les valeurs de $x[i+1]$ et $v[i+1]$.

Q29. Compléter la boucle de la question **Q28** avec le calcul du terme d'énergie totale.

Q30. Écrire une fonction `integration(F)` qui prend en argument le tableau `F[]` des forces d'excitation et renvoie les tableaux `x[]`, `v[]` et `E[]` complétés.

On introduira pour cela une variable `algo` supposée définie en global, permettant d'appliquer l'algorithme d'Euler si `algo==0` ou de Leapfrog si `algo==1`.

On considèrera également le cas $t = 0$.

Q31. Écrire une fonction `force(f, t, w)` qui prend en argument F_0 , t , et ω de l'équation (9) et retourne la valeur de \mathbf{F}_{exc} pour un temps t donné.

Q32. Écrire une fonction `force_exc()` qui complète et retourne le tableau `F[]` des forces d'excitation en fonction du booléen `exc` défini globalement valant `True` si une excitation est appliquée au système, `False` sinon.

Dans le cas où `exc==True`, on appellera la fonction `force` définie précédemment en question **Q31**. Dans le cas où `exc==False`, on prendra alors : $F[i]=0, \forall i$.

Q33. Donner alors les lignes de code permettant de réaliser la simulation numérique à partir des fonctions précédentes.

Q34. On souhaite désormais estimer la qualité des résultats numériques par rapport aux données analytiques de référence. Écrire une fonction `ema(d, dref)` qui calcule l'erreur maximale absolue entre un jeu de données numériques `d` et analytiques `dref`. On supposera que les tableaux `d` et `dref` sont de même dimension n .

Q35. Pour le problème particulier qui nous intéresse, si l'on souhaite appliquer cette fonction aux tableaux contenant les données numériques et analytiques pour E , quel est l'indice maximal de ces tableaux à considérer ?

Réécrire alors la fonction `ema` pour qu'elle soit applicable aux deux algorithmes considérés.

Partie III - Modélisation : analyse des résultats d'un cas simple

Toutes les données numériques suivantes ont été obtenues avec : $t_{\max} = 10$ s, $m = 110$ tonnes, $x_0 = 0,02$ m et $v_0 = 0$ m.s⁻¹ et la valeur de k obtenue en question **Q15**, dans le cas d'un système sans amortissement et sans excitation externe. Le **tableau 2** en page 9 présente les erreurs maximales absolues (EMA) calculées avec la fonction `ema` des énergies obtenues numériquement par rapport à celles obtenues analytiquement, pour les deux algorithmes envisagés et divers pas de temps.

Q36. Justifier l'ordre de grandeur des t considérés du **tableau 2** pour la discréttisation en temps utilisée.

Q37. En utilisant les données numériques du **tableau 2** donner l'ordre approximatif de l'erreur globale sur E des deux algorithmes considérés. Justifier votre réponse.

Q38. Dans le cas simple de l'algorithme d'Euler, comment augmente E lorsqu'on passe de t_n à t_{n+1} ? Relier alors ce résultat aux données du **tableau 2**.

t (s)	EMA	
	Euler	Leapfrog
0,050	128,0203160	0,0837314
0,010	15,1373529	0,0033484
0,005	7,1159053	0,0008371
0,001	1,3556230	0,0000335

Tableau 2 – EMA obtenues pour E (en J) par rapport à la valeur analytique, pour différents pas de temps t , dans le cas où il n'y a pas d'amortissement et d'excitation externe

Une propriété importante que devrait vérifier un algorithme d'intégration est la réversibilité dans le temps : en partant des positions et vitesses d'un temps $t + t$ et en appliquant un pas de temps $-t$, un algorithme réversible en temps devrait redonner les positions et vitesses du temps t . En pratique, en partant d'un couple $(x_n; v_n)$, on applique donc tout d'abord un pas $+t$ pour déterminer $(x_{n+1}; v_{n+1})$, puis on applique un pas $-t$ afin d'obtenir $(\bar{x}_n; \bar{v}_n)$. Si $(\bar{x}_n; \bar{v}_n) = (x_n; v_n)$, alors l'algorithme est dit réversible en temps.

Q39. Pourquoi s'agit-il d'une propriété importante à vérifier pour le problème considéré ?

Q40. Donner l'expression de \bar{x}_n en fonction de x_n pour les deux algorithmes considérés. Peut-on déjà conclure sur la réversibilité en temps de chacun de ces algorithmes ?

Q41. Donner alors l'expression de \bar{v}_n en fonction de v_n lorsque nécessaire et conclure sur la réversibilité en temps.

Q42. On s'intéresse enfin à une simulation de plus grande durée ($t_{\max} = 60$ s). La **figure 5** donne l'évolution de l'erreur absolue sur x entre données numériques et analytiques ($|e_x|$) pour les deux algorithmes choisis. Que peut-on mettre en évidence sur cette figure ?

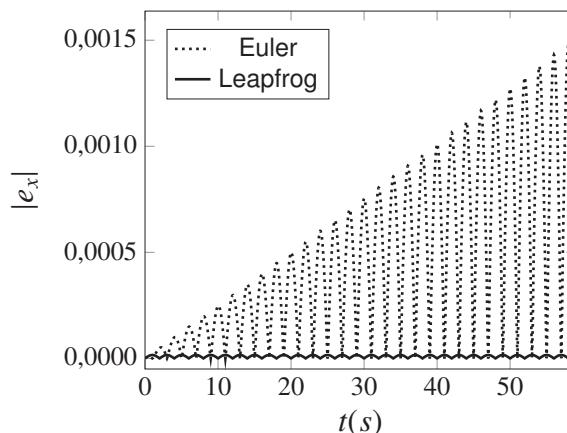


Figure 5 – Erreurs absolues calculées sur la position de la masse à $t = 0,001$ s, pour les deux algorithmes considérés

Q43. Conclure sur les avantages et les inconvénients de chacun de ces algorithmes et sur leur adéquation pour le traitement numérique de ce problème dans le cas où on envisage une simulation de plusieurs heures.

ANNEXE

Aide-mémoire sur numpy

Les bibliothèques sont importées de la façon suivante :

```
from math import *
import numpy as np
```

La création d'un tableau numpy `tab` à une dimension possédant n éléments, tous initialisés à 0, est réalisée à l'aide de l'instruction :

```
>>> tab=np.zeros(n)
```

Celle d'un tableau numpy `tab` à une dimension possédant n éléments, uniformément répartis entre deux valeurs `debut` et `fin`, se fait avec :

```
>>> debut=0; fin=10; n=5
>>> tab=np.linspace(debut,fin,n)
>>> print tab
array([ 0.0  2.5  5.0  7.5 10.0 ])
```

L'accès à un élément du tableau `tab` (en lecture ou en écriture) se fait par `tab[i]`, la numérotation des indices se faisant à partir de 0 :

```
>>> tab=np.zeros(4)
[ 0.0  0.0  0.0  0.0 ]
>>> tab[1]=2; tab[2]=6; print tab
[ 0.0  2.0  6.0  0.0 ]
```

La sélection de l'ensemble des j premiers éléments du tableau `tab` est possible avec :

```
>>> print tab[:3]
[ 0.0  2.0  6.0 ]
```

Le maximum des éléments d'un tableau `tab` s'obtient avec :

```
>>> np.max(tab)
```

FIN

10/10



Epreuve d’Informatique et Modélisation de Systèmes Physiques

Durée 4 h

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

L’usage de calculatrices est interdit.

La **présentation**, la lisibilité, l’orthographe, la qualité de la **rédaction, la clarté et la précision** des raisonnements entreront pour une **part importante dans l’appréciation des copies**. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

L’épreuve comporte différentes parties permettant de mobiliser les compétences du candidat en informatique et en modélisation.

- La partie modélisation est distribuée au début et à la fin du sujet (I-1, I-2 et I-8). Il est conseillé de ne pas y consacrer plus d'une heure et demie.
- La partie informatique est développée au cœur du sujet (de I-3 à I-7) et sa durée de traitement conseillée est de deux heures et demie.

En dernière partie du sujet (II), vous trouverez un aide-mémoire *Python* et les consignes générales à respecter pour les codes exigés.

