

INFORMATIQUE 2018



Table des matières

Option informatique	1
X – ENS A (XULCR) (4h) [i183m1e]	
<i>Nombre chromatique et coloriage de graphe</i>	1
Mines Ponts (3h) [i18mmoe]	
<i>Recherche de motifs, automates à replis, Knuth-Morris-Pratt</i>	8
Centrale Supélec (4h) [i18cmoe]	
<i>Étude du jeu Ricochet Robots, tri, file, tables de hachage, graphe et parcours</i> . .	17
CCP (4h) [i18pmoe]	
<i>Logique, automate minimal, compression de données</i>	23
E3A (3h) [i18rmoe]	
<i>Terminaison, récursivité, graphe, SQL, logique</i>	31
Epita / Ipsa / Esme (2h) [i18wine]	
<i>QCM</i>	39
CAPES externe de Mathématiques, option Informatique [i18c3oe]	
<i>Suite de Lucas, emplois du temps et graphes d'intervalles</i>	47
ENS Paris, sélection internationale, spécialité principale (sujet 2017) [i17uxue]	
<i>Complexités, probabilités, optimisation, structure de données</i>	56
ENS Paris, sélection internationale, spécialité secondaire (sujet 2017) [i17uxve]	
<i>Tris, programmation dynamique</i>	59
ENS Lyon, second concours (sujet 2017) [i1732ue]	
<i>Représentation d'ensembles disjoints</i>	61
Informatique commune	68
X – ENS B (XELCR) (2h) [i183m2e]	
<i>Python et SQL</i>	68
Mines Ponts (1,5h) [i18mice]	
<i>Mesures de houle, traitement de données, SQL</i>	81
Centrale Supélec (3h) [i18cice]	
<i>Simulation de la cinétique d'un gaz parfait</i>	92
CCP - PC (4h) [i18ppce]	
<i>Modélisation de transfert thermique, traitements de données</i>	100
CCP - PSI (3h) [i18psue]	
<i>Trajectoire d'une balle de tennis, système Hawk-eye</i>	120
CCP - TSI (3h) [i18piue]	
<i>Optimisation d'un correcteur P.I.D. par un algorithme génétique</i>	136
CCP - TPC (4h) [i18pcue]	
<i>Modélisation de transfert thermique, traitements de données</i>	164
ICNA (1h) [i18b2uea]	
<i>QCM</i>	184

En guise d'introduction

*Heureux soient les félés car ils laisseront passer la lumière.
Michel Audiard – Le cave se rebiffe*

Voici le recueil de la plupart des sujets d'informatique posés aux concours des grandes écoles scientifiques en 2018 et disponibles actuellement. Contrairement au bulletin 2017, celui-ci ne contient pas les sujets comportant au moins une question d'informatique. Question de temps !

Le recueil comporte deux parties, la première contenant les épreuves de l'option informatique de nos classes ainsi que quelques autres sujets, la seconde contenant les épreuves d'informatique commune.

Les sujets ENS, sélection internationale et second concours sont ceux de 2017. Ils n'avaient pas été publiés dans le bulletin l'an dernier, comme ceux de 2018 ne peuvent l'être cette année puisqu'ils ne sont pas encore disponibles en ligne ! Noter que le sujet ENS, sélection internationale, spécialité principale, porte la mention *Session 2015* bien qu'il s'agisse du sujet de 2017. Une erreur de frappe certainement !

Comme l'an dernier, 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
Août 2018

ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2018

FILIÈRES MP SPECIALITÉ INFO

COMPOSITION D'INFORMATIQUE – A – (XULCR)

(Durée : 4 heures)

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

Le langage de programmation sera **obligatoirement Caml Light**.

*
* *

Nombre chromatique et coloriage de graphe

Préliminaires

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 pire cas.

Lorsque la complexité en temps dépend d'un ou plusieurs paramètres $\kappa_1, \dots, \kappa_r$, on dit qu'elle est en $O(f(\kappa_1, \dots, \kappa_r))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $\kappa_1, \dots, \kappa_r$ suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), la complexité est au plus $C \times f(\kappa_1, \dots, \kappa_r)$.

On dit que la complexité en temps est **linéaire** quand f est une fonction linéaire des paramètres $\kappa_1, \dots, \kappa_r$, **polynomiale** quand f est une fonction polynomiale des paramètres $\kappa_1, \dots, \kappa_r$, et enfin **exponentielle** quand $f = 2^g$, où g est une fonction polynomiale des paramètres $\kappa_1, \dots, \kappa_r$.

Les complexités (en temps) des algorithmes **devront être justifiées**.

Graphes. Rappelons qu'un graphe non-orienté est la donnée (S, A) de deux ensembles finis :

- un ensemble S de **sommets**, et
- un ensemble $A \subseteq S \times S$ d'**arêtes**, tel que pour tout couple de sommets (s, t) , on a $(s, t) \in A$ si et seulement si $(t, s) \in A$.

Étant donné un graphe $G = (S, A)$, le **sous-graphe induit** par un ensemble de sommets $T \subseteq S$ est $(T, A \cap (T \times T))$.

Soit $G = (S, A)$ un graphe et soit $s \in S$ un sommet de G . Un **voisin** de s est un sommet t de G qui est relié à s par une arête, c'est-à-dire tel que $(s, t) \in A$. On note $V(s)$ l'ensemble des voisins de s . Le **degré** $d(s)$ de s est le cardinal de $V(s)$. Le **degré** $d(G)$ de G est le maximum des degrés de ses sommets.

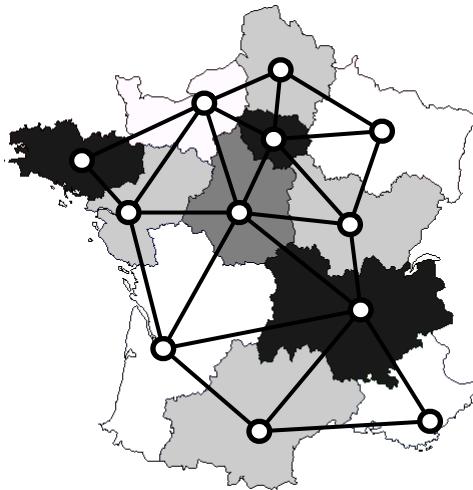


FIGURE 1 – Exemple de graphe colorié : le graphe des régions métropolitaines françaises (hors Corse). Deux régions sont reliées par une arête dans le graphe si et seulement si elles sont voisines. Deux régions voisines sont de couleurs différentes.

Un graphe est dit **étiqueté** lorsque l'on dispose d'une fonction, dite d'étiquetage, de l'ensemble de ses sommets vers un ensemble non-vide arbitraire, que l'on appelle ensemble des étiquettes. Les étiquettes peuvent par exemple être des entiers, des listes ou des chaînes de caractères.

On dit qu'une fonction d'étiquetage L est un **coloriage** des sommets de $G = (S, A)$ lorsque deux sommets voisins ont toujours deux étiquettes distinctes (alors appelées **couleurs**), c'est-à-dire lorsque L vérifie la condition (1) suivante :

$$\forall s, t \in S, \quad (s, t) \in A \Rightarrow L(s) \neq L(t). \quad (1)$$

Un graphe G est dit k -colorable s'il admet un coloriage avec au plus k couleurs. Un graphe est dit colorié s'il est k -colorable pour un $k > 0$. Un exemple de graphe colorié est donné sur la figure 1 ci-dessus.

Le **nombre chromatique** d'un graphe non-orienté G , noté $\chi(G)$, est le nombre minimal k tel que G est k -colorable. Cet énoncé porte sur le calcul de nombres chromatiques et de coloriages.

Représentation des graphes étiquetés. On se fixe dans cet énoncé une représentation des graphes par matrices d'adjacence. On se fixe également comme convention que les étiquetages des graphes sont tous à valeurs entières. L'étiquetage d'un graphe sera donné par un tableau d'entiers. On se donne les types Caml Light suivants :

```
type graphe == bool vect vect;;
type etiquetage == int vect;;
```

Un graphe non-orienté $G = (S, A)$ avec $S = \{0, \dots, n - 1\}$ est représenté par une valeur `gphe` de type `graphe` telle que pour tous sommets $i, j \in S$, on ait `gphe.(i).(j) = true` si et seulement si $(i, j) \in A$. Le graphe G étant supposé non-orienté, on a alors également par symétrie

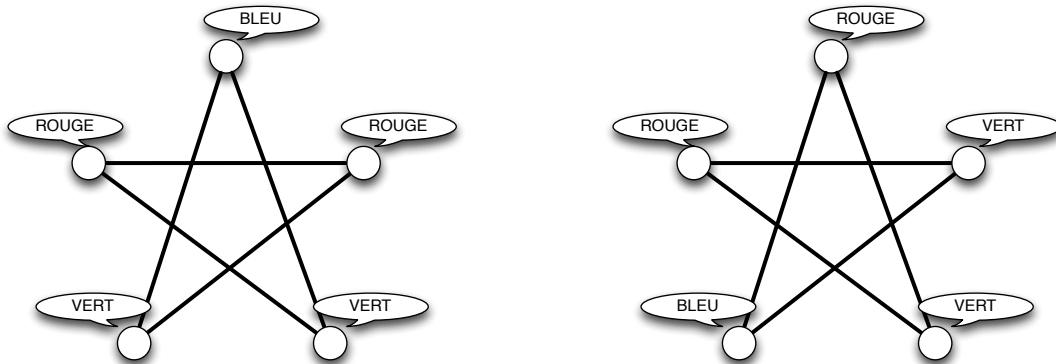
`gphe.(j).(i) = true`. Pour un étiquetage `etiq` de `gphe`, l'étiquette du sommet `i` de `gphe` est donnée par `etiq.(i)`.

En plus des fonctionnalités de base du langage Caml Light, le candidat pourra utiliser les fonctions suivantes sans les programmer :

- `make_vect : int -> 'a -> 'a vect`
`make_vect n v` renvoie un vecteur de longueur `n` et dont toutes les cases valent `v`.
- `vect_length : 'a vect -> int`
`vect_length t` renvoie la longueur de `t`.
- `list_length : 'a list -> int`
`list_length l` renvoie la longueur de `l`.
- `vect_of_list : 'a list -> 'a vect`
`vect_of_list l` renvoie un vecteur `t` de même longueur que `l`, qui contient les mêmes éléments que `l` et dans le même ordre.
- `range : int -> int vect`
`range n` renvoie le vecteur `[|0, ..., n-1|]`.

1 Coloriage

Question 1. Indiquer, pour chacun des graphes suivants, s'il est colorié :



Question 2. Donner le nombre chromatique, ainsi qu'un exemple de coloriage correspondant, pour le **graphe de Petersen** représenté figure 2 page 4.

Question 3. La vérification de la propriété de coloriage est le problème suivant.

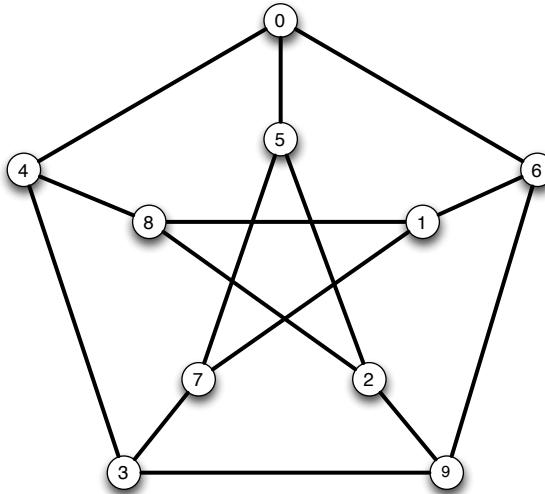
- *Entrée* : un graphe `G`, et un étiquetage `L` de `G`.
- *Question* : `L` est-il un coloriage de `G` ?

Écrire une fonction `est_col : graphe -> étiquetage -> bool`, telle que `est_col gphe etiq` renvoie `true` si et seulement si `etiq` est un coloriage de `gphe`. Dans le cas où la taille de l'étiquetage est strictement inférieure au nombre de sommets du graphe, la fonction renvoie `false`. On demande une complexité quadratique en le nombre de sommets du graphe.

Question 4. Démontrer que le calcul du nombre chromatique d'un graphe peut s'effectuer en temps exponentiel en le nombre de sommets.

2 2-coloriage

Nous avons vu à la question 4 que le calcul du nombre chromatique peut s'effectuer en temps exponentiel en le nombre de sommets du graphe. Dans le cas général, on ne sait aujourd'hui

FIGURE 2 – Le graphe de Petersen, de sommets $0, \dots, 9$.

pas faire mieux. Pour obtenir de meilleures bornes de complexité, il faut donc se limiter à des sous-problèmes. On considère dans cette partie le cas du 2-coloriage.

Graphe biparti. Un graphe G est **biparti** lorsque l'ensemble de ses sommets S peut être divisé en deux sous-ensembles disjoints T et U , tels que chaque arête a une extrémité dans T et l'autre dans U .

Question 5. Démontrer qu'un graphe G est biparti si et seulement s'il est 2-coloriable.

On se propose de programmer la vérification de la 2-colorabilité des graphes en procédant comme suit. On effectue un parcours du graphe en profondeur au cours duquel on construit une 2-coloration du graphe. On se donne pour ce faire trois étiquettes, disons -1 , 0 et 1 . L'étiquetage est initialisé à -1 pour tous les sommets, et on teste la 2-colorabilité avec 0 et 1 . Le principe de l'algorithme est le suivant :

- (1) On choisit un sommet s d'étiquette -1 .
- (2) On colorie les sommets rencontrés lors du parcours en profondeur à partir de s , en alternant entre les couleurs 0 et 1 à chaque incrémentation de la profondeur, et en vérifiant si les sommets déjà coloriés rencontrés sont d'une couleur compatible.
- (3) Enfin, s'il reste des sommets d'étiquette -1 , alors on revient au point (1).

Question 6. Écrire une fonction `deux_col : graphe -> etiquetage` telle que `deux_col gphe` renvoie un 2-coloriage de `gphe` si `gphe` est 2-coloriable. Le coloriage utilisera les couleurs 0 et 1 . On demande une complexité quadratique en le nombre de sommets du graphe. Le comportement de la fonction est laissé au choix du candidat lorsque `gphe` n'est pas 2-coloriable.

Indication : on pourra se donner un étiquetage `etiq : etiquetage de longueur (vect_length gphe)`, dont toutes les cases sont initialisées à -1 , et que l'on met à jour au fur et à mesure du parcours de `gphe`.

3 Algorithmes gloutons

Dans cette partie, nous allons étudier deux algorithmes permettant de colorier un graphe en temps polynomial, mais donnant en général un coloriage sous-optimal : le coloriage obtenu peut dans certains cas utiliser plus de couleurs que le coloriage optimal.