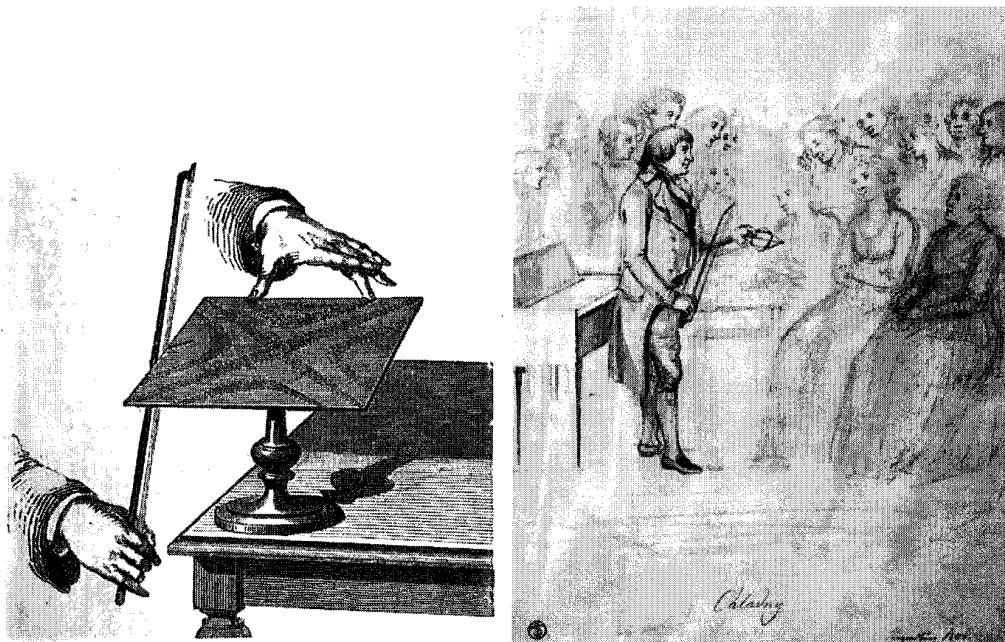
Tout code fourni en réponse à une question doit être commenté de façon claire. Les commentaires sont à placer dans le code, ou éventuellement dans la brève introduction qui le précède.

Le poids relatif des parties est indiqué dans le titre de chaque paragraphe.

I Figures de Chladni

Introduction historique

En 1787, le physicien et musicien, Ernst Florence Friedrich Chladni de Wittenberg, alors qu'il procédait à de nombreuses expériences, fit une intéressante découverte. Il constata que lorsqu'il excitait une plaque de métal avec l'archet de son violon, il la faisait vibrer et il pouvait produire des sons de distinctes tonalités selon l'endroit où il touchait la plaque. La plaque métallique était fixée en son centre, il eut l'idée d'y disperser de la poussière. Lors des phases vibratoires, pour chaque tonalité, la poussière s'arrangeait et se distribuait selon les lignes nodales des vibrations, de magnifiques figures géométriques apparaissaient alors. Expérimentateur chevronné, il prit soin de cataloguer et recenser ces différentes figures, après s'être assuré de leur caractère reproductif.



Dispositif et démonstration de Chladni

Ces figures, désormais dénommées figures de Chladni, en hommage à leur découvreur, attirèrent l'attention de nombreuses personnalités de l'époque. Scientifiques et puissants se pressaient aux nombreuses démonstrations du musicien pour admirer le phénomène. Sa compréhension échappait toutefois à l'entendement des contemporains de Chladni.

L'empereur Napoléon Bonaparte fut enthousiasmé par ces démonstrations permettant d'entendre et "voir" le son. En 1809, il invita l'académie des sciences à proposer un prix pour expliquer le phénomène et il finança la traduction en français du traité majeur d'acoustique de Chladni.

La première à proposer une explication pour ces observations fut la mathématicienne Sophie Germain dans une correspondance privée à partir de 1811. Elle publia quelques années plus tard un résumé de ces études qui constitua le premier modèle mathématique pour la déformation d'une plaque sous une contrainte de force extérieure, son travail "Recherche sur la Théorie des surfaces élastiques" sera couronné par l'Académie des Sciences en 1816. Lagrange et Poisson corrigèrent et améliorèrent ce premier modèle. Il fallut toutefois attendre l'intervention de Kirchoff pour aboutir à une modélisation permettant d'approcher de façon satisfaisante le comportement d'une plaque. Il en profita pour résoudre le problème des figures de Chladni sur une plaque circulaire où les nombreuses symétries permettent de réduire la complexité des solutions.

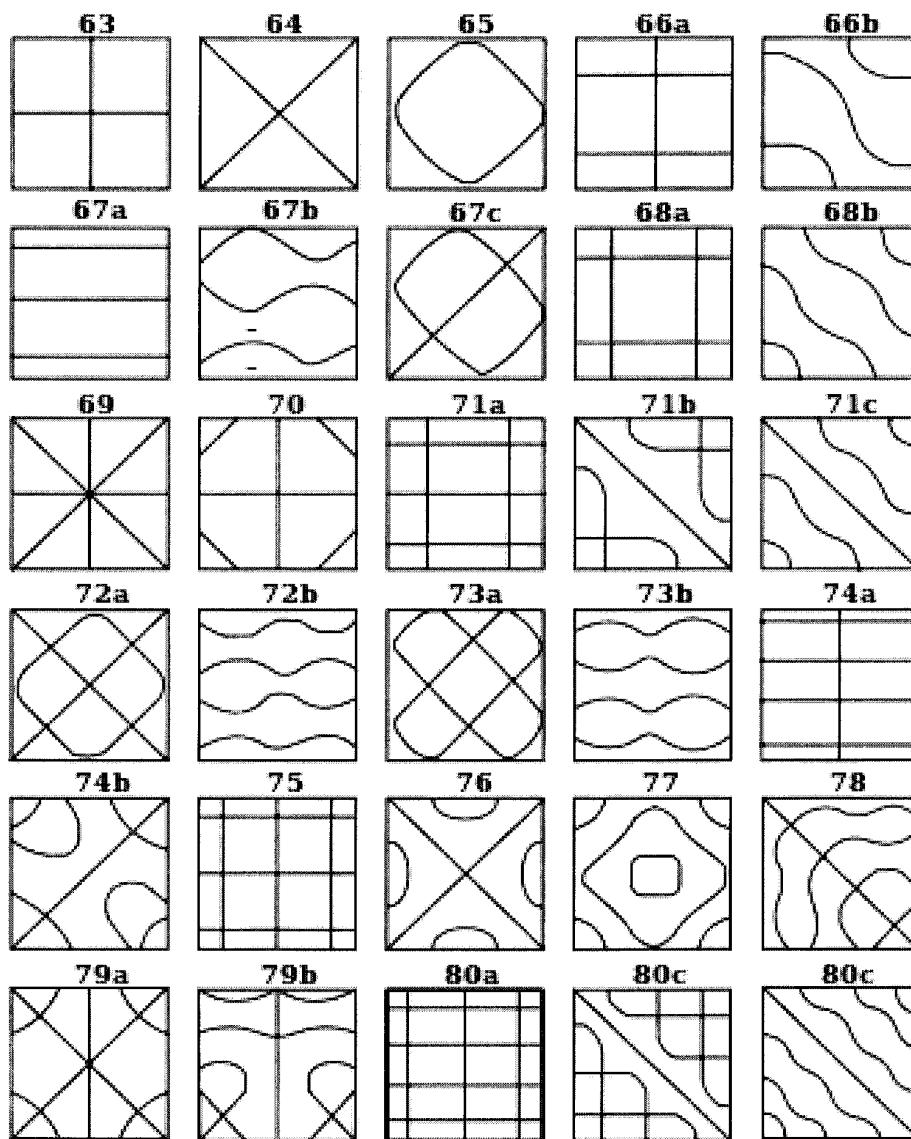


FIGURE 1 – Figures de Chladni

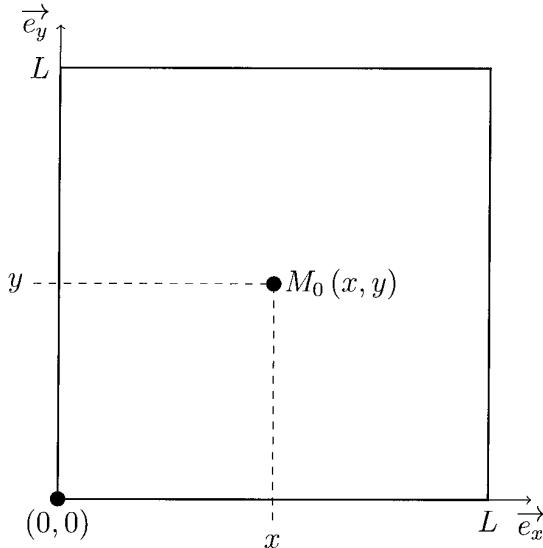
À l'aube du vingtième siècle, l'expert sur la théorie des sons, John William Strut qui passa à la postérité en tant que Lord Rayleigh résuma la situation dans son traité majeur "La Théorie du Son" : *Le problème de la plaque rectangulaire dont les bords sont libres est d'une extrême difficulté et a pour l'essentiel résisté à toutes les tentatives de résolution.* Ce fut la spectaculaire invention de Walther Ritz qui permit en 1909 d'effectuer le premier calcul précis des vibrations d'une plaque carrée.

Nous nous proposons dans le problème suivant de reprendre en partie la démarche d'analyse du problème et de mettre en œuvre la conception du code en permettant une résolution numérique.

I.1 Introduction du modèle physique ($\sim 10\%$)

Nous considérons une plaque métallique carrée de côté L , elle est d'épaisseur e , de module de Young E et sa masse volumique vaut ρ .

Pour suivre les éventuelles déformations de la plaque, nous utilisons un repère cartésien $\mathcal{R}(O, \vec{e}_x, \vec{e}_y, \vec{e}_z)$. Au repos, la plaque se situe dans un plan horizontal de cote $z = 0$ et chaque point de la plaque au repos est repéré par $M_0(x, y)$.



Repérage d'un point sur la plaque

Sous l'action d'une sollicitation extérieure, ou d'une contrainte, la plaque se déforme, chacun des points initialement en $M_0(x, y)$ est alors décrit par le point $M(x', y', z, t)$. M est une fonction du champ initial de position M_0 et du temps t .

Nous supposons que si les sollicitations sont modérées, nous aurons :

$$\forall t \quad x' = x \quad \text{et} \quad y' = y$$

Dans le cadre de cette approximation, le point M est à tout instant à la verticale de M_0 et sa seule variable d'espace indéterminée est son altitude z . Nous pouvons, dans ce cas, considérer que la position de M est décrite par la fonction :

$$z = f(x, y, t)$$

z est l'amplitude de vibration et $f(x, y, t)$ est la fonction d'onde de vibration.

Une analyse du problème utilisant une approche de déformation élastique, permet de considérer que la fonction d'onde f vérifie l'équation :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{1}{c^2} \cdot \frac{\partial^2 f}{\partial t^2} \quad (1)$$

où $c > 0$ est la célérité de l'onde.

- 1°) Quelle est le nom de cette équation d'onde ?
 Quelles sont ces principales propriétés ?
 Citez au moins deux situations physiques distinctes pour lesquelles il existe une équation analogue.
- 2°) Pensez-vous que cette équation soit le reflet d'un modèle physique autorisant des pertes d'énergie ? Justifiez votre réponse.
- 3°) Par analyse dimensionnelle, proposez une expression de c cohérente vis à vis des paramètres physiques de la plaque.
- 4°) Application numérique : masse volumique $\rho = 2,70 \cdot 10^3 \text{ kg m}^{-3}$, épaisseur $e = 1,00 \text{ mm}$ module de Young $E = 69 \text{ GPa}$ et la longueur du côté de la plaque $L = 25,0 \text{ cm}$.
 Calculer c .
- 5°) Soit $f_L(x, y, t)$ une solution de l'équation (1) pour une plaque de côté L , $f_L(x, y, t)$ possède une caractéristique temporelle τ .
 Nous considérons, dans cette seule question, une plaque faite du même matériau, possédant la même épaisseur, et soumises aux mêmes types de conditions aux limites, mais de côté $L' = \gamma \cdot L$.
- a) Déterminer, parmi (α, β) , la valeur du coefficient α qui permet à la fonction $g(x, y, t) = f_L(\alpha x, \alpha y, \beta t)$ d'être solution de l'équation (1) pour cette nouvelle plaque.
 - b) En déduire la caractéristique temporelle τ' correspondante en fonction de γ .

I.2 Modes de vibration ($\sim 5\%$)

En règle générale, les vibrations d'une plaque sont complexes et se décomposent en superposition de *modes*. Chacun de ces modes correspond à une vibration monochromatique (*d'une seule fréquence*).

- 6°) Pour résoudre l'équation (1) en $z(x, y, t)$, nous allons commencer par rechercher des solutions à variables séparables. C'est à dire des solutions qui seront de la forme :

$$z = f(x, y, t) = u(x, y) \times h(t)$$

- a) En déduire les équations différentielles que doivent vérifier les fonctions $u(x, y)$ et $h(t)$.
- b) Pourquoi les solutions acceptables pour $h(t)$ sont-elles seulement sinusoïdales ?
- c) Les solutions $h(t)$ sont mises sous la forme $h(t) = h_0 \exp(i\omega t)$, exprimez alors l'équation vérifiée par $u(x, y)$. Différerait-elle si nous avions privilégié la forme $h(t) = A \cdot \cos(\omega t) + B \cdot \sin(\omega t)$?

I.3 Simulation numérique temporelle ($\sim 20\%$)

L'équation différentielle que doit vérifier la fonction temporelle peut s'écrire après adimensionnement sous la forme :

$$\frac{d^2h}{dt^2}(t) = -h(t). \quad (2)$$

C'est une équation différentielle linéaire d'ordre deux classique appelée équation harmonique. Nous l'utiliserons pour tester la pertinence de certaines méthodes numériques.

7°) Méthode d'Euler

- a) Rappeler, en quelques lignes concises, le principe de la méthode d'Euler pour résoudre une équation différentielle.
- b) Mettre l'équation précédente sous la forme d'un système différentiel linéaire du premier ordre et le présenter sous forme matricielle en caractérisant la matrice A et en exploitant le vecteur $X(t)$ tel que :

$$\frac{dX}{dt}(t) = AX(t) \quad \text{avec} \quad X(t) = \begin{pmatrix} h(t) \\ h'(t) \end{pmatrix}$$

- c) A partir de cette forme matricielle, introduire le pas de **discrétisation temporel** τ et mettre en œuvre la méthode d'Euler pour obtenir une équation aux différences correspondant au système initial.

Vous pourrez noter de façon réduite $X(t_n) = X[n]$

8°) Méthode d'Euler à droite.

Pour obtenir $X(t + \tau)$, nous pouvons procéder de la façon suivante :

$$X(t_n + \tau) = X(t_n) + \int_{t_n}^{t_n + \tau} \frac{dX(t)}{dt} dt \approx X(t_n) + \tau \cdot \frac{dX(t_n + \tau)}{dt}$$

C'est la méthode d'Euler dite à *droite*, ou *implicite*, la version classique étant dite à *gauche*, ou *explicite*.

- a) Exprimer l'équation liant dans ce cas $X[n + 1]$ à $X[n]$.

- b) Le code ci-après correspond à l'implémentation de la méthode d'Euler à gauche.
Le modifier pour implémenter la méthode d'Euler à droite.

```

1 # Constantes paramétriques
2 npoints=2**10 # index maximal des tableaux
3 tporte=5*2*np.pi # durée d'affichage
4 X0=[1.,0] # conditions initiales
5
6 # Initialisation des tableaux
7 t=np.linspace(0,tporte,npoints+1) # temps
8 tau=t[1]-t[0]
9 X=np.zeros((2,npoints+1),dtype=float) # h et h'
10
11 # Algorithme d'Euler
12 X[:,0]=X0 #initialisation
13 M=np.array([[1,tau],[-tau,1]]) # matrice de l'équation
14 for i in range(npoints): # Calcul itératif
    X[:,i+1]=M.dot(X[:,i])
15

```

- c) La méthode d'Euler à droite est elle plus pertinente que celle à gauche ?

9°) Représentations graphiques

- a) A partir du code donné à la question précédente, mettre en place les instructions pour obtenir l'affichage correspondant aux figures présentées ci-après.
Vous veillerez à la présence des légendes des courbes, des axes et des titres.

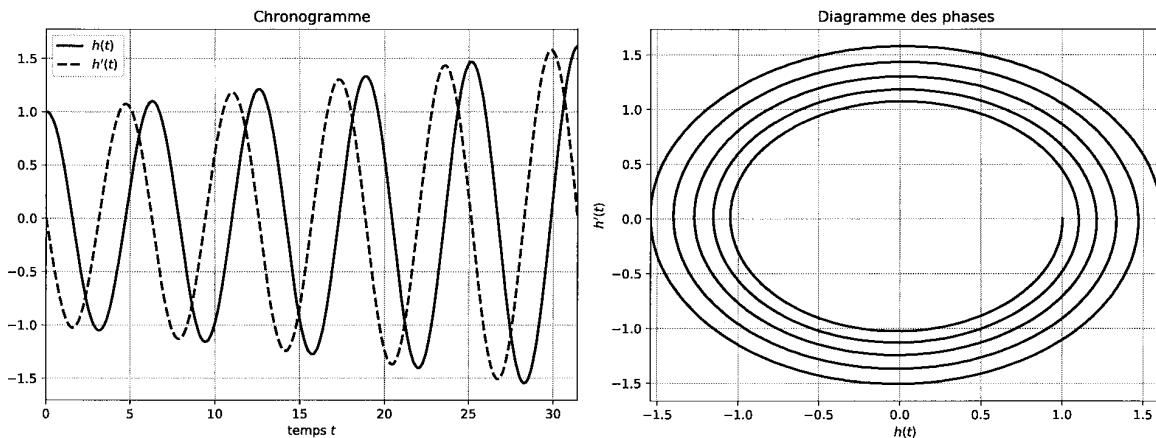


FIGURE 2 – Méthode d'Euler

- b) Commentez les courbes obtenues et corrélez leurs allures aux propriétés de la méthode d'Euler et à l'équation (2).