Ces deux algorithmes prennent en paramètre un ordre sur les sommets du graphe, que l'on appellera **ordre de numérotation**. Par exemple, $1 < 3 < 4 < 0 < 2 < 6 < 5 < 9 < 8 < 7$ et $0 < 7 < 2 < 5 < 4 < 6 < 8 < 1 < 3 < 9$ sont deux ordres de numérotation des sommets du graphe de Petersen (figure 2 page 4).

Pour un graphe `gphe` à n sommets, on implémente un ordre de numérotation de ses sommets par un tableau `num` de n valeurs entières, tel que $\text{num}(k) = j$ si et seulement si le sommet j apparaît en $(k+1)^{\text{ième}}$ position dans l'ordre.

Nous commençons par l'**algorithme glouton** de coloriage. Cet algorithme construit un coloriage L d'un graphe G en utilisant au plus $d(G) + 1$ couleurs. Son principe est le suivant :

On parcourt la liste des sommets du graphe, dans l'ordre de numérotation des sommets donné. Pour chaque sommet s parcouru :

- (1) On calcule l'ensemble $C(s) = \{L(t) \mid t \in V(s)\}$ des couleurs déjà données aux voisins de s .
- (2) On cherche le plus petit entier naturel c qui n'appartient pas à $C(s)$.
- (3) On pose $L(s) = c$.

Question 7. Considérons le graphe de Petersen (figure 2, page 4) et les deux ordres de numérotation `num1 = [[1; 3; 4; 0; 2; 6; 5; 9; 8; 7]]` et `num2 = [[0; 7; 2; 5; 4; 6; 8; 1; 3; 9]]`. Donner les coloriages obtenus par l'algorithme glouton décrit ci-dessus pour le graphe de Petersen et chacun de ces deux ordres de numérotation, ainsi que les nombres de couleurs correspondants.

Question 8. Écrire une fonction `min_couleur_possible : graphe -> etiquetage -> int -> int` telle que pour un graphe `gphe` à n sommets, un étiquetage `etiq` à valeurs dans $\{-1, \dots, n-1\}$, et pour un sommet `s` de `gphe`, l'appel de `min_couleur_possible gphe etiq s` renvoie le plus petit entier naturel n'appartenant pas à l'ensemble $\{\text{etiq}.(t) \mid t \in V(s)\}$. On demande une complexité en $O(n)$.

Question 9. Écrire une fonction `glouton : graphe -> int vect -> etiquetage`, telle que, pour un graphe `gphe` et un ordre de numérotation `num` de ses sommets, `glouton gphe num` renvoie le coloriage glouton de `gphe`, avec au plus $d+1$ couleurs, où d est le degré de `gphe`. On demande une complexité en $O(n^2)$, où n est le nombre de sommets de `gphe`.

Dans le cas où le tableau `num` contient autre chose qu'un ordre de numération des sommets de `gphe`, le résultat de la fonction est laissé au choix du candidat.

Question 10. Montrer que l'algorithme de coloriage glouton construit toujours un coloriage, et que ce coloriage utilise au plus $d+1$ couleurs, où d est le degré du graphe en entrée.

Question 11. Soit G un graphe. Montrer que pour tout coloriage L de G , il existe un ordre de numérotation des sommets tel que le coloriage glouton L' associé vérifie $L'(s) \leq L(s)$ pour tout sommet s de G . En déduire qu'il existe une numérotation des sommets telle que l'algorithme glouton renvoie un coloriage optimal.

Les questions 7 et 11 indiquent que l'efficacité de l'algorithme glouton est en grande partie dépendante de l'ordre dans lequel on choisit de parcourir les sommets du graphe. L'ordre correspondant à la représentation choisie du graphe (dans notre cas les indices de la matrice d'adjacence, c'est-à-dire la permutation identité) est le plus simple à calculer, mais a peu de

chances d'être efficace. A contrario, on pourrait essayer de déterminer l'ordre optimal, dont on a prouvé l'existence à la question 11, mais cela n'apporterait aucun bénéfice vis-à-vis de la complexité temporelle du problème.

Une alternative est donnée par l'optimisation de Welsh-Powell. L'idée est de parcourir l'ensemble des sommets du graphe par ordre de degré décroissant. Le tri des sommets par degré décroissant ne prend pas plus de temps que le parcours glouton, mais permet d'obtenir un algorithme raisonnablement efficace en pratique.

Question 12. Écrire une fonction `tri_degre: graphe -> int vect`, qui calcule le tableau des sommets d'un graphe trié par ordre décroissant de leurs degrés. En déduire une fonction `welsh_powell: graphe -> etiquetage` qui implémente l'optimisation de Welsh-Powell, et justifier le choix de votre algorithme de tri pour la fonction `tri_degre`.

4 Algorithme de Wigderson

Considérons un graphe G avec n sommets. Supposons que G soit 3-coloriable, mais que l'on ait cette information sans pour autant effectivement disposer d'un 3-coloriage de G . Trouver un 3-coloriage de G pourrait prendre un temps exponentiel en n .

L'algorithme de Wigderson permet, pour un graphe G supposé 3-coloriable, de trouver en temps polynomial en n un coloriage de G avec $O(\sqrt{n})$ couleurs (au sens où il existe $C > 0$ tel que pour tout n suffisamment grand, ce coloriage ait au plus $C\sqrt{n}$ couleurs).

Cet algorithme repose sur la propriété établie dans la question qui suit.

Question 13. Soit $k > 0$. Montrer que si G est $(k + 1)$ -coloriable, alors pour tout sommet s de G le sous-graphe induit par $V(s)$ est k -coloriable.

Voici le principe de l'algorithme de Wigderson. Soit G un graphe à n sommets, et tel que G est 3-coloriable.

- (1) On se donne comme couleur initiale $c = 0$.
- (2) Pour chaque sommet s de G pas encore colorié et ayant au moins \sqrt{n} voisins pas encore coloriés :
 - (a) On 2-colorie, avec les couleurs c et $c + 1$, le sous-graphe induit par l'ensemble des voisins de s pas encore coloriés.
 - (b) On incrémente c du nombre de couleurs utilisées en (a).
- (3) Enfin, on utilise l'algorithme glouton (avec un ordre de numérotation quelconque) pour colorier, avec des couleurs supérieures ou égales à c , le sous-graphe induit par l'ensemble des sommets pas encore coloriés.

Question 14. Montrer que l'algorithme de Wigderson appliqué à un graphe 3-coloriable construit toujours un coloriage, et que ce coloriage utilise un nombre de couleur en $O(\sqrt{n})$, où n est le nombre du sommets du graphe.

Nous allons maintenant implémenter cet algorithme. Commençons par programmer quelques fonctions auxiliaires simples.

Question 15. Écrire une fonction `sous_graphe : graphe -> int vect -> graphe` telle que pour `gphe : graphe` et `sg : int vect`, si `sg` est à valeurs dans $\{0, \dots, (\text{vect_length gphe}) - 1\}$ et sans répétition, alors `sous_graphe gphe sg` renvoie la matrice d'adjacence du graphe de

sommets $\{0, \dots, (\text{vect_length sg}) - 1\}$, et qui a une arête entre les sommets s et t si et seulement si $\text{gphe}.\text{(sg.(s))}.\text{(sg.(t))} = \text{true}$.

Dans le cas où sg a des valeurs hors de $\{0, \dots, (\text{vect_length gphe}) - 1\}$, ou a des répétitions, le comportement de la fonction `sous_graphe` est laissé au choix du candidat.

Nous nous proposons d'utiliser pour les étiquetages la même convention que précédemment : on se donnera un étiquetage `etiq` : `etiquetage` de longueur $(\text{vect_length gphe})$, initialisé à -1 , et que l'on mettra à jour au fur et à mesure de l'algorithme.

Question 16. Écrire une fonction `voisins_non_colories` : `graphe` \rightarrow `etiquetage` \rightarrow `int` \rightarrow `int list` telle que `voisins_non_colories` $gphe$ $etiq$ s renvoie la liste des voisins t de s tels que $etiq.(t) = -1$.

En déduire une fonction `degre_non_colories` : `graphe` \rightarrow `etiquetage` \rightarrow `int` \rightarrow `int` telle que `degre_non_colories` $gphe$ $etiq$ s renvoie le nombre de voisins t de s tels que $etiq.(t) = -1$.

Question 17. Écrire une fonction `non_colories` : `graphe` \rightarrow `etiquetage` \rightarrow `int list` telle que `non_colories` $gphe$ $etiq$ renvoie la liste des sommets s de $gphe$ tels que $etiq.(s) = -1$.

Nous disposons maintenant de toutes les briques nécessaires à l'implémentation de l'algorithme de Wigderson.

Question 18. Écrire une fonction `wigderson` : `graphe` \rightarrow `etiquetage` telle que si $gphe$ est 3-coloriable, alors `wigderson` $gphe$ renvoie un coloriage de $gphe$ obtenu par l'algorithme de Wigderson décrit plus haut. On demande une complexité polynomiale en le nombre de sommets de $gphe$. De plus, les propriétés sur le coloriage établies à la question 14 devront être respectées et justifiées.

Le comportement de la fonction est laissé au choix du candidat lorsque $gphe$ n'est pas 3-coloriable.

Question 19. Comment pourrait-on étendre l'algorithme de Wigderson à des graphes de nombre chromatique connu et strictement supérieur à 3 ?

* *
*

A2018 – INFO MP

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

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

CONCOURS 2018**ÉPREUVE D'INFORMATIQUE 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 8 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.*

Épreuve d'option informatique MP 2018

L'épreuve est composée d'un unique problème, comportant 24 questions. Après un préliminaire et des définitions générales, ce problème est divisé en 4 parties ; la partie 2 est indépendante de la partie 1.

Le but du problème est d'étudier des algorithmes permettant de rechercher efficacement des occurrences d'une ou plusieurs chaînes de caractères (appelées *motifs*) à l'intérieur d'une longue chaîne de caractères.

Préliminaire 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 dans l'écriture de cette fonction de tester si les hypothèses sont bien vérifiées.

Dans les questions du problème, 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).

Définitions générales

On fixe un *alphabet* Σ , c'est-à-dire un ensemble fini, dont les éléments sont appelés *symboles*. On note $\lambda > 0$ le nombre d'éléments de cet alphabet, et on suppose que cette taille λ est disponible depuis les fonctions Caml à implémenter sous la forme d'une constante globale `lambda`. Par exemple :

```
let lambda = 5;;
```

On supposera que les éléments de Σ sont ordonnés, par exemple par ordre alphabétique, et on les note dans l'ordre $\alpha_0 < \dots < \alpha_{\lambda-1}$. On dit que le *code* d'un symbole α de l'alphabet est l'entier i tel que $\alpha = \alpha_i$ ($0 \leq i \leq \lambda - 1$).

Une *chaîne de caractères* s est une suite finie *non vide* $u_1 \dots u_k$ de symboles de Σ . La *longueur* de s est son nombre de symboles, c'est-à-dire, ici, k . On représentera en Caml les chaînes de caractères par des listes d'entiers (`int list`), chaque entier étant le code d'un symbole. Ainsi, si l'alphabet est $\Sigma = \{a, b, c\}$, dans cet ordre, la chaîne de caractères « *abac* » sera représentée par la liste `[0; 1; 0; 2]`. En particulier, on n'utilisera jamais le type `string` de Caml.

 Épreuve d'option informatique MP 2018

1 Recherche naïve d'un motif

Une *occurrence* d'une chaîne de caractères $s = \alpha_1 \dots \alpha_k$ (aussi appelée un *motif*) dans une autre chaîne de caractères $t = \beta_1 \dots \beta_n$ est un entier naturel y avec $1 \leq y \leq n$ tel que, pour tout entier i tel que $0 \leq i \leq k - 1$, $\alpha_{k-i} = \beta_{y-i}$. En d'autres termes, l'occurrence indique la position du dernier caractère du motif s au sein de la chaîne de caractères t (les positions commençant à 1).

Par exemple, si $\Sigma = \{a, b, c, d, e\}$, s est le motif « *abc* » et t la chaîne de caractères « *abcabcdababcdababcdabde* », il y a quatre occurrences de s dans t , à savoir :

- 3
- 6
- 12
- 16

- 1 – Soit s, t deux chaînes de caractères. Est-il possible d'avoir deux occurrences y et y' de s dans t avec $y < y'$ et $y \geq y' - k + 1$? Si oui, donner un exemple ; sinon, le prouver.
- 2 – Quel est le nombre maximal d'occurrences d'un motif de longueur k dans une chaîne de caractères de longueur $n \geq k$? Prouver que cette borne est toujours respectée, et que cette borne est atteinte.
- 3 – Programmer en Caml une fonction `longueur : 'a list -> int` qui prend en entrée une liste et qui renvoie la longueur de cette liste. Quelle est la complexité de cette fonction, en terme du nombre n d'éléments de la liste ?
- 4 – Programmer en Caml une fonction `int list -> int list -> bool` prenant en entrée deux listes d'entiers s et t codant des chaînes de caractères et testant si s est un préfixe de t . Quelle est la complexité de cette fonction, en terme des longueurs k et n de s et t ?
- 5 – À l'aide des fonctions `longueur` et `prefixe`, programmer une fonction `recherche_naive : string -> string -> int list`, la plus directe possible, telle que si s est un motif et t une chaîne de caractères, `recherche_naive s t` renvoie la liste des entiers y qui sont des occurrences de s dans t , triés par ordre croissant.
- 6 – Quelle est la complexité de la fonction `recherche_naive`, en terme des longueurs k et n de ses deux paramètres s et t ?

2 Automates finis déterministes à repli

Un *automate fini déterministe à repli* (ou *AFDR*) sur l'alphabet Σ est un quadruplet $\mathcal{A} = (k, F, \delta, \rho)$ tel que :

- $k \in \mathbb{N}$ est un entier strictement positif représentant le *nombre d'états* de \mathcal{A} ; l'ensemble des *états* de \mathcal{A} est $Q_{\mathcal{A}} = \{0, 1, \dots, k - 1\}$ et 0 est appelé l'*état initial*.
- $F \subseteq Q_{\mathcal{A}}$ est un ensemble d'états, appelés *finals*.
- $\delta : Q_{\mathcal{A}} \times \Sigma \rightarrow Q_{\mathcal{A}}$ est une *fonction partielle de transition* (c'est-à-dire, une fonction dont le domaine de définition est un sous-ensemble de $Q_{\mathcal{A}} \times \Sigma$) ; on impose que $\delta(0, \alpha)$ soit défini pour tout $\alpha \in \Sigma$.
- $\rho : Q_{\mathcal{A}} \setminus \{0\} \rightarrow Q_{\mathcal{A}}$ est une application (c'est-à-dire, une fonction totale), appelée *fonction de repli*, telle que pour tout $q \in Q_{\mathcal{A}}$ avec $q \neq 0$, $\rho(q) < q$. On prolonge ρ en convenant $\rho(0) = 0$.

Un AFDR est représenté en Caml par le type enregistrement suivant :

```
type afdr = {
  final : bool vect;
  transition : int vect vect;
  repli: int vect;
};;
```

où :

- *final* est un tableau de taille k de booléens tel que si $q \in Q_{\mathcal{A}}$, *final.(q)* contient *true* si et seulement si $q \in F$;
- *transition* est un tableau de taille k de tableaux de taille λ d'entiers, tel que si $q \in Q_{\mathcal{A}}$ et $\alpha_i \in \Sigma$ de code i , *transition.(q).(i)* contient -1 si $\delta(q, \alpha_i)$ n'est pas défini, et contient $\delta(q, \alpha_i)$ sinon ;
- *repli* est un tableau de taille k d'entiers tel que *repli.(0)* contient 0 et *repli.(q)* pour $q \in Q_{\mathcal{A}} \setminus \{0\}$ contient $\rho(q)$.

On observe que le type *afdr* ne code pas explicitement la valeur de k , mais k est par exemple la longueur du tableau *final*.

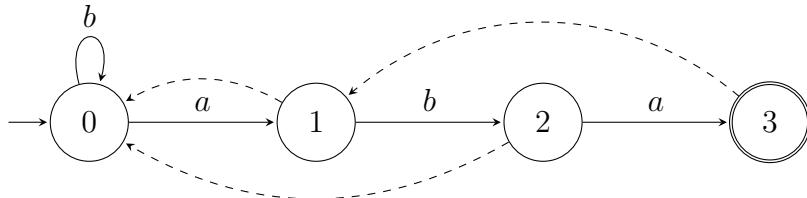
On dessine un AFDR de manière similaire au dessin d'un automate fini déterministe classique, en faisant figurer en pointillés une flèche indiquant les replis d'un état vers un autre. Les états finals sont figurés par un double cercle.

Considérons par exemple l'automate $\mathcal{A}_1 = (k_1, F_1, \delta_1, \rho_1)$ sur l'alphabet $\Sigma = \{a, b\}$ défini par $k_1 = 4$, $F_1 = \{3\}$, et les fonctions δ_1 et ρ_1 suivantes :

δ_1			ρ_1	
q	$\delta_1(q, a)$	$\delta_1(q, b)$	q	$\rho_1(q)$
0	1	0	0	
1		2	1	0
2	3		2	0
3			3	1

 Épreuve d'option informatique MP 2018

\mathcal{A}_1 peut être dessiné comme suit :



- 7 – Soit $\mathcal{A} = (k, F, \delta, \rho)$ un AFDR. Pour tout $q \in Q_{\mathcal{A}}$, on note $\rho^j(q)$ pour $j \in \mathbb{N}$ l'application répétée j fois de la fonction ρ à q , c'est-à-dire $\underbrace{\rho(\rho(\dots(\rho(q))\dots))}_{j \text{ fois}}$. Par convention, pour tout état q , on note $\rho^0(q) = q$.

Montrer que pour tout $q \in Q_{\mathcal{A}}$, pour tout $\alpha \in \Sigma$, il existe $j \geq 0$ tel que $\delta(\rho^j(q), \alpha)$ est défini.

On dit qu'un AFDR $\mathcal{A} = (k, F, \delta, \rho)$ accepte un mot $u = u_1 \dots u_p \in \Sigma^*$ s'il existe une suite finie $q_0, q'_1, q_1, q'_2, q_2, \dots, q'_{p-1}, q_{p-1}, q'_p, q_p$ d'états de \mathcal{A} avec :

- $q_0 = 0$;
- pour tout $1 \leq i \leq p$, $q'_i = \rho^j(q_{i-1})$ avec $j \geq 0$ le plus petit entier tel que $\delta(\rho^j(q_{i-1}), u_i)$ est défini ;
- pour tout $1 \leq i \leq p$, $q_i = \delta(q'_i, u_i)$;
- $q_p \in F$.

Le langage accepté par un AFDR est l'ensemble des mots acceptés.

Ainsi, \mathcal{A}_1 accepte le mot « ababa », comme le montre la suite d'états parcourus

$$0, 0, 1, 1, 2, 2, 3, 1, 2, 2, 3.$$

On remarque qu'un AFDR dont la fonction de transition δ est définie partout peut être vu comme un automate fini déterministe classique, et que cet automate fini déterministe est *complet* (ce qui signifie précisément que sa fonction de transition est définie partout). En effet, les puissances non nulles de la fonction de repli ρ ne sont utilisées que si la fonction de transition n'est pas définie. On appellera un tel automate fini déterministe complet un *AFDC*.

- 8 – Construire (sans justification) un AFDC sur l'alphabet $\{a, b\}$ reconnaissant le même langage que \mathcal{A}_1 et ayant le même nombre d'états que \mathcal{A}_1 .

- 9 – Donner (sans justification) une description concise du langage reconnu par l'AFDR \mathcal{A}_1 .

 Épreuve d'option informatique MP 2018

- 10 – Programmer en Caml une fonction `copie_afdr : afdr -> afdr` qui renvoie un nouvel AFDR identique à l'AFDR fourni en entrée, mais ne partageant aucune donnée avec lui. On pourra utiliser la fonction standard `copie_vect : 'a vect -> 'a vect` qui renvoie un nouveau tableau contenant les mêmes éléments que les éléments d'entrée et s'exécute en un temps proportionnel au nombre d'éléments du tableau d'entrée.
- 11 – En s'inspirant de la réponse aux questions 7 et 8, et en utilisant la fonction `copie_afdr`, programmer une fonction `enleve_repli : afdr -> afdr` telle que, si \mathcal{A} est un AFDR, `enleve_repli A` renvoie un AFDC reconnaissant le même langage. On souhaite que cette fonction s'exécute en temps $O(k \times \lambda)$, où k est le nombre d'états de \mathcal{A} et λ est la taille de l'alphabet. Montrer que la fonction proposée a bien cette complexité.
- 12 – Étant donné un AFDC \mathcal{A} et un mot $u = u_1 \dots u_n$ sur Σ , proposer un algorithme (pas un programme Caml) en $O(n)$ pour calculer la liste triée des entiers i avec $1 \leq i \leq n$ tels que le préfixe $u_1 \dots u_i$ de u est accepté par \mathcal{A} .
- 13 – Implémenter en Caml l'algorithme de la question précédente : programmer une fonction `occurrences : afdr -> int list -> int list` telle que, si \mathcal{A} est un AFDC et *liste* une liste d'entiers j_1, \dots, j_n codant des symboles $\alpha_{j_1}, \dots, \alpha_{j_n}$ de l'alphabet Σ , `occurrences A liste` renvoie la liste des entiers i avec $1 \leq i \leq n$ tels que le mot $\alpha_{j_1} \dots \alpha_{j_i}$ est reconnu par \mathcal{A} . Quelle est la complexité de cette fonction en terme de la longueur n de la liste d'entiers, du nombre k d'états de l'automate \mathcal{A} et de λ ?

3 Automate de Knuth–Morris–Pratt

L'*automate de Knuth–Morris–Pratt* (ou *automate KMP*) associé à un motif $s = u_1 \dots u_k$ sur l'alphabet Σ est un AFDR $\mathcal{A}_s^{\text{KMP}} = (k', F, \delta, \rho)$ sur Σ avec :

- $k' = k + 1$.
- $F = \{k\}$.
- Pour tout $1 \leq i \leq k$, $\delta(i-1, u_i) = i$ et, pour tout $\alpha \in \Sigma \setminus \{u_1\}$, $\delta(0, \alpha) = 0$; aucune autre transition n'est définie.
- Pour tout $1 \leq i \leq k$, $\rho(i)$ est le plus grand entier $0 \leq j < i$ tel que $u_1 \dots u_j$ est un *suffixe* de $u_1 \dots u_i$.

On peut ainsi vérifier que l'automate \mathcal{A}_1 de la question précédente est l'automate KMP associé à « *aba* » sur l'alphabet $\Sigma = \{a, b\}$.

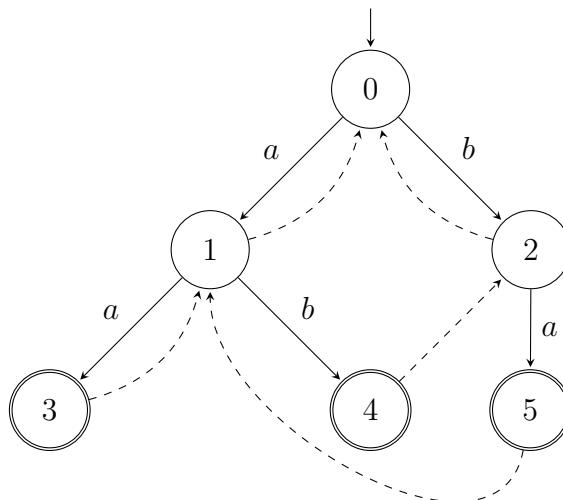
- 14 – Construire (sans justification) l'automate KMP associé à « *ababc* » sur l'alphabet $\{a, b, c\}$.

 Épreuve d'option informatique MP 2018

- 15 – Donner (sans justification) une description concise du langage reconnu par l'AFDR $\mathcal{A}_s^{\text{KMP}}$ pour un motif s arbitraire.
- 16 – Montrer que si $s = u_1 \dots u_k$ est un motif et $(k+1, F, \delta, \rho)$ l'automate KMP associé à s , alors pour tout $1 \leq i \leq k$, si $j \geq 0$ est le plus petit entier tel que $\delta(\rho^j(\rho(i-1)), u_i)$ est défini (qui existe d'après la question 7), alors $\rho(i) = \delta(\rho^j(\rho(i-1)), u_i)$.
- 17 – En utilisant la caractérisation de la question 16, programmer une fonction `automate_kmp : int list -> afdr` qui prend en entrée une liste d'entiers s codant une chaîne de caractères s et qui renvoie l'AFDR $\mathcal{A}_s^{\text{KMP}}$.
- 18 – Pour un motif $s = u_1 \dots u_k$ et pour tout $1 \leq i \leq k$, on note j_i le plus petit entier tel que $\delta(\rho^{j_i}(\rho(i-1)), u_i)$ est défini, comme à la question 16. Montrer que pour tout $1 \leq i \leq k$, $\rho(i) \leq \rho(i-1) + 1 - j_i$ et en déduire que $\sum_{i=1}^k j_i$ est en $O(k)$.
- 19 – Quelle est la complexité de la fonction `automate_kmp` en terme de la longueur k du motif s passé en argument et de λ ? Prouver cette affirmation.
- 20 – En s'appuyant sur les fonctions Caml `automate_kmp`, `enleve_repli` et `occurrences`, programmer une fonction Caml `recherche_kmp : string -> string -> int list` telle que, si s est un motif de longueur k et t une chaîne de caractères, `recherche_kmp s t` renvoie la liste des occurrences y de s dans t , triés par ordre croissant.
Quelle est la complexité de cette fonction en terme des longueurs k et n de s et t et de λ ? Comparer avec la complexité de la fonction `recherche_naive` obtenue en question 6.

4 Ensemble de motifs et automates à repli arborescents

On considère maintenant l'AFDR \mathcal{A}_2 suivant, sur l'alphabet $\Sigma = \{a, b\}$:



21 – Donner (sans justification) une description concise du langage reconnu par l'AFDR \mathcal{A}_2 .

22 – On considère l'alphabet $\Sigma = \{a, b, c\}$. Construire (sans justification) un AFDR dont le langage reconnu est l'ensemble des mots dont un suffixe est « *baa* », « *bab* » ou « *bc* ». Indication : on cherchera un AFDR tel que si $\delta(q, \alpha)$ est défini et q est non nul, alors $\delta(q, \alpha) > q$.

On fixe maintenant un ensemble fini S de motifs. L'ensemble des *occurrences de S dans une chaîne de caractères t* est l'ensemble des occurrences de chacun des motifs de S dans t .

23 – En utilisant la fonction `recherche_kmp`, programmer une fonction `recherche_dictionnaire_kmp` : `int list list -> int list -> int list` qui est telle que, si S est un ensemble fini de motifs codé comme une liste de motifs, et t une chaîne de caractères, `recherche_dictionnaire_kmp S t` renvoie la liste des occurrences y d'un motif $s \in S$ dans t . On n'impose pas d'ordre particulier sur cette liste, et on autorise les doublons.

Quelle est la complexité de cette fonction, en terme du nombre $|S|$ de motifs, de la longueur k maximale d'un motif, de λ et de la longueur n de t ?

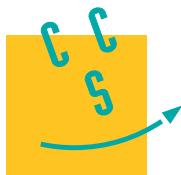
24 – En s'inspirant de l'automate \mathcal{A}_2 , de la réponse à la question 22 et de la partie 3, proposer une approche pour calculer efficacement les occurrences d'un ensemble fini S de motifs dans une chaîne de caractères t . On ne demande

Épreuve d'option informatique MP 2018

ni une formalisation complète de cette approche, ni une implémentation, mais une stratégie générale et les grandes étapes nécessaires à la résolution du problème. On pourra faire l'hypothèse, pour simplifier, que l'ensemble S ne contient pas deux mots qui sont préfixes l'un de l'autre.

Quelles difficultés sont à prévoir dans l'implémentation ? Quelle complexité peut-on espérer avec une telle approche, en terme du nombre $|S|$ de motifs, de la longueur k maximale d'un motif, de λ et de la longueur n de t ? Comparer avec la complexité obtenue en réponse à la question précédente.

FIN DE L'ÉPREUVE



CONCOURS CENTRALE-SUPÉLEC

Option informatique

MP

2018

4 heures

Calculatrices autorisées

Étude du jeu *Ricochet Robots*

À travers l'étude d'un jeu de société, ce sujet s'intéresse aux mouvements de robots, qui possèdent des capacités limitées de localisation. Avec le développement de la robotique, plusieurs problèmes de ce type font l'objet de nombreuses recherches : parcours minimum pour examiner une surface donnée, stratégies collectives avec plusieurs robots en interaction proche, nombre de robots nécessaires pour que tous les points d'une surface avec obstacles soient accessibles, etc.

Ce sujet porte sur la résolution de la situation pratique du jeu « Ricochet Robots » (*Rasende Roboter* pour la première édition en allemand) créé par Alex Randolph en 1999. Ce jeu se déroule sur un plateau de 16×16 cases, avec 4 robots et des murs. À chaque mouvement, un des robots se déplace, dans une des quatre directions, jusqu'à ce qu'il rencontre un obstacle (un mur ou autre robot). Le but est de trouver le nombre de coups minimal pour déplacer un robot particulier de son point de départ jusqu'à une case précise du plateau. Un exemple en 8 coups est donné figure 1.