- c) Sans modifier le principe de l'algorithme utilisé, nous pouvons néanmoins obtenir les courbes ci-après. Quelle est, selon vous, la modification apportée au code précédent ? Et quel en est l'inconvénient ?

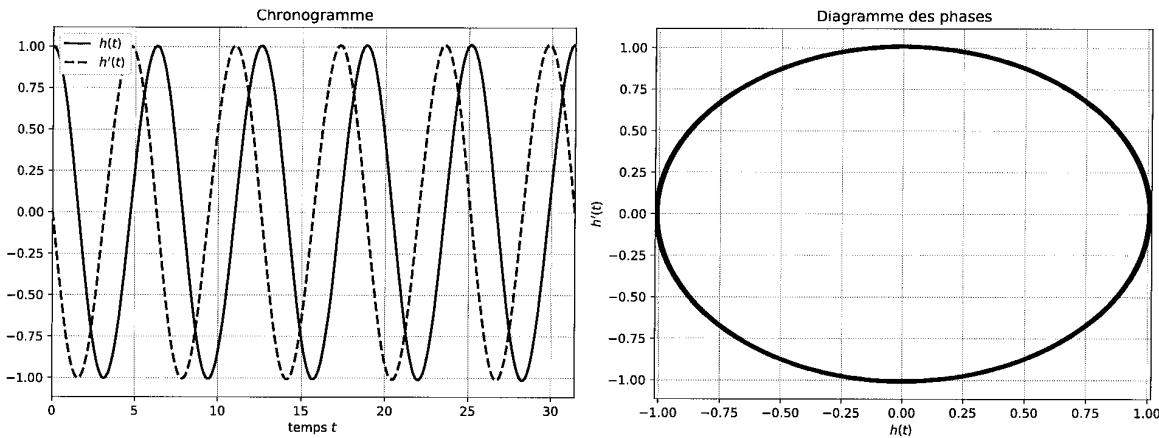


FIGURE 3 – Méthode d’Euler - version alternative

- 10°)** Méthode de Runge-Kutta d’ordre deux (*aucune connaissance préalable n’est nécessaire*). $X(t + \tau)$ s’obtient exactement à partir de $X(t)$ avec la classique relation intégrale :

$$X(t_n + \tau) = X(t_n) + \int_{t_n}^{t_n + \tau} \frac{dX(t)}{dt} dt$$

Nous pouvons obtenir une version approchée de cette relation en utilisant l’approximation :

$$X(t_n + \tau) \approx X(t_n) + \frac{\tau}{2} \cdot \left(\frac{dX(t_n + \tau)}{dt} + \frac{dX(t_n)}{dt} \right)$$

- a) Quel est le nom de la méthode employée pour approximer l’intégrale ? Qualitativement, pourquoi est-elle préférable aux méthodes précédentes ?
- b) L’analyse est faite sur la durée $T = N\tau$.

La valeur de la dérivée de X au temps $t_n + \tau$ demande la mise en œuvre d’une méthode de calcul implicite, nous l’éviterons en estimant la valeur de X au temps $t_n + \tau$ par la méthode d’Euler, c’est le principe de la méthode de Runge-Kutta. L’algorithme de calcul est alors le suivant :

$$K_1 = \frac{dX(t_n)}{dt} = A \cdot X[n] \quad K_2 = A \cdot (X[n] + \tau K_1)$$

$$X[n + 1] = X[n] + \frac{\tau}{2} \cdot (K_1 + K_2)$$

Reprenez le code écrit pour la méthode d’Euler et adaptez le pour mettre en œuvre la méthode de Runge-Kutta.

- c) La simulation numérique dans des conditions identiques à la méthode d'Euler nous fournit les graphes ci-après. Ils semblent correspondre aux attendus usuels.
 Peut-on conclure à l'adéquation de la méthode de Runge-Kutta d'ordre deux ? Argumentez votre réponse.

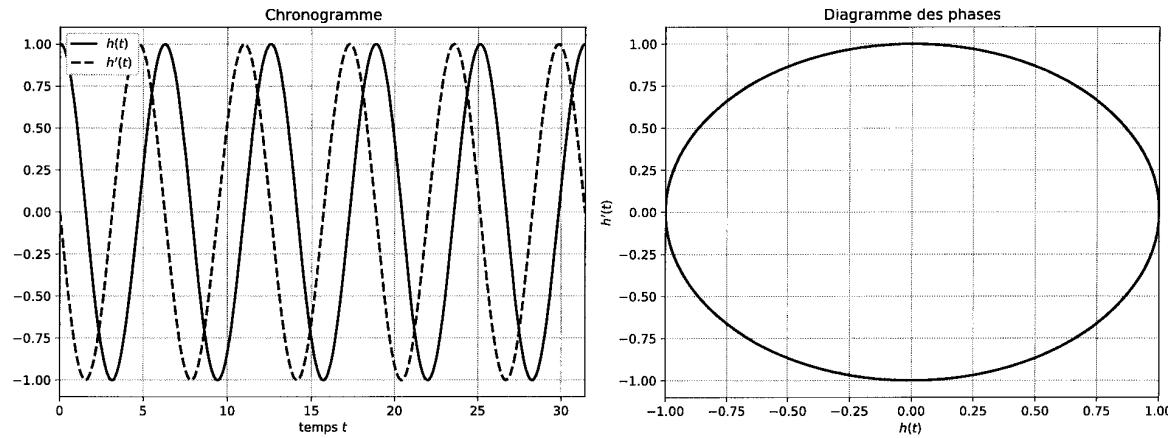


FIGURE 4 – Méthode de Runge-Kutta

11°) Méthode "stable".

Nous implémentons désormais le code ci-après :

```

1 # Constantes paramétriques
2 npoints=2**10 # index maximal des tableaux
3 tporte=100*2*np.pi # durée d'affichage
4 X0=[1.,0] # conditions initiales
5
6 # Initialisation des tableaux
7 t=np.linspace(0,tporte,npoints+1) # temps
8 tau=t[1]-t[0]
9 X=np.zeros((2,npoints+1),dtype=float) # h et h'
10
11 # Algorithme XXX
12 X[:,0]=X0 #initialisation
13 A=np.array([[0,1],[-1,-tau]]) # matrice de couplage
14 M=np.identity(2)+tau*A # matrice d'évolution
15 for i in range(npoints): # Calcul itératif
16     X[:,i+1]=M.dot(X[:,i])
17

```

Il nous permet d'obtenir les courbes de la figure (5).

- a) A partir du code déployé, donnez les équations itératives permettant la mise en œuvre de la simulation.
- b) Quel est l'ordre de cette méthode ?
 Pourquoi est elle "stable" ?

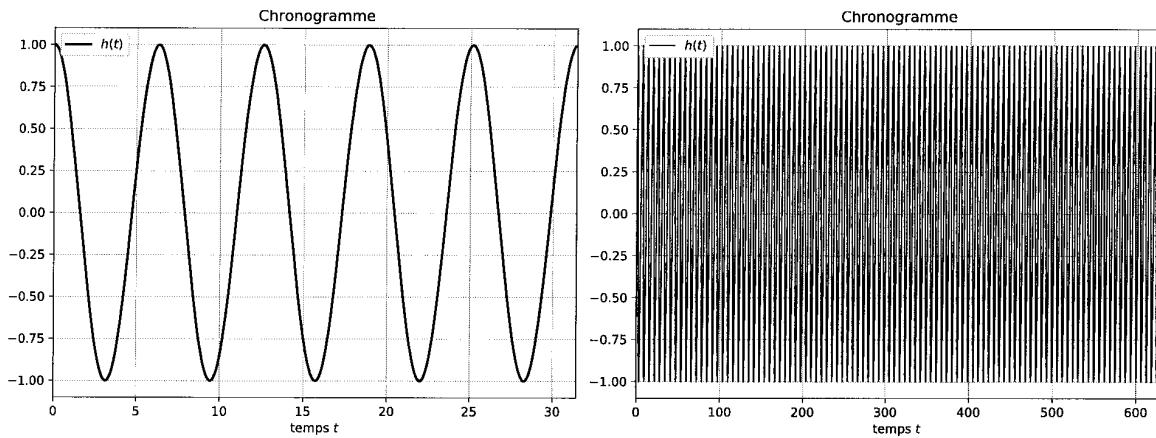


FIGURE 5 – Méthode "Stable"

I.4 Discrétisation spatiale du problème ($\sim 15\%$)

L'équation correspondant à un mode propre de pulsation ω peut s'écrire, pour $u(x, y)$ sous la forme :

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -\lambda \cdot u \quad (3)$$

où $\lambda > 0$ est la valeur propre associée au mode étudié.

Nous nous proposons de résoudre numériquement cette équation en utilisant une discréétisation et la méthode des différences finies. Nous introduisons, pour cela, un maillage de la plaque étudiée, de pas régulier, uniforme $d = \frac{L}{N}$ où N représente le nombre de segments du maillage.

12°) Équation adimensionnée

Nous allons effectuer le changement de variable ci-après pour faciliter le processus de discréétisation. Nous posons, pour une plaque de côté L :

$$X = \frac{x}{L} \times N, \quad Y = \frac{y}{L} \times N, \quad U(X, Y) = u(x, y)$$

L'équation vérifiée par $U(X, Y)$ se mettra sous la forme.

$$\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} = -\lambda' \cdot U \quad (4)$$

Déterminer l'expression de λ' en fonction de la pulsation ω .

Que devient le pas de discréétisation initial dans ce nouveau jeu de variables ?

Nous allons chercher à résoudre numériquement cette équation différentielle linéaire en la discréétisant avec le pas régulier unité. Nous découpons les axes X et Y en sous intervalles délimités par des ensembles de points définis de la manière suivante :

$$\forall i \in [0, N] \quad X_i = i$$

$$\forall j \in [0, N] \quad Y_j = j$$

La plaque sera représentée par un ensemble de points, distribués comme ci-après si $N > 4$.

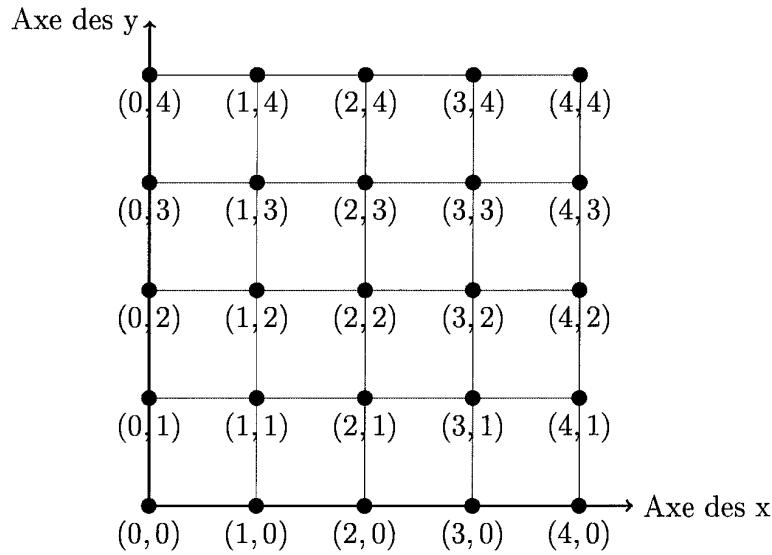


FIGURE 6 – Discréétisation de la plaque

Nous noterons, selon la convention suivante, la valeur approchée de $U(X, Y)$ correspondant au point de coordonnées (X_i, Y_j) :

$$U(X_i, Y_j) \approx U_{i,j}$$

13°) Discréétisation autour d'un point central

Nous étudions la situation d'un point du maillage loin des bords. Nous supposons que ce dernier de coordonnées (X_i, Y_j) possède ses huit plus proches voisins comme indiqué sur la figure (7).

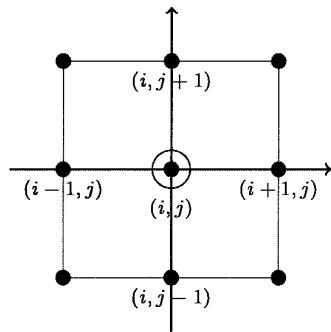


FIGURE 7 – Points voisins d'un point commun de coordonnées (i, j)

- a) Exprimer la relation liant $U(X_{i-1}, Y_j)$ à $U(X_i, Y_j)$ par un développement limité à l'ordre deux.
- b) Faire de même avec tous les points cardinaux du maillage et démontrer la formule approchée suivante :

$$\Delta U = \frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2} \approx U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4 \cdot U_{i,j} = -\lambda' \cdot U_{i,j}$$

14°) Discrétisation autour des bords

Les bords de la plaque doivent être considérés différemment selon les conditions aux limites que nous désirons imposer. Il nous faut gérer les configurations types suivantes que nous désignerons respectivement par les termes de côté vertical, côté horizontal et de coins.

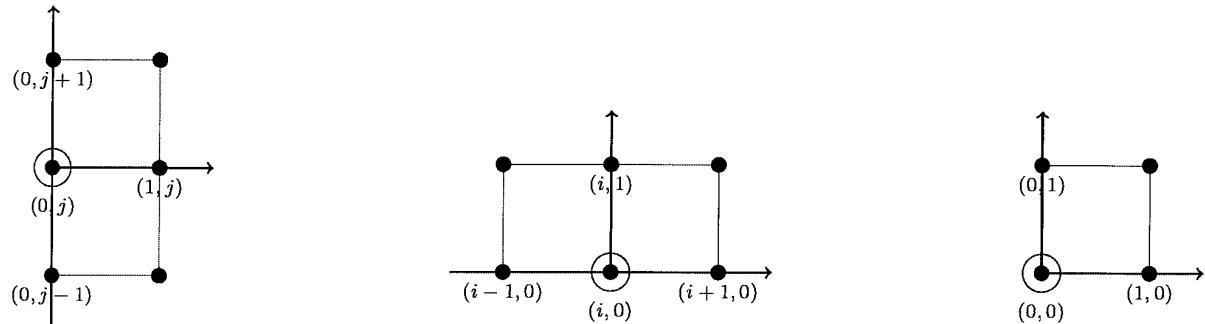


FIGURE 8 – Figuration des effets de bord

Les conditions aux limites classiques sont de deux types : soit les bords sont considérés comme fixes, soit ils sont considérés comme libres.

Dans notre cas, nous allons considérer que les points de la périphérie doivent être traités comme des points centraux, en associant aux points manquants de la distribution des valeurs nulles.

- En déduire la formule approchée de (4) pour le côté vertical (X_0, Y_j)
- En déduire la formule approchée de (4) pour le côté horizontal (X_i, Y_0)
- En déduire la formule approchée de (4) pour le coin (X_0, Y_0)
- Quelles sont les conditions aux limites retenues dans ce cas ?

Justifiez votre réponse.

15°) Détermination de l'équation matricielle équivalente

Nous introduisons le vecteur F_i tel que : $\forall i \in [0 \cdots N] \quad F_i = \begin{pmatrix} U_{i,0} \\ \vdots \\ U_{i,N} \end{pmatrix}$

Nous utilisons l'ensemble des vecteurs F_i pour générer le vecteur d'état F de l'ensemble de la plaque :

$$F = \begin{pmatrix} F_0 \\ \vdots \\ F_N \end{pmatrix}$$

Le système d'équations précédentes peut alors se mettre sous la forme d'une équation matricielle :

$$MF = \lambda' F \quad (5)$$

Les vecteurs F solutions de ce système sont les vecteurs propres de la matrice M , les λ' correspondants sont les valeurs propres associées.

- Combien y-a-t-il de modes propres possibles ?
- Nous nous plaçons dans le cas où $N = 4$ (cf figure 6), déterminer la matrice M .

c) Généralisation

Dans le cas où $N > 4$, la matrice M peut se mettre sous la forme suivante où I_N est la matrice identité de trace $N + 1$, et 0_N la matrice associée nulle de même dimension.

$$M = \begin{pmatrix} A & -I_N & 0_N & \cdots & 0_N \\ -I_N & A & \ddots & \ddots & \vdots \\ 0_N & \ddots & A & \ddots & 0_N \\ \vdots & \ddots & \ddots & A & -I_N \\ 0_N & \cdots & 0_N & -I_N & A \end{pmatrix}$$

Déterminer l'expression de A .