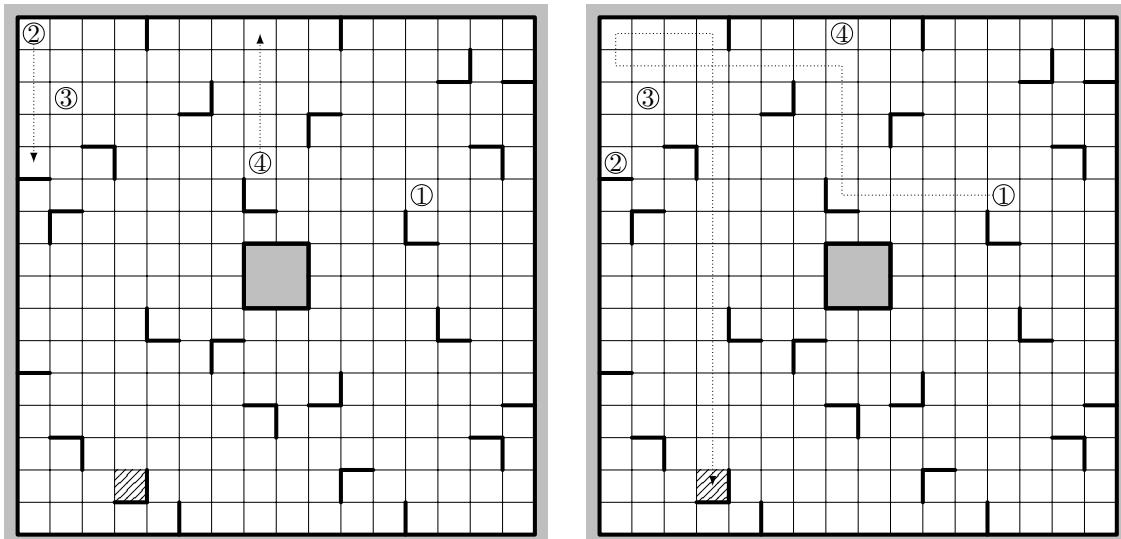


Figure 1 Le jeu des robots : le but est d'amener le robot 1 sur la case hachurée. À gauche : deux déplacements des robots 2 et 4 ; à droite : six déplacements du robot 1. Le jeu est résolu en 8 mouvements (solution optimale)

On rappelle la définition des fonctions suivantes, disponibles dans la bibliothèque standard de Caml :

- `copy_vect` : '`a vect` -> '`a vect` telle que l'appel `copy_vect v` renvoie un nouveau tableau contenant les valeurs contenues dans `v` ;
- `make_vect` : `int -> 'a -> 'a vect` telle que l'appel `make_vect n x` renvoie un nouveau tableau de longueur `n` initialisé avec des éléments égaux à `x` ;
- `make_matrix` : `int -> int -> 'a -> 'a vect vect` telle que l'appel `make_matrix p q x` renvoie une nouvelle matrice à `p` lignes et `q` colonnes initialisée avec des éléments égaux à `x`.

I Déplacement d'un robot dans une grille

On considère pour le moment une grille sans robots du jeu Ricochet Robots. Notons N le nombre de cases par ligne et colonne de la grille (16 dans le jeu originel). *Dans les fonctions demandées, on supposera que N est une variable globale*. On numérote chaque case par un couple (a, b) de $\llbracket 0, N - 1 \rrbracket^2$, correspondant à la ligne a et à la colonne b . On numérote également les lignes horizontales et verticales séparant les cases à l'aide d'un entier de $\llbracket 0, N \rrbracket$, de sorte que la case (a, b) est délimitée par les lignes horizontales a (au dessus) et $a + 1$ (en dessous), de même que par les lignes verticales b (à gauche) et $b + 1$ (à droite) (figure 2).

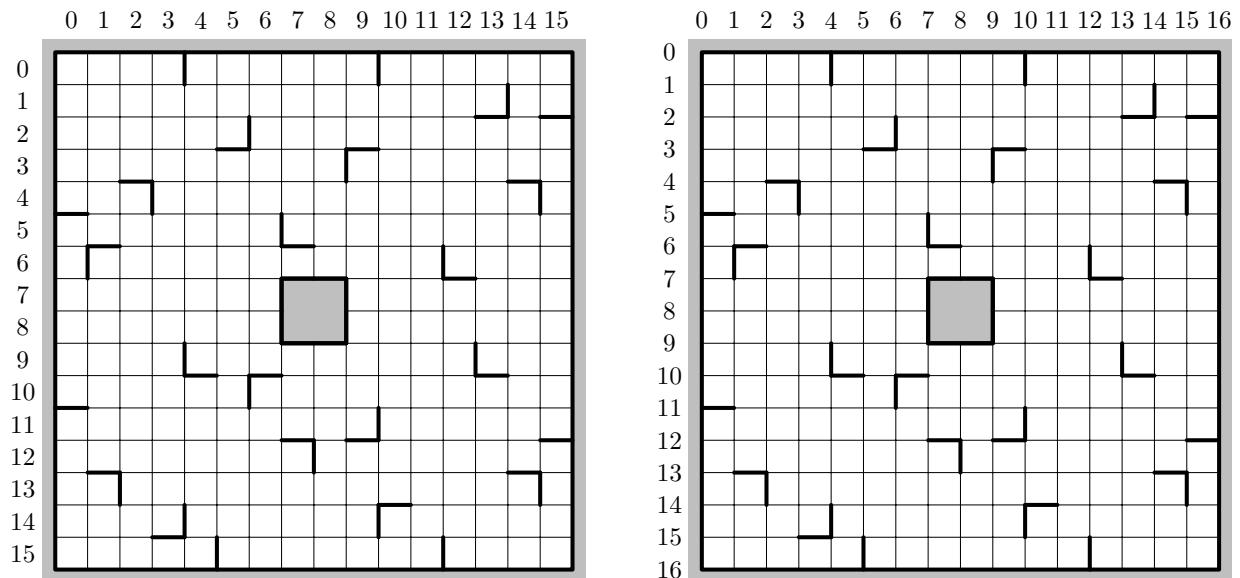


Figure 2 À gauche : numérotation des cases par ligne/colonne ; à droite : numérotation des lignes horizontales et verticales

Pour représenter en Caml la grille avec ses obstacles, on se donne deux tableaux (vecteurs) de taille N . Le premier contient les obstacles verticaux sur chacune des lignes, le second contient les obstacles horizontaux sur chacune des colonnes. Un obstacle est donné par le numéro de la ligne (verticale ou horizontale) auquel il appartient. Les obstacles sur une ligne (ou colonne) sont donnés sous la forme d'un tableau ordonné dans l'ordre croissant. Par exemple, la représentation du plateau de la figure 1 est donnée figure 3.

```
let obstacles_lignes = [| [|0; 4; 10; 16|]; [|0; 14; 16|]; [|0; 6; 16|];
                      [|0; 9; 16|]; [|0; 3; 15; 16|]; [|0; 7; 16|]; [|0; 1; 12; 16|]; [|0; 7; 9; 16|];
                      [|0; 7; 9; 16|]; [|0; 4; 13; 16|]; [|0; 6; 16|]; [|0; 10; 16|]; [|0; 8; 16|];
                      [|0; 2; 15; 16|]; [|0; 4; 10; 16|]; [|0; 5; 12; 16|] |];

```

```
let obstacles_colonnes = [| [|0; 5; 11; 16|]; [|0; 6; 13; 16|]; [|0; 4; 16|];
                           [|0; 15; 16|]; [|0; 10; 16|]; [|0; 3; 16|]; [|0; 10; 16|]; [|0; 6; 7; 9; 12; 16|];
                           [|0; 7; 9; 16|]; [|0; 3; 12; 16|]; [|0; 14; 16|]; [|0; 16|]; [|0; 7; 16|];
                           [|0; 2; 10; 16|]; [|0; 4; 13; 16|]; [|0; 2; 12; 16|] |];

```

Figure 3 Exemple de représentation en Caml

Notez que les bordures de la grille sont considérées comme des obstacles. Ainsi, les entiers 0 et N sont présents dans les tableaux associés à chaque ligne/colonne.

Q 1. Écrire une fonction `dichotomie a t` de signature `int -> int vect -> int` telle que si `t` est un tableau d'entiers strictement croissants et `a` un élément supérieur ou égal au premier élément du tableau et strictement inférieur au dernier, la fonction renvoie l'unique indice `i` tel que `t.(i) ≤ a < t.(i + 1)`. La fonction doit avoir une complexité logarithmique en la taille du tableau.

On considère un robot positionné en (a, b) , avec $0 \leq a, b < N$. Il peut se déplacer dans les quatre directions cardinales ouest/est/nord/sud représentées sur la figure 4.

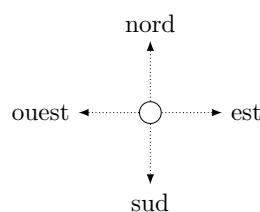


Figure 4 Déplacements cardinaux

Q 2. Écrire une fonction `déplacements_grille (a,b)` de signature `int * int -> (int * int) vect` fournissant les 4 cases atteintes par les déplacements en question, sous forme d'un tableau à 4 éléments

(ouest/est/nord/sud). Si le robot ne peut pas bouger dans une direction donnée (car il est contre un obstacle), on considérera que le résultat du déplacement dans cette direction est la case (a, b) elle-même. Les deux tableaux `obstacles_lignes` et `obstacles_colonnes` sont des variables globales.

Q 3. Écrire une fonction `matrice_deplacements ()`, de type `unit -> (int * int) vect vect vect` produisant une matrice `m` telle que `m.(a).(b)` contienne le vecteur des déplacements possibles pour un robot depuis la case (a, b) , et ce pour tous $0 \leq a, b < N$. Donner la complexité de création de la matrice.

On cherche maintenant à intégrer les positions d'autres robots dans le déplacement d'un robot. On utilise la fonction précédente pour créer une matrice `mat_deplacements` que l'on considérera comme globale.

Q 4. Écrire une fonction `modif t (a,b) (c,d)` de signature

```
(int * int) vect -> int * int -> int * int -> unit
```

telle que si `t` est le tableau de taille 4 donnant les déplacements ouest/est/nord/sud d'un robot placé en (a, b) dans la grille ne contenant pas d'autres robots, et (c, d) la position d'un autre robot, alors la fonction modifie si nécessaire le tableau `t` en prenant en compte le robot en (c, d) .

On s'intéresse maintenant au déplacement d'un robot situé en (a, b) dans la grille, avec d'autres robots éventuellement présents, dont les positions sont stockées dans une liste.

Q 5. Déduire des questions précédentes une fonction `deplacements_robots (a,b) q` de signature

```
int * int -> (int * int) list -> (int * int) vect
```

donnant les déplacements ouest/est/nord/sud d'un robot situé en (a, b) dans la grille, les positions des autres robots étant stockées dans la liste `q`. On ne modifiera pas la matrice `mat_deplacements` : on souhaite une copie modifiée de `mat_deplacements.(a).(b)`.

Q 6. Si on suppose que la solution optimale demande au plus k mouvements, une solution possible pour résoudre le jeu Ricochet Robots consiste à générer toutes les suites possibles de k déplacements. Avec 4 robots en tout, estimer la complexité d'une telle approche (on utilisera la notation O).

La suite du problème a pour objet de proposer une solution plus efficace pour la résolution du jeu Ricochet Robots.

II Quelques fonctions utilitaires

II.A – Une fonction de tri

Q 7. Écrire une fonction `insertion x q` de signature `'a -> 'a list -> 'a list` prenant en entrée un élément `x` et une liste `q` triée dans l'ordre croissant, et renvoyant une liste triée dans l'ordre croissant, constituée des éléments de `q` et `x`.

Q 8. En déduire une fonction `tri_insertion q` de signature `'a list -> 'a list` permettant de trier une liste dans l'ordre croissant.

Q 9. Rappeler la complexité de ce tri dans le pire et le meilleur cas. Que peut-on dire de la complexité si dans la liste `q`, tous les éléments excepté peut-être un sont dans l'ordre croissant ?

II.B – Quelques fonctions sur les listes

Q 10. Écrire une fonction `mem1 x q` de signature `'a -> ('a * 'b) list -> bool` testant l'appartenance d'un couple dont le premier élément est `x` dans la liste `q`.

Q 11. Écrire une fonction `assoc x q` de signature `'a -> ('a * 'b) list -> 'b` renvoyant, s'il existe, l'élément `y` du premier couple (x,y) appartenant à la liste `q`.

II.C – Implantation d'une structure de file

On rappelle que l'on peut facilement planter une structure de file à l'aide de deux listes : une des listes est utilisée pour rajouter des éléments, l'autre pour enlever des éléments. On définit ainsi le type

```
type 'a file = {mutable entree: 'a list; mutable sortie: 'a list};;
```

Lorsqu'on veut retirer un élément de la file alors que la deuxième liste est vide, on remplace celle-ci par la première, renversée.

On pourra utiliser les fonctions suivantes, qui permettent de manipuler une file ainsi définie :

<code>creer_file_vide : unit -> 'a file</code>	crée une file vide
<code>est_vide_file : 'a file -> bool</code>	teste si une file est vide
<code>enfiler : 'a file -> 'a -> unit</code>	ajoute un élément à une file
<code>defiler : 'a file -> 'a</code>	supprime l'élément en tête de file et le renvoie

On pourra supposer dans la suite que ces fonctions sont écrites de sorte que toute suite de p opérations `enfiler/defiler` à partir d'une file vide (ne produisant pas d'erreur) se fait en complexité $O(p)$.

III Tables de hachage

Dans l'optique de résoudre le problème du jeu des robots, nous allons travailler sur un graphe dont les sommets seront étiquetés par les positions des robots. Le nombre de sommets possibles étant élevé, il est nécessaire d'utiliser une structure de données adaptée pour travailler sur ce graphe. Nous allons donc réaliser une structure de dictionnaire permettant, en particulier, de tester facilement si un sommet a déjà été vu ou non et d'associer un sommet à chaque sommet découvert.

Une structure de dictionnaire est un ensemble de couples (clé, élément), les clés (nécessairement distinctes) appartenant à un même ensemble K , les éléments à un ensemble E . La structure doit garantir les opérations suivantes :

- recherche d'un élément connaissant sa clé ;
- ajout d'un couple (clé, élément) ;
- suppression d'un couple connaissant sa clé.

Une structure de dictionnaire peut-être réalisée à l'aide d'une table de hachage. Cette table est implantée dans un tableau de w listes (appelées *alvéoles*) de couples (clé, élément). Ce tableau est organisé de façon à ce que la liste d'indice i contienne tous les couples (k, e) tels que $h_w(k) = i$ où $h_w : K \rightarrow \llbracket 0, w - 1 \rrbracket$ s'appelle *fonction de hachage*. On appelle w la *largeur de la table* de hachage et $h_w(k)$ le *haché* de la clé k .

Ainsi pour rechercher ou supprimer l'élément de clé k , on commence par calculer son haché qui détermine l'alvéole adéquate et on est alors ramené à une action sur la liste correspondante. De même pour ajouter un nouvel élément au dictionnaire on l'ajoute à l'alvéole indiquée par le haché de sa clé.

III.A – Une famille de fonctions h_w

Nous commençons par nous doter d'une famille de fonctions h_w , pour les listes de couples de $\llbracket 0, N - 1 \rrbracket^2$. Un hachage naturel d'une liste comportant les couples $(a_i, b_i)_{0 \leq i < p}$ avec $0 \leq a_i, b_i < N - 1$ est donnée par :

$$P_w(N) = \left(\sum_{i=0}^{p-1} (a_i + b_i N) N^{2i} \right) \text{ modulo } w$$

Autrement dit, on évalue le polynôme dont les coefficients sont donnés par les a_i et b_i en N , et on ne considère que le reste dans la division euclidienne par w . On rappelle qu'on a supposé que N est une variable globale.

Q 12. Écrire une fonction récursive `hachage_liste w q` de signature `int -> (int * int) list -> int` calculant la quantité précédente.

III.B – Tables de hachage de largeur fixée

Dans cette sous-section, on fixe une largeur de hachage w . Un bon choix pour w serait par exemple un nombre premier ni trop petit, ni trop grand, comme 997. Pour les listes, on considérerait alors la fonction de hachage h_{997} donnée en Caml par `hachage_liste 997`. On définit en toute généralité le type suivant :

```
type ('a, 'b) table_hachage = {
  hache: 'a -> int;
  donnees: ('a * 'b) list vect;
  largeur: int};;
```

III.B.1) Implantation de la structure de dictionnaire

Q 13. Écrire une fonction `creer_table h w` de signature `('a -> int) -> int -> ('a, 'b) table_hachage` telle que `creer_table h w` renvoie une nouvelle table de hachage vide de largeur w munie de la fonction de hachage h .

Q 14. Écrire une fonction `recherche t k` de signature `('a, 'b) table_hachage -> 'a -> bool` renvoyant un booléen indiquant si la clé k est présente dans la table t . On pourra utiliser les fonctions de la partie II.

Q 15. Écrire une fonction `element t k` de signature `('a, 'b) table_hachage -> 'a -> 'b` renvoyant l'élément e associé à la clé k dans la table t , si cette clé est bien présente dans la table.

Q 16. Écrire une fonction `ajout t k e` de signature `('a, 'b) table_hachage -> 'a -> 'b -> unit` ajoutant l'entrée (k, e) à la table de hachage t . On n'effectuera aucun changement si la clé est déjà présente.

Q 17. Écrire enfin une fonction `suppression t k` de signature `('a, 'b) table_hachage -> 'a -> unit` supprimant l'entrée de la clé k dans la table t . On n'effectuera aucun changement si la clé n'est pas présente.

III.B.2) Étude de la complexité de la recherche d'un élément

Nous étudions ici la complexité de la recherche d'une clé dans une table de hachage. Dans le pire cas, toutes les clés sont hachées vers la même alvéole, ainsi la complexité de la recherche d'une clé dans une table de hachage n'est pas meilleure que la recherche dans une liste. Cependant, si la fonction de hachage h_w est bien choisie, on

peut espérer que les clés vont se répartir de façon apparemment aléatoire dans les alvéoles, ce qui donnera une complexité bien meilleure.

Nous faisons donc ici l'hypothèse de *hachage uniforme simple* : pour une clé donnée, la probabilité d'être hachée dans l'alvéole i est $1/w$, indépendante des autres clés. On note n le nombre de clés stockées dans la table et on appelle $\alpha = n/w$ le *facteur de remplissage* de la table. On suppose de plus, que le calcul du haché d'une clé se fait en temps constant.

Q 18. On se donne une clé k non présente dans la table. Montrer que l'espérance de la complexité de la recherche de k dans la table est un $O(1 + \alpha)$.

Q 19. On prend au hasard une clé présente dans la table ; toutes les clés sont équiprobables. Montrer qu'alors la recherche de la clé se fait en $O(1 + \alpha)$, en moyenne sur toutes les clés présentes.

III.C – Tables de hachage dynamique

Les deux questions précédentes montrent que l'on peut assurer une complexité moyenne constante pour la recherche dans une table de hachage, sous réserve que le facteur de remplissage α soit borné. Il en va de même des opérations d'insertion et de suppression, pour peu que les clés à ajouter/supprimer vérifient des hypothèses d'indépendance. Bien souvent, et cela va être le cas dans notre problème, on ne sait pas à l'avance quel sera le nombre de clés à stocker dans la table, et on préfère ne pas surestimer ce nombre pour garder un espace mémoire linéaire en le nombre de clés stockées. Ainsi, il est utile de faire varier la largeur w de la table de hachage : si le facteur de remplissage devient trop important, on réarrange la table sur une largeur plus grande (de même, on peut réduire la largeur de la table lorsque le facteur de remplissage devient petit). On parle alors de tables de hachage dynamiques pour ces tables à largeur variable.

À une table de hachage dynamique est associée une *famille de fonctions de hachage* (h_w). Par exemple, pour les listes de couples de $\llbracket 0, N - 1 \rrbracket^2$, la fonction `hachage_liste` précédemment écrite fournit une telle famille. On définit en toute généralité le type suivant :

```
type ('a,'b) table_dyn = {
    hache: int -> 'a -> int;
    mutable taille: int;
    mutable donnees: ('a * 'b) list vect;
    mutable largeur: int};;
```

On notera trois différences par rapport au type précédent :

- la fonction `hache` possède un paramètre supplémentaire qui est la largeur de hachage, elle correspond maintenant à la famille de fonctions de hachage (h_w) ;
- on a rendu les champs `donnees` et `largeur` modifiables ;
- un champ `taille` (modifiable) est rajouté, il doit à tout moment contenir le nombre de clés présentes dans la table.

Q 20. Écrire une fonction `creer_table_dyn h` permettant de créer une table de hachage dynamique initialement vide, avec la famille de fonctions de hachage `h` et la largeur initiale 1.

On admet avoir écrit deux fonctions `recherche_dyn t k` et `element_dyn t k`, variantes des fonctions `recherche` et `element` précédentes, basées sur le même principe. On va maintenant développer une stratégie pour maintenir à tout moment un facteur de remplissage borné.

Q 21. Écrire une fonction `rearrange_dyn t w2` prenant en entrée une table de hachage dynamique et une nouvelle largeur de hachage `w2`, qui réarrange la table sur une largeur `w2`. En supposant que le calcul des valeurs de hachage se fasse en temps constant, la complexité doit être en $O(n + w + w_2)$ où n est le nombre de clés présentes dans la table (sa taille), w est l'ancienne largeur de la table, w_2 la nouvelle.

Une stratégie heuristique simple pour garantir que le facteur de remplissage reste borné, tout en garantissant une bonne répartition des clés dans le cas des listes de couples à valeurs dans $\llbracket 0, N - 1 \rrbracket$ avec $N = 16$, est d'utiliser les puissances de 3 comme largeurs de hachage. Après ajout d'un élément à la table, si celle-ci est de taille strictement supérieure à trois fois sa largeur w , on la réarrange sur une largeur $w' = 3w$.

Q 22. Écrire une fonction `ajout_dyn t k e` ajoutant le couple (k, e) à la table de hachage (si la clé k n'est pas présente), en réarrangeant si nécessaire la table, en suivant le principe ci-dessus.

Dans l'hypothèse que chaque ajout se fait en temps $O(1 + \alpha)$, où α est le facteur de remplissage de la table, on peut montrer qu'une série de p ajouts dans une table initialement vide prend un temps $O(p)$.

On pourrait écrire de même une fonction de suppression dynamique, de sorte de maintenir un facteur de remplissage de la table borné, et qu'une série de p opérations licites d'insertion/suppression dans la table prenne un temps $O(p)$.

IV Résolution du jeu des robots

IV.A – Graphe orienté associé au jeu des robots

La résolution du jeu des robots peut se faire en traduisant le problème sous forme d'un graphe dans lequel chaque sommet représente une position des robots sur le plateau de jeu. On distingue le « robot principal » (celui que l'on veut amener sur une case donnée) et les autres robots. Ainsi, un sommet est représenté par le type suivant :

```
type sommet = {robot: int * int; autres_robots: (int * int) list};;
```

Pour chaque sommet, on impose que la liste `autres_robots` soit triée dans l'ordre croissant en suivant l'ordre lexicographique (l'ordre naturel pour les couples en Caml). Cet ordre est défini par $(a, b) \leq (a', b')$ si $a < a'$ ou si $a = a'$ et $b \leq b'$. Par exemple, Caml évalue l'expression $(2, 3) < (3, 0)$ en `true`.

Les arcs dans le graphe (orienté) sont définis naturellement : un sommet s est relié à un sommet s' si on peut passer de s à s' par un mouvement licite d'un des robots.

Q 23. Avec p robots en tout sur un plateau de taille $N \times N$, quel est le nombre possible de sommets ? Donner le nombre exact pour $p = 4$ et $N = 16$.

Q 24. Écrire une fonction `sommets_accessible s` de type `sommet -> sommet list` prenant en entrée un sommet et renvoyant la liste des sommets accessibles via s à partir du déplacement d'un des robots. S'il y a p robots en tout, la fonction renverra une liste de $4p$ sommets, certains sommets pouvant être égaux au sommet s : ils correspondent au mouvement d'un robot dans une direction où il est bloqué.

IV.B – Parcours en largeur : étude théorique

On se donne un graphe $G = (S, A)$ orienté, S dénote l'ensemble des sommets et A l'ensemble des arcs. On se donne un sommet $s_0 \in S$ du graphe et on considère l'algorithme 1 (parcours en largeur).

```

Entrées : un arbre  $G = (S, A)$ , orienté, un sommet de départ  $s_0$ 
Sortie : un tableau de booléens, un tableau de prédecesseurs
 $F \leftarrow \text{creer\_file\_vide}();$ 
Enfiler  $s_0$  dans  $F$ ;
 $b_{s_0} \leftarrow \text{vrai};$ 
 $b_s \leftarrow \text{faux}$  pour tout  $s \in S$ ; (* un tableau de booléens pour chaque sommet, tous faux *)
 $\pi_s \leftarrow s$  pour tout  $s \in S$ ; (* un tableau de prédecesseurs pour chaque sommet, initialement  $\pi[s] = s$  *)
tant que  $F$  est non vide faire
     $s \leftarrow \text{defiler}(F);$ 
    pour tout  $s'$  voisin de  $s$  tel que  $b_{s'}$  est faux faire
         $b_{s'} \leftarrow \text{vrai}; \pi_{s'} \leftarrow s$ ; enfiler  $s'$  dans  $F$ ;
    fin pour
fin tant que
renvoyer  $b, \pi$ 
```

Algorithme 1 Parcours en largeur

Q 25. Montrer que l'algorithme termine.

Q 26. Montrer que l'algorithme visite tous les sommets s du graphe pour lesquels il existe un chemin de s_0 à s .

Q 27. Pour un sommet s visité par l'algorithme (c'est-à-dire tel que b_s soit `vrai` à la fin de l'algorithme), expliquer à partir de π comment retrouver un chemin de s_0 à s .

Q 28. Montrer que ce chemin est un plus court chemin de s_0 à s .

Q 29. On suppose que les voisins sont implantés par liste d'adjacence, la complexité est linéaire en le nombre de voisins pour les parcourir. Les opérations de file et les opérations sur les tableaux π et b s'effectuent en temps constant, donner la complexité de l'algorithme en fonction de $|S|$ et $|A|$.

Un parcours en largeur du graphe associé au jeu permet donc de trouver une solution qui nécessite le minimum de déplacements des robots. La difficulté dans l'implantation de cet algorithme réside dans le grand nombre de sommets du graphe. Pour pallier cette difficulté, on remplace les tableaux b et π par un dictionnaire implanté dans une table de hachage dynamique. Les clés et les éléments du dictionnaires sont tous les deux des sommets du graphe tels que l'élément associé à la clé s soit π_s . On remplace ainsi le test de $b_{s'}$ par l'existence de la clé s' dans le dictionnaire.

• • • FIN • • •



ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

INFORMATIQUE

Jeudi 3 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, toutes indépendantes.

Partie I - Logique et calcul des propositions

Vous avez été sélectionné(e) pour participer au jeu "Cherchez les Clés du Paradis (CCP)". Le jeu se déroule en trois épreuves, au cours desquelles vous devez collecter des clés vertes. L'issue de chacune d'entre elles, vous passez à l'épreuve suivante en cas de succès et êtes éliminé(e) en cas d'échec.

I.1 - Première épreuve

Jean-Pierre Pendule, le célèbre animateur, vous accueille pour la première épreuve. Il vous explique la règle du jeu. Devant vous, deux boîtes et sur chacune d'entre elles une inscription. Chacune des boîtes contient soit une clé verte, soit une clé rouge. Vous devez choisir l'une des boîtes : si le résultat est une clé rouge, alors vous quittez le jeu, si c'est une clé verte vous êtes qualifié(e) pour l'épreuve suivante.

Jean-Pierre Pendule dévoile les inscriptions sur chacune des boîtes et vous affirme qu'elles sont soit vraies toutes les deux, soit fausses toutes les deux :

- sur la boîte 1, il est écrit : "Une au moins des deux boîtes contient une clé verte" ;
- sur la boîte 2, il est écrit : "Il y a une clé rouge dans l'autre boîte".

Dans toute cette partie, on note P_i la proposition affirmant qu'il y a une clé verte dans la boîte i .

- Q1.** Donner une formule de la logique des propositions représentant la phrase écrite sur la boîte 1.
- Q2.** Donner de même une formule de la logique des propositions pour l'inscription de la boîte 2.
- Q3.** Donner une formule représentant l'affirmation de l'animateur. Simplifier cette formule de sorte à n'obtenir qu'une seule occurrence de chaque P_i .
- Q4.** Quel choix devez-vous faire pour continuer le jeu à coup sûr ?

I.2 - Deuxième épreuve

Bravo, vous avez obtenu la première clé verte. Jean-Pierre Pendule vous félicite et vous annonce que cette première épreuve n'était qu'une mise en jambe. Avec les mêmes règles du jeu, l'animateur vous propose alors deux nouvelles boîtes portant les inscriptions suivantes :

- sur la boîte 1, il est écrit : "Il y a une clé rouge dans cette boîte, ou bien il y a une clé verte dans la boîte 2" ;
- sur la boîte 2, il est écrit : "Il y a une clé verte dans la boîte 1".

- Q5.** Donner une formule de la logique des propositions pour chaque affirmation.
- Q6.** Sachant qu'encore une fois les deux affirmations sont soit vraies toutes les deux, soit fausses toutes les deux, donner le contenu de chaque boîte. En déduire votre choix pour remporter la deuxième clé verte.