- d) La matrice M est symétrique, à coefficients réels, elle est, d'après le *Théorème spectral*, diagonalisable. C'est une matrice tridiagonale par blocs. Nous admettrons de plus qu'aucune de ces valeurs propres n'est nulle.

Nous supposons que le code qui génère la matrice A sous la forme d'un tableau 2D de flottants existe, et a été produit par d'autres.

Construire la fonction *genereM* recevant A en entrée et renvoyant M .

```
def genereM(A):
    """Fonction dont l'objectif est de renvoyer M
    à partir de A"""
    ...
    return M # renvoie la matrice M
```

I.5 Méthode de la puissance itérée ($\sim 10\%$)

Nous allons chercher l'une des solutions de l'équation aux valeurs propres (5) en utilisant la méthode dite de la *puissance itérée*. Soit $\mathcal{M}_{n,p}(\mathbb{R})$ l'ensemble des matrices de taille $n \times p$, et $\mathcal{M}_n(\mathbb{R}) = \mathcal{M}_{n,n}(\mathbb{R})$ sa restriction à l'ensemble des matrices carrées.

- 16°)** À chaque matrice $M = (m_{ij}) \in \mathcal{M}_{n,p}(\mathbb{R})$, nous associons un nombre réel positif appelé norme de M et défini par :

$$\|M\| = \max |m_{ij}| \quad \forall i, j \in [0 \cdots n-1, 0 \cdots p-1]$$

- a) Écrire en python la fonction *normeM* qui reçoit en entrée une matrice M de taille quelconque et renvoie la norme de M .

Le recours à une fonction max native de python est interdit pour cette question.

```
def normeM(M):
    """Fonction dont l'objectif est de renvoyer
    la norme de M à partir de M"""
    ...
    return r # norme de M
```

- b) Écrire en python la fonction *normevecteur* qui reçoit en entrée un vecteur (matrice colonne) F non nul, de taille quelconque, et renvoie un vecteur de même forme F^* défini par :

$$F^* = \frac{F}{\|F\|}$$

Le recours à une fonction max native de python est toujours interdit pour cette question.

```
def normevecteur(F):
    ...
    return Fn # vecteur normé
```

- 17°) Soit une matrice carrée $M \in \mathcal{M}_n(\mathbb{R})$, et soit F_0 un vecteur aléatoire de \mathbb{R}^n . Nous formons la suite $(F_p)_{p \geq 0}$ d'éléments de \mathbb{R}^n définie par la relation de récurrence :

$$F_{p+1} = \frac{M \cdot F_p}{\|M \cdot F_p\|}$$

- a) Écrire une fonction *puissanceiter* qui à partir d'une matrice carrée M et un entier p , choisit aléatoirement un vecteur F_0 de dimension adéquate, puis calcule et renvoie le p ème terme de la suite F_p .

```
def puissanceiter(M,p):
    ...
    return Fp # pième vecteur
```

- b) Nous admettrons que la suite F_p converge dans certaines conditions vers le vecteur propre principal associé à la valeur propre maximale en valeur absolue. Nous allons donc itérer le processus de calcul précédent jusqu'à ce que la suite F_p se stabilise. Soit $e_r > 0$ la valeur de seuil choisie pour la stabilité, il nous faut calculer les termes de la suite F_p jusqu'à ce que $\|F_p - F_{p-1}\| < e_r$.

Écrire le code de la fonction *iterstabilise* qui à partir d'une matrice carrée M et un réel $e_r > 0$, choisit aléatoirement un vecteur F_0 de dimension adéquate, puis calcule et renvoie le premier terme vérifiant la condition de stabilité de la suite F_p .

```
def iterstabilise(M,er):
    ...
    return F # vecteur stabilisé
```

- c) Pour produire le code de la question précédente, il vous faudra utiliser une boucle conditionnelle. Ce code va donc posséder un *variant* et un *invariant* de boucle. Après avoir rappelé les définitions de ces deux grandeurs, identifiez les dans le code que vous avez produit.

- 18°) Vecteur et valeur propre principaux

- a) Le vecteur F renvoyé par la fonction est alors le vecteur propre principal de M . Écrire le code python pour obtenir la valeur propre associée λ' .
- b) En déduire l'expression de la fréquence propre en Hertz du mode correspondant en fonction de λ' .

I.6 Représentation graphique ($\sim 5\%$)

Nous supposons que, par des méthodes numériques dont nous ne donnerons pas ici le détail, nous obtenons la suite des vecteurs propres et valeurs propres correspondants aux modes propres. Ces valeurs sont stockées dans deux tableaux de taille notée $n + 1$.

- Le tableau des valeurs propres est un tableau 1D où les valeurs propres sont ordonnées par ordre croissant, il est nommé *valprope*.
- Le tableau des vecteur propres est un tableau 2D où les vecteurs propres sont stockés selon l'indexation des valeurs propres, il est nommé *vecteurprope*. Le $i^{\text{ème}}$ vecteur propre s'obtient par *vecteurpropre*[$i, :$].

```

1 #librairies complémentaires
2 from mpl_toolkits.mplot3d import Axes3D
3 from matplotlib import cm
4
5 def graphe(Z):
6     n=Z.shape[0]
7     X=Y = np.arange(n)
8     X, Y = np.meshgrid(X, Y)
9
10    fig = plt.figure(0,figsize=(10,8))
11    ax = fig.gca(projection='3d')
12    surf = ax.plot_surface(X, Y, Z,rstride=1,cstride=1,
13                           cmap=cm.spectral_r, linewidth=0, antialiased=False)
14    ax.set_zlim(-1, 1)
15    fig.colorbar(surf, shrink=0.5, aspect=10)
16    plt.tight_layout()
17
18    fig1=plt.figure(1,figsize=(6,6))
19    cont=plt.contour(Z,np.arange(21)/10-1,cmap=cm.spectral_r)
20    plt.clabel(cont, cont.levels, inline=True,
21               fmt='%1.1f', fontsize=10)
22    plt.xlabel('x')
23    plt.ylabel('y')
24    plt.title('Courbes de niveau')
25    plt.tight_layout()
26    plt.show()
```

- 19°)** Le code ci-dessus permet la représentation des courbes de la figure (9). Pour fonctionner, il attend la matrice Z des valeurs U_{ij} .

$$Z = \begin{pmatrix} U_{00} & \cdots & U_{0j} & \cdots & U_{0n} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ U_{i0} & \cdots & U_{ij} & \cdots & U_{in} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ U_{n0} & \cdots & U_{nj} & \cdots & U_{nn} \end{pmatrix}$$

Il faut la fabriquer à partir du vecteur propre de type F (définition en Q15).

Écrire la fonction *transformeF* qui à partir d'un vecteur propre F renvoie la matrice U .

```
def transforme(F):
    ...
    return Z # matrice des valeurs U
```

- 20°)** Vous trouverez en page suivante (figure 9), quelques uns des modes de résonance de la plaque. Commenter ces courbes et corréler leurs tracés aux choix de modélisation effectués.

I.7 Distribution des valeurs propres ($\sim 15\%$)

Nous disposons du tableau des valeurs propres. C'est un tableau 1D *Numpy* où les valeurs propres sont ordonnées par ordre croissant, il est nommé *valproprie*.

- 21°)** Nous cherchons à savoir si une valeur se trouve dans les valeurs propres de la plaque.

Le recours à une fonction `in` native de python est interdit pour cette question.

- a) Si le tableau n'était pas ordonné, quelle serait la complexité asymptotique d'une telle recherche ?
- b) Écrire le code permettant une telle recherche pour la valeur v , tolérant un écart e_r de résolution et renvoyant un booléen comme résultat.

```
def recherchebrute(valproprie,v,er):
    ...
    return r # booléen
```

- c) Le tableau étant ordonné, une recherche par dichotomie est préférable.
Quelle est la complexité asymptotique d'une recherche dichotomique ?
- d) Rappelez le détail d'un algorithme de dichotomie.
- e) Écrire l'algorithme correspondant pour la valeur v .
Pour cette implémentation, vous pouvez ignorer l'erreur de précision et la considérer comme nulle.

```
def recherchedicho(valproprie,v):
    ...
    return r # booléen
```

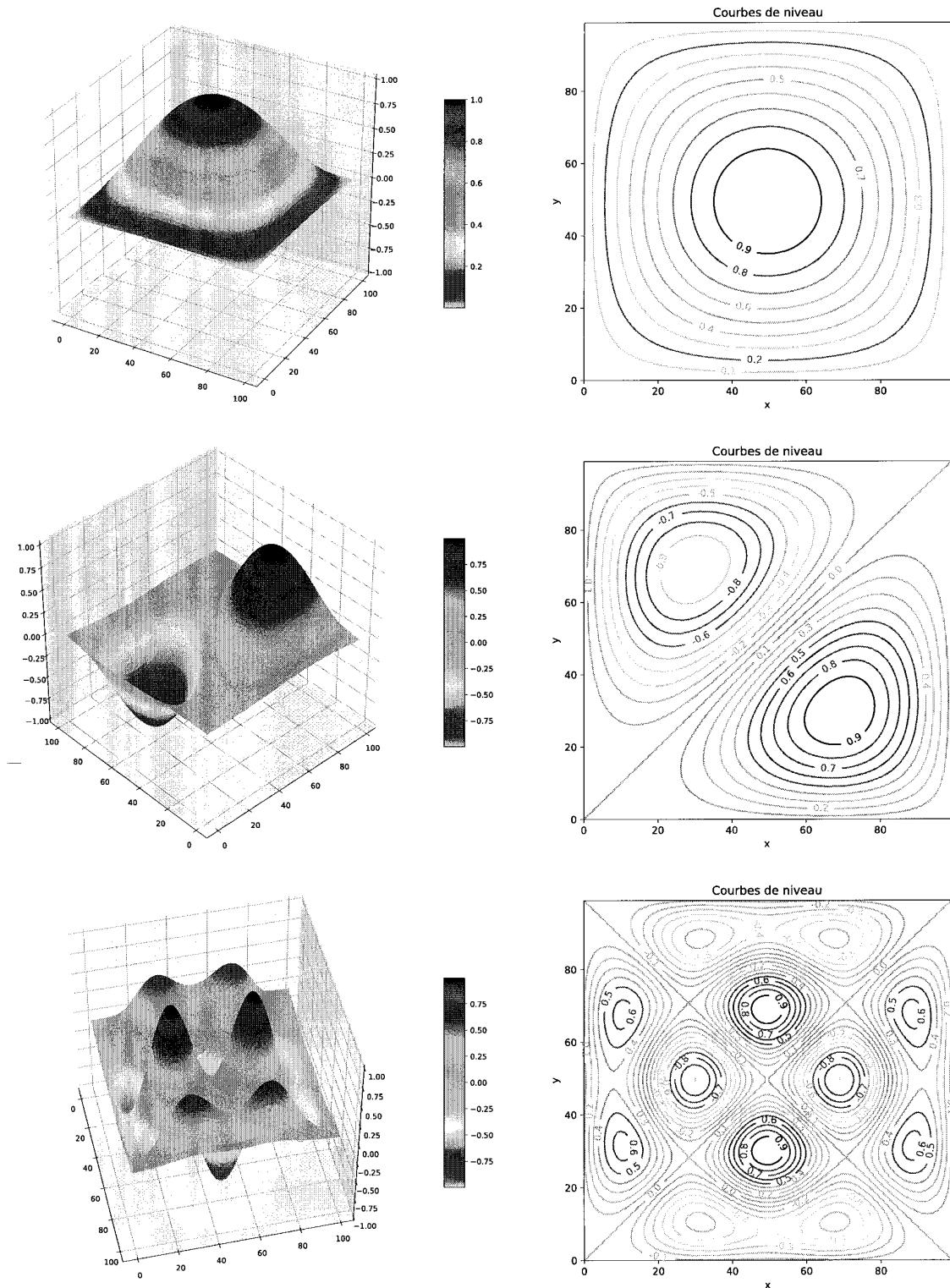


FIGURE 9 – Quelques modes propres

22°) En pratique, ces valeurs propres correspondent aux fréquences de résonance de la plaque. Dans l'étude d'ouvrages, il est bon de savoir si dans une plage de fréquence donnée, la structure possède des valeurs propres.

Nous pouvons alors prévoir les risques de mise en résonance de l'ouvrage. Toute résonance engendrera des oscillations de fortes amplitudes qui risquent d'amener les matériaux à sortir de leur plage de linéarité, endommageant ainsi durablement l'ouvrage.

Par exemple, le **pont de la Basse-Chaîne** en 1850 à Angers, et le **pont de Broughton** en 1831 près de Manchester, s'effondrent sous le passage d'une troupe militaire avançant au pas cadencé. Plus connu, l'effondrement du **pont de Tacoma** en 1940 est associé, dans la culture populaire, à l'existence d'une résonance du pont sous l'action des vents. L'exemple contemporain le plus proche est celui du **pont piéton du Millénium Bridge**. Inauguré en 2000 à Londres, il dut fermer deux jours plus tard en raison d'un phénomène de résonance provoqué par la marche des piétons.

- a) Nous désirons disposer d'une fonction nous renvoyant sous forme de tableau 1D *Numpy* toutes les valeurs propres de la plaque dans un intervalle délimité ouvert par *valmin* et *valmax*. Écrire le code permettant une telle recherche et utilisant la dichotomie de la question précédente.

```
def rechercheplagedicho(valpropre, valmin, valmax):  
    ...  
    return valselect # tableau1D
```

- b) Un habitué de la librairie *Numpy* écrit le code python suivant qui renvoie le tableau attendu. Quel est son principe de fonctionnement ? Commentez chacun des ordres présentés.

```
1 def valsearch(valpropre, valmin, valmax):  
2     mask=(valpropre>valmin) * (valpropre<valmax)  
3     return valpropre[mask]
```

- c) Quelle est la complexité asymptotique de ce code ?
d) En pratique, ce code s'avère près d'une centaine de fois plus rapide que celui de dichotomie. Comment pouvez-vous expliquer ce constat ?

I.8 Confrontation du modèle aux expériences ($\sim 20\%$)

Nous réalisons un dispositif expérimental permettant de confronter nos simulations à nos mesures. Nous considérons une plaque métallique uniforme et carrée sur laquelle nous disposons de fins grains de sable. La plaque est excitée par une onde acoustique émise par un haut parleur proche, pour certaines fréquences, la plaque entre en résonance avec l'émetteur sonore.

Nous pouvons alors distinguer, sur la plaque, la présence stable de ventres où l'amplitude de vibration est maximale, et la présence de noeuds où elle est nulle. Les grains sont expulsés des ventres et se concentrent à proximité des noeuds. Ces noeuds forment des courbes dites *lignes nodales*. Ce sont celles qu'a relevé Chladni et dont vous trouvez quelques exemples sur la figure 1. Nous avons relevé quelques unes des fréquences liées à ces résonances et nous les avons associées aux valeurs propres de l'équation (4) dans le tableau ci-après.

Nombre	1	2	3	4	5	6	7	8	9	10	11	12	13
valeur propre en ratio $\left(\frac{\lambda}{c_0}\right)$	2	5	5	8	10	10	13	13	17	17	18	20	20
fréquence en ratio $\left(\frac{\nu}{\nu_1}\right)$	1	2,5	2,5	4	5	5	6,5	6,5	8,5	8,5	9	10	10

Pour éviter toute difficulté de calcul, les valeurs propres λ ont été divisées par une constante dimensionnée c_0 dont la valeur n'a pas à être connue. Il en est de même pour les fréquences relevées qui sont rapportées à la première fréquence de résonance détectée qui est de l'ordre de la dizaine de *Hertz*.

L'analyse, par transformée de *Fourier*, des courbes obtenues nous montre que les solutions spatiales caractérisées par la simulation numérique sont des familles de fonctions $u(x, y)$ du type ci-après :

$$u(x, y) = C \sin\left(m\pi \frac{x}{L}\right) \sin\left(n\pi \frac{y}{L}\right)$$

où $m, n \in \mathbb{N}^*$ sont des entiers caractérisant le mode mn .

23°) Modèle initial. Il repose sur l'équation d'onde (1).

- a) En considérant les formes d'ondes introduites, établir l'équation de dispersion liant la fréquence temporelle de l'onde ν aux entiers m, n .
- b) Application numérique.
Déterminer l'ordre de grandeur de la fréquence de résonance la plus basse ν_1 .
Est-il compatible avec celui donné par l'expérience ?
- c) La distribution des premières fréquences de résonance est elle conforme à celle prévue par l'équation de dispersion ? (Une réponse argumentée est attendue)
- d) Que pouvez-vous en conclure sur la pertinence du modèle utilisé ?

24°) Un improbable modèle.

Un étudiant suggère d'utiliser le modèle régit par l'équation d'onde ci-après :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \alpha_1 \cdot \frac{\partial f}{\partial t}$$

- a) Expliquez en quelques lignes pourquoi ce modèle est inadéquat.
- b) Quelle est la dimension de la constante α_1 .