I.3 - Troisième épreuve

Le suspens est à son comble, vous voici arrivé(e) à la dernière épreuve. À votre grande surprise, Jean-Pierre Pendule vous dévoile une troisième boîte et vous explique les règles du jeu. Dans une des boîtes se cache la clé verte qui vous permet de remporter la victoire finale. Dans une autre boîte se cache une clé rouge qui vous fait tout perdre. La dernière boîte est vide. Encore une fois, chacune des boîtes porte une inscription :

- sur la boîte 1, il est écrit : "La boîte 3 est vide" ;
- sur la boîte 2, il est écrit : "La clé rouge est dans la boîte 1" ;
- sur la boîte 3, il est écrit : "Cette boîte est vide".

L'animateur affirme que l'inscription portée sur la boîte contenant la clé verte est vraie, celle portée par la boîte contenant la clé rouge est fausse. L'inscription affichée sur la boîte vide est aussi vraie.

- Q7.** Donner une formule de logique des propositions pour chaque inscription.
- Q8.** Donner une formule de logique des propositions synthétisant l'information que vous a apportée l'animateur.
- Q9.** En supposant que la clé verte est dans la boîte 2, montrer par l'absurde que l'on aboutit à une incohérence.
- Q10.** Donner alors la composition des trois boîtes.

Partie II - Automates

Un langage est régulier si et seulement si il est accepté par un automate fini (en particulier déterministe). Cependant, plusieurs automates peuvent accepter le même langage. L'objectif de cette partie est de montrer que tout langage reconnaissable L est reconnu par un unique (au renommage près des états) automate déterministe, tel que tout automate déterministe reconnaissant L a au moins autant d'états que lui. Cet unique automate est appelé automate minimal reconnaissant L .

II.1 - Définitions

Définition 1. Automate déterministe

Un automate déterministe est un quintuplet $= (Q, \Sigma, q_0, F, \delta)$, 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 définie sur $Q \times \Sigma$ tout entier.

Définition 2. Soit Σ un alphabet. Σ^* est l'ensemble des mots construits à partir de Σ , le mot vide et $|u|_a$ le nombre d'occurrences de la lettre $a \in \Sigma$ dans le mot u .

Définition 3. Résiduel d'un langage par rapport à un mot

Soient Σ un alphabet, $L \subseteq \Sigma^*$ un langage et $u \in \Sigma^*$. Le résiduel à gauche de L par rapport à u est le langage :

$$u^{-1}L = \{v \in \Sigma^*, uv \in L\}.$$

Q11. Pour $L = \{ab, ba, aab\}$, donner le résiduel de L par rapport à a .

Définissons alors une relation \sim_L sur Σ^* , dite congruence de Nerode, de la manière suivante : pour tous $u, v \in \Sigma^*$:

$$u \sim_L v \Leftrightarrow u^{-1}L = v^{-1}L.$$

Q12. Montrer que \sim_L est une relation d'équivalence et que : $\forall u, v, w \in \Sigma^* \quad (u \sim_L v) \Rightarrow (uw \sim_L vw)$.

Q13. Posons $L = \{u \in \Sigma^*, |u|_a = 0 \bmod 3\}$ le langage basé sur $\Sigma = \{a, b\}$, composé des mots de Σ^* ayant un nombre de a multiple de 3. Pour chacun des cas suivants, déterminer si les deux mots sont équivalents par \sim_L :

- (i). b et ab ,
- (ii). aba et bab ,
- (iii). $abbaba$ et aaa .

Définition 4. Soit $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe. La fonction de transition δ^* étendue aux mots est définie de manière récursive par :

- $\forall q \in Q \quad \delta^*(q, \epsilon) = q,$
- $\forall a \in \Sigma, \forall m \in \Sigma^*, \forall q \in Q \quad \delta^*(q, m.a) = \delta(\delta^*(q, m), a).$

Définition 5. Soit $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe. Pour $q \in Q$ et $G \subseteq Q$, on note :

$$q^{-1}G = \{u \in \Sigma^*, \delta^*(q, u) \in G\}.$$

$q^{-1}G$ est donc l'ensemble des mots qui correspondent à des chemins débutant en q et aboutissant dans un état de G .

Une relation d'équivalence \sim est également définie sur Q par :

$$p \sim q \Leftrightarrow p^{-1}F = q^{-1}F.$$

Si $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ est un automate déterministe acceptant un langage $L = q_0^{-1}F$ et si $q \in Q$ et $u \in \Sigma^*$ sont tels que $\delta^*(q_0, u) = q$ alors on peut montrer que $q^{-1}F = u^{-1}L$. Ainsi, la congruence de Nerode peut être utilisée pour définir un automate particulier, appelé automate minimal de L .

Définition 6. Automate minimal

Soit L un langage. L'automate minimal de L est défini par le quintuplet $\mathcal{A}_L = (Q_L, \Sigma, q_{L_0}, F_L, \delta_L)$, avec :

- $Q_L = \{u^{-1}L, u \in \Sigma^*\},$
- $q_{L_0} = \epsilon^{-1}L = L,$
- $F_L = \{u^{-1}L, u \in L\} = \{q \in Q_L, \delta_L(q, \epsilon) = q\},$
- $\forall q \in Q_L, \forall a \in \Sigma \quad \delta_L(q, a) = a^{-1}q.$

On admettra qu'un automate minimal définit bien un automate.

Q14. Montrer que l'automate minimal d'un langage régulier $L \subseteq \Sigma^*$ est un automate fini, c'est-à-dire un automate possédant un nombre fini d'états.

Définition 7. Automate accessible

Un automate déterministe $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ est accessible si pour tout $q \in Q$, il existe $u \in \Sigma^*$, tel que $\delta^*(q_0, u) = q$.

Définition 8. Automate réduit

Un automate déterministe $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ est réduit si pour tous $p, q \in Q$, $(p^{-1}F = q^{-1}F) \Rightarrow p = q$. Il est donc réduit si les langages acceptés depuis deux états distincts sont distincts, ou encore si chaque classe d'équivalence pour la relation \sim sur Q est un singleton.

Pour $L \subset \Sigma^*$, il est possible de montrer que l'automate minimal \mathcal{A}_L de L est accessible et réduit. Le paragraphe suivant s'intéresse à la construction de l'automate minimal d'un automate \mathcal{A} donné, exploitant cette propriété.

II.2 - Construction de l'automate minimal

Soit $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ un automate déterministe acceptant le langage $L \in \Sigma^*$. Trouver l'automate minimal \mathcal{A}_L de L revient à trouver un automate fini déterministe accessible et réduit équivalent.

Pour trouver un automate accessible, il suffit par exemple de visiter les états qui peuvent être atteints par δ depuis q_0 et d'éliminer les autres états. Il reste donc à trouver une méthode pour rendre \mathcal{A} réduit. Par définition de la relation \sim sur Q , \mathcal{A} est réduit si pour tout couple $(p, q) \in Q^2$, avec $p \neq q$, $p \not\sim q$.

En particulier, $p \not\sim q$ s'il existe $u \in \Sigma^*$, tel que $(p, u) \in F$ et $(q, u) \notin F$ ou $(q, u) \in F$ et $(p, u) \notin F$. On dit alors que u distingue p et q ou que le couple (p, q) est distingué par u .

L'algorithme 1 est un algorithme de réduction d'un automate utilisant ces notions. Dans la suite, N_k désigne l'ensemble des couples d'états de Q qui sont distingués par un mot de longueur k et qui ne sont distingués par aucun autre mot plus court.

Algorithme 1: Algorithme de recherche des états équivalents

Entrée : un automate déterministe $= (Q, \Sigma, q_0, F,)$

S sortie : les ensembles d'états équivalents

$k \leftarrow 0$

***** Initialisation *****

$N_0 \leftarrow \emptyset$

pour tous $p \in F$ et $q \in Q \setminus F$ **faire**

La paire (p,q) est distinguée.

$N_0 \leftarrow N_0 \cup \{(p, q)\}$

tant que $N_k = \emptyset$ **faire**

****Construction de N_{k+1} ****

$N_{k+1} \leftarrow \emptyset$

pour chaque paire $(p, q) \in N_k$ **faire**

pour chaque $a \in \Sigma$ **faire**

pour chaque $(r, s) \in Q^2$ **tel que** $(r, a) = p, (s, a) = q$ **faire**

si $(r, s) \notin \bigcup_{i=0}^k N_i$ **alors**

$N_{k+1} \leftarrow N_{k+1} \cup \{(r, s), (s, r)\}$

$k \leftarrow k + 1$

Q15. Montrer pourquoi, dans la phase d'initialisation, la paire (p, q) est distinguée.

Q16. Montrer pourquoi, si $N_i = \emptyset$ alors $\forall j > i, N_j = \emptyset$.

Soit alors $= (Q, \Sigma, q_0, F,)$ l'automate déterministe suivant :

— $Q = \{1, 2, 3, 4, 5, 6\}$

— $\Sigma = \{a, b\}$

— $q_0 = \{4\}$

— $F = \{2, 5\}$

	1	2	3	4	5	6
a	2	4	6	5	1	6
b	1	3	2	1	6	2

où (i, j) représente donc l'état atteint à partir de l'état i lorsque le symbole j est présenté.

Q17. Représenter graphiquement l'automate . La représentation suivra les contraintes suivantes :

— les états sont des cercles, le nom de l'état est écrit à l'intérieur du cercle ;

— les transitions sont représentées par des flèches partant de l'état de départ et pointant sur l'état d'arrivée. Le symbole définissant la transition est indiqué au milieu de la flèche ;

— l'état initial est signalé par une flèche sans étiquette pointant sur cet état ;

— les états finaux sont entourés d'un deuxième cercle, externe au premier.

- Q18.** Appliquer l'algorithme 1 pour trouver l'ensemble des états équivalents. Pour chaque itération k , la trace de l'algorithme sera donnée par une matrice carrée T de taille $|Q| \times |Q|$, avec $T(i, j) =$ longueur d'un chemin, s'il existe, qui distingue le couple (i, j) ($T(i, j)$ vide sinon). En déduire les classes d'équivalence des états de \mathcal{L} .

Un théorème, non détaillé ici, permet alors de projeter \mathcal{L} sur \mathcal{L} et de préciser états et transitions de cet automate minimal. Il permet en particulier de définir les états de \mathcal{L} comme étant les classes d'équivalence issues de l'algorithme précédent. Il permet également d'affirmer que si un état de \mathcal{L} correspond à une classe d'équivalence $[q]$ pour la relation \sim , alors la lecture d'un symbole $a \in \Sigma$ depuis cet état dans \mathcal{L} conduit à l'état correspondant à la classe $[(q, a)]$.

- Q19.** Représenter graphiquement l'automate minimal de la question précédente, avec ses états et ses transitions.

Partie III - Algorithmique et programmation

Nous proposons dans cette partie d'étudier une méthode de compression de données. L'algorithme proposé ici implémente plusieurs couches d'arrangement de données et de compression successives, utilisées dans l'ordre suivant pour la compression et l'ordre inverse pour la décompression :

- (i). Transformation de Burrows-Wheeler (BWT),
- (ii). Codage par plages (RLE),
- (iii). Codage de Huffman.

Définition 9. Soit Σ un alphabet de symboles, de cardinal $|\Sigma| = h$. On munit Σ d'une relation d'ordre \leq . Σ^k est l'ensemble des mots de longueur k construits à partir de Σ . Σ^k est muni d'une relation d'ordre lexicographique induite par la relation d'ordre \leq .

Pour $\mu \in \Sigma^k$, on note $|\mu| = k$ la taille de μ et $|\mu|_a$ le nombre d'occurrences de $a \in \Sigma$ dans μ .

Dans toute cette partie, lorsqu'il s'agira de coder une fonction CAML, un mot $\mu \in \Sigma^k$ sera représenté par une liste de caractères en CAML (char list).

Les paragraphes suivants étudient les algorithmes et propriétés de ces phases, chacun d'entre eux pouvant être abordé **de manière indépendante**.

III.1 - Transformation de Burrows-Wheeler (BWT)

Soit $\mu \in \Sigma^k$ un mot. La transformation BWT réalise une permutation des symboles de μ de sorte que les symboles identiques sont regroupés dans de longues séquences. Cette transformation n'effectue pas de compression, mais prépare donc à une compression plus efficace.

Dans la suite, nous étudions le codage et le décodage d'un mot transformé par cette opération.

– Phase de codage –

On rajoute à la fin de μ un marqueur de fin (par convention noté \mid , inférieur par \leq à tous les autres symboles de Σ). Dans toute la suite $\hat{\mu}$ désigne le mot auquel on a ajouté le symbole \mid .

- Q20.** Pour $\mu = turlututu$, construire une matrice M dont les lignes sont les différentes permutations circulaires successives du mot $\hat{\mu}$. Les permutations seront ici envisagées par décalage à droite des caractères.
- Q21.** Écrire une fonction récursive CAML circulaire : ' a list -> ' a list qui réalise une permutation à droite d'un mot μ donné en entrée.
- Q22.** Écrire une fonction CAML matrice_mot : ' a list -> ' a list list qui construit la matrice M à partir d'un mot passé en entrée. La valeur de retour est une liste de liste de symboles (une liste de mots). Cette fonction utilisera la fonction circulaire : ' a list -> ' a list.

Une permutation des lignes de M est alors effectuée, de sorte à classer les lignes par ordre lexicographique. On note $M' = P.M$ la matrice obtenue, P étant la matrice de permutation.

- Q23.** Donner les matrices P et M' dans le cas du mot $\hat{\mu}$, pour $\mu = turlututu$.

Pour construire la matrice de permutation P , il faut trier la liste des mots définissant M . La méthode de tri choisie ici est le tri par insertion.

- Q24.** Écrire une fonction récursive CAML tri : ' a list -> ' a list qui réalise le tri par insertion d'une liste d'éléments.
- Q25.** En déduire une fonction matrice_mot_triee : ' a list -> ' a list list qui construit M' à partir de M .

- Q26.** Pour $\mu \in \Sigma^k$, donner le nombre de comparaisons de symboles nécessaires au pire des cas, pour trier deux permutations circulaires du mot μ .

- Q27.** En déduire la complexité dans le pire des cas pour le tri des k permutations circulaires d'un mot $\mu \in \Sigma^k$ (exprimée en nombre de comparaisons de symboles).

La transformation BWT consiste alors à coder le mot μ par la dernière colonne de la matrice M' obtenue à l'aide de $\hat{\mu}$. On note $BWT(\mu)$ ce codage.

- Q28.** Écrire alors une fonction codageBWT : `char list -> char list` qui encode un mot passé en entrée. On utilisera une fonction récursive permettant de récupérer le dernier symbole de chacun des mots de M' . Donner le codage du mot $\mu = turlututu$.

- Phase de décodage -

Pour décoder un mot codé par BWT, il est nécessaire de reconstruire itérativement M' à partir de la seule donnée du mot codé $BWT(\mu)$. Par construction, $BWT(\mu)$ est la dernière colonne de M' .

On pose ici comme exemple $BWT(\mu) = edngvnea$.

- Q29.** Construire, à partir de la seule donnée de $BWT(\mu)$, la première colonne de M' . Justifier le principe de construction.

La dernière et la première colonne de M' donnent alors tous les sous-mots de longueur 2 du mot μ .

- Q30.** Proposer un algorithme permettant d'obtenir la deuxième colonne de M' . Donner cette deuxième colonne pour $BWT(\mu) = edngvnea$.

- Q31.** On dispose à l'itération n des $(n - 1)$ premières colonnes de M' et de sa dernière colonne. Proposer un principe algorithmique permettant de construire la n -ème colonne de M' .

- Q32.** En déduire un algorithme itératif permettant de reconstruire M' .

- Q33.** Quel décodage obtient-on pour le mot $BWT(\mu)$ proposé ?

III.2 - Codage par plages RLE [Informatique pour tous]

Le codage RLE (Run Length Coding), ou codage par plages, est une méthode de compression dont le principe est de remplacer dans une chaîne de symboles une sous-chaîne de symboles identiques par le

couple constitué du nombre de symboles identiques et du symbole lui-même. Par exemple, la chaîne "aaababb" est compressée en [(3,'a'),(1,'b'),(1,'a'),(2,'b')].

Q34. Proposer un type naturel Python pour la compression RLE, qui permet de représenter le résultat comme indiqué précédemment.

– Phase de codage –

On s'intéresse tout d'abord au codage RLE d'un mot.

Q35. Écrire une fonction itérative en Python `def RLE (mot)` : qui code un mot passé en entrée par codage RLE.

– Phase de décodage –

On s'intéresse maintenant au décodage d'une liste.

Q36. Écrire une fonction itérative en Python `def decodeRLE (codeRLE)` : qui décode une listecodeRLE issue du codage RLE d'un mot.

III.3 - Codage de Hu man [Informatique pour tous]

La dernière étape de l'algorithme proposé implémente le codage de Huffman, qui utilise la structure d'arbre binaire. Le principe est de coder un symbole de manière d'autant plus courte que son nombre d'occurrences dans le mot est élevé. L'arbre de Huffman se construit à l'aide de l'algorithme 2.

Algorithme 2: Codage de Huffman

Entrée : μ un mot de taille $|\mu|$

Sortie : $Huffman(\mu)$ le codage de Huffman de μ

pour $a \in \Sigma$ faire

si $|\mu|_a > 0$ alors

 créer un noeud $(a, |\mu|_a)$

$\mathcal{L} \leftarrow$ liste des noeuds dans l'ordre croissant des poids

$\mathcal{A} \leftarrow$ liste vide

tant que (*longueur*(\mathcal{L}) *longueur*(\mathcal{A})>1) faire

 (g, d) \leftarrow deux noeuds de plus faible poids parmi les 2 premiers noeuds de \mathcal{L} et les 2 premiers noeuds de \mathcal{A}

 Créer un noeud t

$n_t \leftarrow n_g + n_d$

gauche(t) $\leftarrow g$

 Coder la branche de t à g par 0

droite(t) $\leftarrow d$

 Coder la branche de t à d par 1

 Insérer t à la fin de \mathcal{A}

 Retirer g et d de \mathcal{L} ou de \mathcal{A}

$Huffman(\mu) \leftarrow \mathcal{A}$

Q37. Construire le codage de Huffman du mot $\mu = "turlututu"$ en utilisant l'algorithme 2. Vous expliciterez par des dessins d'arbres chacune des étapes de construction de $Huffman(\mu)$.

Q38. Quelle est la forme de l'arbre de Huffman dans un mot où tous les symboles ont le même nombre d'occurrences ?

FIN

**CONCOURS ARTS ET MÉTIERS ParisTech - ESTP - POLYTECH****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 5 exercices, totalement indépendants. Le langage à utiliser pour un exercice est indiqué à côté du numéro de l'exercice.

Un candidat pourra toujours admettre le résultat des questions qu'il n'a pas faites pour faire les questions suivantes.

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.

Tournez la page S.V.P.

Exercice 1 (Python)

On dit qu'une fonction *termine* si elle renvoie une valeur ou si elle lève une exception (par exemple `ZeroDivisionError`). Une fonction peut terminer ou continuer à calculer à l'infini. Ainsi, la fonction `fun1` (voir ci-dessous), pour un entier `n`, termine pour `n` inférieur ou égal à 10 (elle renvoie `None`) et ne termine pas pour `n` strictement plus grand que 10. En outre, cette fonction termine pour une chaîne de caractères `n` (en levant l'exception `TypeError`).

- Pour quelles valeurs de `n` dans \mathbb{Z} la fonction `fun2` termine-t-elle ? Même question pour `fun3`.

```
def fun1 (n):           def fun2 (n):           def fun3 (n):
    while n != 10 :     if n % 2 == 0:       S = 0
        n = n + 1         return 0
                           else :
                           while True:   S = S + n
                               n = n + 1   n = n - 2
                                         return S
```

- Détailler l'exécution de `fun3(10)`.
- Écrire en Python une fonction `ForEver(n)` qui ne termine jamais.

Le module `dis` permet, à partir du nom d'une fonction, de trouver quel est le bytecode Python qu'elle exécute, et d'analyser ce bytecode. On se demande quelles propriétés sur la fonction peuvent être déduites de cette analyse. Plus précisément, nous souhaitons répondre au problème suivant.

Problématique

Est-il possible d'écrire en Python une fonction `arret`, qui termine toujours, et telle que `arret(f, x)` renvoie `True` si le calcul de `f(x)` termine et `False` sinon ?

- Dans cette question, nous supposons qu'une telle fonction `arret` existe.
 - Écrire en Python une fonction `strange(f, x)` qui termine si et seulement si le calcul de `f(x)` ne termine pas.
 - Écrire en Python une fonction `paradox(f)` qui termine si et seulement si le calcul de `f(f)` ne termine pas.
- Le calcul de `paradox(paradox)` termine-t-il ? Qu'en déduire quant à l'existence de `arret` ?

Exercice 2 (Caml)

Tout entier naturel non nul n s'écrit de manière unique $n = 2^v k$ avec v un entier naturel (qui peut valoir zéro) et k un entier naturel impair. Dans la suite de cet exercice, nous appelons valuation de n l'entier v et résidu de n l'entier k . Par convention, le résidu de zéro est zéro.

1. Expliquer succinctement comment, à partir de l'écriture en base deux de $n \in \mathbb{N}^*$, on peut lire la valuation et le résidu de n .
2. L'entier 192 s'écrit en base deux 11000000. Donner sa valuation et son résidu.
3. Écrire en Caml une fonction `residu : int -> int` qui prend en argument un entier positif **ou nul** et renvoie son résidu.

Pour calculer le pgcd (plus grand commun diviseur) de deux entiers, nous pouvons utiliser l'algorithme des soustractions successives décrit ci-après.

- 1 Entrées : deux entiers naturels non nuls a et b
- 2 Tant que b est non nul :
- 3 Remplacer b par $|a - b|$.
- 4 Remplacer a par le minimum de a et de l'ancienne valeur de b .
- 5 Fin du “Tant que”.
- 6 Renvoyer a

4. Écrire en Caml une fonction `pgcd1 : int -> int -> int` qui calcule le pgcd de deux entiers naturels non nuls en utilisant cet algorithme et pas un autre. Cette fonction ne doit pas être récursive ni faire appel à une ou des fonctions auxiliaires récursives.
5. Écrire en Caml une fonction **réursive** `pgcd2 : int -> int -> int` qui calcule le pgcd de deux entiers naturels non nuls en utilisant la méthode des soustractions successives.
6. Estimer (en justifiant) la complexité de cet algorithme en fonction de $n = \max(a, b)$.

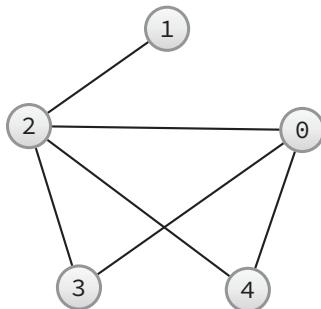
La méthode chinoise, mentionnée dans *Les Neuf Chapitres sur l'art mathématique* écrit aux débuts de la dynastie Han, consiste à remplacer la ligne 3 par la ligne suivante :

- 3 Remplacer b par le résidu de $|a - b|$.

On admet que cette méthode permet de calculer le pgcd de a et b si a ou b est **impair**. Ce résultat peut notamment être utilisé pour répondre à la question suivante.

7. Que calcule la méthode chinoise lorsque a et b sont **pairs** ? Le démontrer la-coniquement.
Indication : On peut distinguer le cas $a = b$ du cas $a \neq b$.
8. Estimer (en justifiant) la complexité de cette méthode en fonction de $n = \max(a, b)$, dans le cas où a et b sont tous les deux impairs.
9. En déduire une fonction `pgcd_chinois` : `int -> int -> int` qui calcule le pgcd de deux entiers (pairs ou impairs) avec une complexité du même ordre de grandeur que la complexité calculée la question précédente. Justifier la complexité de `pgcd_chinois`.

Exercice 3 (Caml)



1. Écrire la matrice d'adjacence du graphe ci-dessus.
2. Écrire en Caml une fonction `chemin` : `int vect vect -> int list -> bool` qui prend en entrée la matrice d'adjacence d'un graphe et un chemin (une liste de sommets du graphe) et qui vérifie si ce chemin est possible dans le graphe. Par exemple, sur le graphe ci-dessus, avec le chemin `[2;1;0;4]` la fonction `chemin` doit renvoyer `false` car les sommets 1 et 0 ne sont pas connectés. Avec le chemin `[1;2;3]` la fonction `chemin` doit renvoyer `true` car les sommets 1 et 2 sont connectés, ainsi que les sommets 2 et 3.

Exercice 4 (SQL)

Nous nous intéressons à une base de données des zoos qui contient deux tables. La première table, `zoos`, a quatre colonnes dont `id`, un identifiant unique pour chaque zoo. Quelques lignes sont données ci-après.

id	nom	pays	continent
FR42	Zoo de La Flèche	France	Europe
RU12	Parc zoologique de Novossibirsk	Russie	Asie
RU5	Parc zoologique de Saint-Pétersbourg	Russie	Europe
:	:	:	:

La seconde table, animaux, a 6 colonnes, notamment un identifiant unique pour chaque animal (id) et l'identifiant du zoo qui héberge l'animal (zoo).

id	nom	espece	sexe	naissance	zoo
ke860	Kaiko	Chameau	F	2013	FR42
ic431	Jeffrey	Python royal	M	2016	RU12
gz599	Antaeus	Annaconda vert	M	2016	RU12
:	:	:	:	:	:

Les questions suivantes demandent d'écrire des requêtes SQL. À chaque fois quelques lignes de la table attendue sont données en exemple .

- Écrire une requête SQL renvoyant la table des chamelles (chameaux femelles).

id	nom	naissance	zoo
ke860	Kaiko	2013	FR42
md375	Aimy	2012	CG01
:	:	:	:

- Écrire une requête SQL renvoyant la table des bonobos mâles vivant en Asie.

id	nom	naissance	zoo
yv919	Finn	2008	CN33
qv139	Proteus	2013	KR08
:	:	:	:

- Écrire une requête renvoyant la liste des pays ayant des zoos sur plusieurs continents.

pays
Russie
:

Exercice 5 (Caml)

Une *formule positive* est une formule propositionnelle n'utilisant comme connecteurs logiques que “ou” et “et”. Ces connecteurs sont notés, respectivement, \vee et \wedge . On représente les formules positives en Caml par le type suivant :

```
type fp = OU of fp * fp | ET of fp * fp | VAR of string ;;
```

1. Considérons la formule " $X \vee (Y \wedge Z)$ " et notons-la φ_0 .

- (a) Dessiner l'arbre correspondant à φ_0 .
- (b) Écrire φ_0 en Caml en utilisant le type fp.

On définit par récurrence les *disjonctions de variables propositionnelles* (DVP) ainsi :

- Si X est une variable propositionnelle, alors X est une DVP.
 - Si φ et ψ sont des DVP alors $\varphi \vee \psi$ est aussi une DVP.
2. Écrire en Caml une fonction dvp : fp -> bool prenant en argument une formule positive f et renvoyant true si et seulement si f est une DVP.

On appelle *forme normale conjonctive positive* (FNCP) une conjonction de DVP. Ainsi $(X \vee U) \wedge (X \vee Y \vee Z) \wedge T$ est une forme normale conjonctive positive, mais pas $(X \wedge U) \vee (X \wedge (Y \vee Z))$. Plus précisément, on définit les FNCP par récurrence comme suit :

- Si φ est une DVP alors φ est une FNCP.
 - Si φ et ψ sont des FNCP, alors $\varphi \wedge \psi$ aussi.
3. Écrire en Caml une fonction fncp de type fp -> bool prenant en argument une formule positive f et renvoyant true si et seulement si f est une FNCP.

On définit la fonction norm qui, étant donné une formule positive f, renvoie une FNCP logiquement équivalente à f.

```
let rec norm f = match f with
  VAR _ -> f
  | ET(a,b) -> ET (norm a, norm b)
  | OU(ET(a,b),c) -> ET( norm (OU(a,c)) , norm (OU(b,c)) )
  | OU(c,ET(a,b)) -> ET( norm (OU(a,c)) , norm (OU(b,c)) )
  | OU (a,b) -> let c = OU( norm a, norm b) in
    if f=c
    then f
    else norm c;;
```

0	
1	
2	
3	
4	
5	
6	
7	
8	

4. Expliquer brièvement pourquoi si f est une FNCP alors norm f termine et renvoie f.
5. Montrer que si norm f termine, alors elle renvoie une FNCP.

6. Exhiber, sans démonstration, une fonction simple λ de l'ensemble des formules dans \mathbb{N} telle que pour toutes formules a, b, c et d :

- $\lambda(a \vee b) = \lambda(a \wedge b) > \max(\lambda(a), \lambda(b))$,
- $\lambda((a \wedge b) \vee c) = \lambda(c \vee (a \wedge b)) > \max(\lambda(a \vee c), \lambda(b \vee c))$,
- Si $\lambda(a) \geq \lambda(c)$ et $\lambda(b) \geq \lambda(d)$ alors $\lambda(a \vee b) \geq \lambda(c \vee d)$.

On introduit la fonction μ de l'ensemble des formules dans \mathbb{N} . $\mu(\varphi)$ est la profondeur minimale d'un nœud ET dans la formule φ . Plus précisément, μ est définie comme suit :

- $\mu(X) = 0$ si X est une variable propositionnelle,
- $\mu(\varphi \wedge \psi) = 1$ pour toutes formules φ et ψ ,
- $\mu(\varphi \vee \psi) = \min(\mu(\varphi), \mu(\psi))$ pour toutes formules φ et ψ .