Tournez la page S.V.P.

Figures de Chladni

25°) Un autre modèle.

Un étudiant suggère d'utiliser le modèle régit par l'équation d'onde ci-après :

$$\Delta(\Delta f) = \nabla^2 (\nabla^2 f) = \nabla^4 f = \alpha_2 \cdot \frac{\partial^2 f}{\partial t^2}$$

- a) Quelle est la dimension de la constante α_2 .
- b) Par analyse dimensionnelle, proposez une expression de la constante α_2 .
- c) Quelle est l'équation de dispersion correspondant à cette équation d'onde pour les formes d'ondes introduites dans cette section ?
- d) La distribution des fréquences obtenue expérimentalement est elle conforme à celle prévue par cette nouvelle équation de dispersion ?
- e) Supposons que ce modèle soit plus pertinent que celui décrit par l'équation (1), montrer que, même dans ce cas, la recherche des solutions numériques de l'équation (3) reste adéquate.

26°) Dégénérescence des modes.

- a) Dans le tableau des relevés expérimentaux, nous constatons la présence de fréquences identiques, comment pouvez vous justifier ces dernières ?
- b) La prise en compte des fréquences d'index 2 et 3 nous permet de proposer une solution $z(x, y, t)$ sous la forme :

$$z(x, y, t) = C \left(\sin\left(\pi \frac{x}{L}\right) \sin\left(2\pi \frac{y}{L}\right) + \sin\left(2\pi \frac{x}{L}\right) \sin\left(\pi \frac{y}{L}\right) \right) \cos(\omega t)$$

Déterminer l'équation de la ligne nodale correspondante.

II Annexe : Rappel Python

L'objectif de cette épreuve est de modéliser des systèmes, et d'assurer, par le biais d'une approche numérique, la résolution des problématiques posées. L'étude des figures de Chladni nous amène à simuler numériquement la réponse d'une plaque métallique. Notre approche se situe dans le cadre de l'ingénierie numérique du programme et conformément aux recommandations évoquées dans ce dernier nous privilégierons l'usage de python avec les bibliothèques *Numpy*.

Rappel du programme officiel - extrait Ingénierie numérique et simulation :

...

L'objectif est de familiariser les étudiants avec un environnement de simulation numérique. Cet environnement doit permettre d'utiliser des bibliothèques de calcul numérique et leur documentation pour développer et exécuter des programmes numériques. On veillera à faire aussi programmer par les étudiants les algorithmes étudiés. Aucune connaissance des fonctions des bibliothèques n'est exigible des étudiants. Au moment de l'élaboration de ces programmes d'enseignement, l'atelier logiciel Scilab ou le langage de programmation Python, avec les bibliothèques Numpy/Scipy, sont les environnements choisis.

...

Scilab n'a pas été retenu pour cette épreuve et nous nous limitons au langage *Python*.

En accord avec les exigences officielles, les fonctions et procédures spécifiques de la bibliothèque *Numpy*, dont nous avons l'usage, sont rappelées dans la présente annexe.

Recommandations :

Les fonctions attendues dans le sujet n'utiliseront pas la récursivité.

Elles ne doivent disposer dans leur code d'implémentation que d'un seul **return**.

Les éventuelles procédures n'en auront, bien sur, aucun.

II.1 Librairies utilisées

Nous partons du principe que tous les programmes demandés sont précédés des appels suivants :

```
# -*- coding:Utf-8-*  
  
#Librairies utilisées et alias  
import numpy as np  
import matplotlib.pyplot as plt  
import numpy.random as rd
```

Les codes produits s'inspireront de ces quelques lignes pour l'exploitation des alias facilitant l'usage des fonctions et procédures des librairies appelées.

II.2 Usage courant de Numpy

La librairie *Numpy* privilégie l'usage du type tableau. Ces derniers seront ici essentiellement des tableaux à une dimension ou deux dimensions. A l'occasion, les premiers seront parfois désignés comme des vecteurs, et les seconds comme des matrices.

Nous n'utiliserons toutefois pas le type spécifique python *Matrix*.

```
## commandes de base

# définir un tableau nul de taille nxp
A=np.zeros(shape=(n,p))

# définir un tableau identité de taille nn
B=np.eye(n)

# définir un tableau rempli de 1 uniformément nxp
C=np.ones(shape=(n,p))

# obtenir les dimensions d'un tableau A : nxp
n,p=A.shape # usage de l'attribut shape

# obtenir le nombre d'éléments d'un tableau A
nb_elements=A.size # usage de l'attribut size
```

Les tableaux se prêtent à de nombreuses opérations arithmétiques. Par défaut, chaque opération est effectuée sur l'ensemble des cellules contenues dans le tableau.

```
## Opérations élémentaires sur les tableaux

# Addition d'un scalaire j à chaque terme du tableau
j=1
D=A+j

# multiplication par un scalaire k
k=2
D=k*D

# addition, soustraction, produit terme à terme
AddBC=B+C
SubBC=B-C
ProdBC=B*C
```

Attention le produit de base est un produit terme à terme de chaque élément du tableau. Il ne correspond pas au produit matriciel dont certains ont coutume.

Pour effectuer un produit matriciel, il faut utiliser la méthode *dot*.

```
## Opérations "matricielles" courantes

# produit matriciel B x C
PmatBC=B.dot(C)

# Tranposée du tableau C
C.T

# Lecture de la cellule [i,j] du tableau C à 2 dimensions
i,j=1,2
r=C[i,j]
```

Nous rappelons à cette occasion que l'indexation d'un tableau 2D à $N \times N$ éléments couvre en python la distribution $(i, j) \in [0, 1, \dots, N - 1]$.

II.3 Manipulation et copies de tableaux

Pour assurer la copie complète d'un tableau, il faut utiliser la méthode *copy*. Le simple égal est, à l'instar des listes de base, l'occasion de définir des alias

```
# Alias
F = C # F et C référencent le même tableau

## Copie d'un tableau
F=C.copy() # F référence une copie du tableau C
```

la Librairie propose le redimensionnement d'un tableau avec la fonction *reshape*. Il faut bien sur que la forme visée ait le même nombre d'éléments que la forme initiale.

```
## Reformage

# un tableau peut être réorganisé par la méthode reshape
E=np.ones(9) # vecteur 1D de neuf 1
print(E)
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.]

F=E.reshape(3,3) # reforme E en tableau 2D 3x3
print(F)
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]

G=F.flatten() # renvoie le tableau sous forme 1D
print(G)
[ 1.  1.  1.  1.  1.  1.  1.  1.]
```

Numpy permet d'assembler les vecteurs et les matrices, de les concaténer en utilisant la fonction correspondante *concatenate*. Par défaut l'assemblage se fait selon la 1ère dimension (les lignes, donc assemblage vertical). L'option axis=1 assemble “horizontalement”.

```
a=np.arange(4).reshape(2,2)
    array([[0, 1],
           [2, 3]])

b=4+np.arange(4).reshape(2,2)
    array([[4, 5],
           [6, 7]])

np.concatenate((a,b))
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7]])

np.concatenate((a,b),axis=0)
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7]])

np.concatenate((a,b),axis=1)
    array([[0, 1, 4, 5],
           [2, 3, 6, 7]])
```

Les plus coutumiers d'entre vous pourront utiliser les techniques de slicing et de masquage qui ne seront pas rappelées ici. Si la librairie favorise le traitement en masse, nous acceptons néanmoins tous les codes utilisant de simples boucles.

II.4 Utilisation du générateur aléatoire - *numpy.random*

Python dispose de plusieurs librairies permettant la génération aléatoire de nombres. Pour des raisons pratiques évidentes (vectorisation), nous exploitons ici la librairie random intégrée au sein de *Numpy*. Les utilisations correctes d'autres librairies, correctement introduites et exploitées seront bien sur acceptées.

La fonction *rand()* crée un tableau d'un format donné de réels aléatoires dans $[0, 1]$. La dimension du tableau est donnée par le paramètre passé dans le champ argument de la fonction, qui n'est autre que la forme *shape* du tableau renvoyé.

Exemple :

```
import numpy.random as rd # Appel de la librairie

# tableau 1D comportant 10 éléments
print(rd.rand(10))
>>> [0.16653536 0.37216705 0.56565704 0.62948842 0.65555812 0.38543034
     0.1486412 0.52610282 0.4686787 0.81558018]

# tableau 2D comportant 3X2 éléments
print(rd.rand(3,2))
>>> [[0.98992966 0.58907376]
     [0.6404654 0.53852321]
     [0.12416014 0.28279454]]
```

FIN DE L'ÉPREUVE