7. On considère une formule f telle que :

- f correspond au cas de filtrage (*pattern matching* en anglais) de la ligne 5 mais ne correspond à aucun des cas des lignes 1 à 4,
- Le calcul de c à la ligne 5 termine.

- (a) Montrer que si $f \neq c$ alors $\mu(f) > \mu(c)$.
- (b) Montrer que $\lambda(f) \geq \lambda(c)$.

8. Démontrer soigneusement que `norm` termine sur toutes les formules positives.



Samedi 7 Avril 2018

OPTION : SCIENCES DU NUMERIQUE

MP / PC / PSI / PT / TSI

Durée : 2 Heures

Condition(s) particulière(s)

Calculatrice interdite

Remettre le QCM avec vos copies d'examen

Consignes Python

Tout code doit être écrit dans le langage Python.

- Tout code Python non indenté ne sera pas corrigé.
- Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué ci-dessous.
- Vous pouvez écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.

Fonctions et méthodes autorisées

Vous pouvez utiliser la fonction range :

```
>>> for i in range(10):
...:     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...:     print(i, end=' ')
5 6 7 8 9
```

Sur les listes, vous pouvez utiliser la méthode append et la fonction len :

```
>>> help ( list . append )
Help on method_descriptor : append (...)
L. append ( object ) -> None -- append object to end of L

>>> help (len)
Help on built-in function len in module builtins : len (...)
len ( object )
Return the number of items of a sequence or collection .
```

Les matrices sont représentées par des listes de listes comme dans l'exemple ci-dessous :

```
>>> M1 = [[1, 10, 3, 0, 3, 10, 1],
...:         [1, 0, 1, 8, 1, 0, 1],
...:         [10, 9, 4, 1, 4, 9, 10],
...:         [10, 3, 7, 1, 7, 3, 10],
...:         [7, 8, 5, 1, 5, 8, 7]]
```

A) Pile ou file ...

On ajoute, dans cet ordre, les valeurs A, B, C, D, E et F à une structure linéaire vide. Pour chacun des ordres de sortie suivant, indiquer si la structure en question peut être : une pile, une file (ce peut être les deux), ou aucune des deux (ni une pile, ni une file).

- A B C D E F
- D E C B F A
- B D E F A C
- F E D C B A

B) Tri fusion ...

1. Écrire la fonction **partition(L)** qui sépare une liste **L** en deux (nouvelles) listes de longueurs quasi identiques (à 1 près) : une moitié dans chaque liste.

Exemples d'application:

```
>>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])

>>> partition([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Écrire la fonction **merge(L1, L2)** qui fusionne deux listes triées en ordre croissant **L1** et **L2** en une seule nouvelle liste triée.

Exemple d'application:

```
>>> merge([1,5,8], [2,3,4,8])
[1, 2, 3, 4, 5, 8, 8]
```

3. Pour trier une liste **L**, on procède (récursevement) de la façon suivante :

- Une liste de longueur < 2 est triée.
- Une liste de longueur ≥ 2 :
 - on partitionne la liste **L** en deux sous-listes **L1** et **L2** de longueurs quasi identiques (à 1 près) ;
 - puis, on trie récursevement les deux listes **L1** et **L2**;
 - enfin, on fusionne les listes **L1** et **L2** en une liste triée.

Utiliser les deux fonctions précédentes (quelles soient écrites ou non) pour écrire la fonction **mergesort(L)** qui trie en ordre croissant une liste **L** (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

Exemple d'application:

```
>>> mergesort([5,3,2,8,7,1,5,4,0,6,1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

C) Maximum Gap ...

Pour cet exercice, on définit le *gap* (écart) d'une liste comme étant l'écart maximum entre deux valeurs de la liste. Par exemple, dans la matrice ci-dessous le *gap* de la première ligne est 13.

Mat1

1	10	3	0	-3	2	8
-1	0	1	8	5	0	-4
10	9	14	1	4	-5	1
10	-3	7	11	6	3	0
7	8	-5	1	5	4	10

Écrire la fonction **maxGapMatrix(M)** qui retourne le gap maximum des lignes d'une matrice **M** (que l'on supposera non vide).

Exemple d'application avec la matrice Mat1 ci-dessus:

```
>>> maxGapMatrix(Mat1)
19
```

En effet le gap maximum est celui de la ligne du milieu (19 = 14 - (-5)).

D) Matrices : symétrique ...

La matrice transposée d'une matrice **A** est la matrice **A^T**, obtenue en échangeant les lignes et les colonnes de **A**.

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \text{ alors } A^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Une matrice symétrique est une matrice carrée qui est égale à sa propre transposée. Écrire la fonction **is_symmetric(A)** qui teste si une matrice **A** non vide est symétrique.

QCM

Ce qcm est un peu particulier: chaque question peut comporter 0, 1, ou plusieurs bonnes réponses! Pour gagner des points, il faut répondre correctement (les mauvaises réponses feront perdre des points).

La majorité des questions suivantes sont formulées au singulier ou au pluriel, par commodité pour la grammaire française, mais sans corrélation directe avec le nombre de réponses correctes.

? Parmi les termes suivants, lequel a un lien direct avec les réseaux informatiques ?

- UPS
- TCP
- ICBM
- DHCP

? Le noyau de Linux a été initialement écrit par

- Richard Matthew Stallman
- Linus Torvalds
- Andrew Tannenbaum

? Informaticien célèbre

- Alan Turing
- Edgar J. Hoover
- Alan Parker
- Edsger W. Dijkstra
- Donald Trump
- Donald E. Knuth

? Laquelle des affirmations suivantes est vraie

- L'électronique numérique fonctionne essentiellement selon trois états: 0 (pas de courant), 1 (courant) ou haute impédance
- Certains robots mobiles utilisent pour se déplacer des moteurs pas-à-pas
- La monnaie virtuelle bitcoin est générée par des «miners» virtuels hébergés par le jeu MineCraft
- Android est un système d'exploitation pour téléphones, tablettes, et ordinateurs

? Que fait le code suivant

```
def fubar(n, k):  
    if k * k >= n:  
        return n  
    if n%k:  
        return fubar(n, k+2)  
    else:  
        return k  
  
def fubar(n):  
    if n%2:  
        return fubar(n, 3)  
    else:  
        return 2
```

- Il affiche une décomposition en facteurs premiers
- Il calcule une factorielle
- Il détermine le plus petit facteur non trivial d'un nombre
- Il calcule le PGCD de deux nombres

? La technologie Java appartient à

- Sun MicroSystems
- Android
- Microsoft
- Oracle

? Noms de microprocesseurs

- sparc
- tinkerbell
- coldwater
- arm
- head&shoulders
- mip-mips
- mips

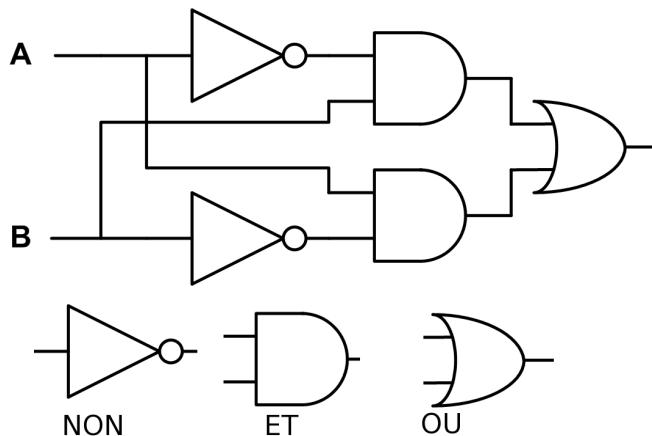
? L'acronyme GAFA englobe les plus grosses compagnies informatiques liées à internet, il comprend

- Google
- Android
- Facetime
- Twitter
- Apple

?

Certaines calculatrices ont un mode BCD (binnaire codé décimal). Le principe: on découpe un octet en deux groupes de quatre bits, et chaque groupe code un chiffre en base 10, ce qui évite d'avoir à convertir pour l'affichage, et permet d'avoir un nombre de chiffres "exact" en représentation. Avec ce schéma, la valeur 85 donnera en binaire

- 10000101
- 10001001
- 01010101
- 10000100



?

Le schéma précédent combine les entrées A et B et forme

- La négation du "et" de A et B
- Le ou exclusif de A et B
- La retenue de la somme de A et B
- La somme de A et B, en oubliant la retenue

CAPES de mathématiques Option Informatique–Session 2018

Le sujet est constitué de deux problèmes indépendants.

Problème n° 1 : suite de Lucas

Rappels et notations. Pour x un nombre réel, il existe un plus grand entier inférieur ou égal à x , appelé *plancher* de x ou *partie entière* de x . Ce nombre entier est noté $\lfloor x \rfloor$. Il existe un plus petit entier supérieur ou égal à x , appelé *plafond* de x .

Les fonctions `ceil` et `floor` du module `math` de Python calculent respectivement le plafond et le plancher d'un nombre flottant x : Ainsi `ceil(1.24)` vaut 2 et `floor(1.24)` vaut 1.

Soit a un nombre réel strictement positif. On désigne par \log_a la fonction logarithme en base a : si x est un nombre réel strictement positif,

$$\log_a x = \frac{\ln x}{\ln a},$$

où \ln désigne la fonction logarithme néperien.

Dans ce problème, on étudie plusieurs algorithmes de calcul des nombres de Lucas.

Ces nombres sont les termes de la suite $(L_n)_{n \geq 0}$ définie par les relations suivantes :

$$\begin{cases} L_0 = 2, \\ L_1 = 1, \\ L_n = L_{n-1} + L_{n-2} \text{ pour } n \geq 2. \end{cases}$$

- I.** Calculer L_2, L_3, L_4, L_5, L_6 et L_7 .
- II.** Montrer que pour tout $n \geq 0$, L_n est un entier naturel.
- III.** On considère l'équation $x^2 - x - 1 = 0$.
 - 1.** Montrer que cette équation possède deux solutions, l'une positive que l'on note ϕ , l'autre négative que l'on note $\hat{\phi}$. Des valeurs approchées à 10^{-4} près de ces deux nombres sont $\phi \approx 1,6180$ et $\hat{\phi} \approx -0,6180$.
 - 2.** Justifier que

$$\phi + 1 = \phi^2, \quad \hat{\phi} + 1 = \hat{\phi}^2, \quad \phi + \hat{\phi} = 1, \quad \phi\hat{\phi} = -1.$$

- IV.** Montrer que pour tout entier naturel n , $L_n = \phi^n + \hat{\phi}^n$. On pourra raisonner par récurrence.
- V.** On donne cette valeur approchée du logarithme en base 10 de ϕ :

$$\log_{10} \phi = \frac{\ln \phi}{\ln 10} \approx 0,2090.$$

Montrer que pour tout entier naturel p ,

$$n \geq 5p \implies L_n \geq 10^p.$$

- VI.** 1. On propose la fonction suivante pour calculer le n -ième nombre de Lucas.
On rappelle que `**` est l'opérateur Python d'élévation à la puissance.

```

0 from math import *
1
2 def lucas1(n):
3     if n == 0:
4         return 2
5     phi = (1+sqrt(5))/2
6     phi2 = (1-sqrt(5))/2
7     return phi**n + phi2**n

```

L'évaluation de `[lucas1(n) for n in range(8)]` renvoie la liste

`[2, 1.0, 3.0, 4.0, 7.000000000000001, 11.000000000000002,
18.000000000000004, 29.000000000000007].`

Pourquoi ne s'agit-il pas d'une liste d'entiers ?

2. On propose maintenant la fonction suivante pour calculer le n -ième nombre de Lucas.

```

0 from math import *
1
2 def lucas2(n):
3     if n == 0:
4         return 2
5     phi = (1+sqrt(5))/2
6     if n%2 == 0:
7         return ceil(phi**n)
8     else:
9         return floor(phi**n)

```

L'évaluation de `[lucas2(n) for n in range(8)]` renvoie la liste

`[2, 1, 3, 4, 7, 11, 18, 29].`

Expliquer le choix de ces fonctions en lignes 6 à 9 et démontrer que, si les calculs en flottants sont exacts, `lucas2(n)` calcule bien L_n .

3. Un calcul exact montre que $L_{36} = 33385282$, mais `lucas2(36)` renvoie la valeur 33385283. Comment expliquez-vous cela ?
- VII.** On souhaite écrire une nouvelle fonction de calcul des termes de la suite de Lucas. Pour éviter tout problème lié au calcul avec des flottants, on souhaite ne travailler qu'avec des entiers, sur lesquels Python calcule de manière exacte.

1. Recopier et compléter la fonction suivante, qui renvoie la valeur de L_n .

```

0 def lucas3(n):
1     if n == 0:
2         return 2
3     if n == 1:
4         return 1
5     a,b = (2,1)
6     for i in range(n):
7         a,b = (... , ...)
8     return ...

```

2. Déterminer un invariant de boucle qui précise, en fonction de la valeur de i , les valeurs des variables a et b avant et après l'exécution de la ligne 7 et en déduire que `lucas3` renvoie un résultat exact.

3. Pour $n \geq 2$, combien d'additions sont effectuées par `lucas3(n)` ?

VIII. 1. Démontrer que, pour tout entier $n \geq 1$,

$$L_n^2 - L_{n+1}L_{n-1} = 5(-1)^n.$$

2. Démontrer que, pour tout entier $n \geq 1$,

$$\begin{cases} L_{2n} &= L_n^2 - 2(-1)^n, \\ L_{2n+1} &= L_n L_{n+1} - (-1)^n. \end{cases}$$

IX. 1. Si k est un entier, que calcule l'expression Python suivante : `1 - 2*(k % 2)` ?

On rappelle que l'opérateur `%` calcule le reste dans la division entière : `7 % 3` vaut `1`; `8 % 3` vaut `2` et `9 % 3` vaut `0`.

2. Recopier et compléter la fonction récursive suivante, de sorte que `lucas4(n)` renvoie le couple (L_n, L_{n+1}) .

```

0 def lucas4(n):
1     if n == 0:
2         return (2,1)
3     if n == 1:
4         return (1,3)
5     k = n // 2
6     u = 1 - 2*(k % 2)
7     a,b = lucas4(....)
8     if n % 2 == 0:
9         return (..., ...)
10    else:
11        return (..., ...)
```

3. Pour tout entier $n \geq 2$, exprimer en fonction de n le nombre d'appels récursifs que réalise `lucas4(n)`.

X. Pour n un entier naturel, on considère le vecteur colonne de \mathbb{R}^2 défini par

$$V_n = \begin{pmatrix} L_n \\ L_{n+1} \end{pmatrix}.$$

On considère également la matrice $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Pour tout entier $n \geq 1$, exprimer V_n en fonction de A et de V_{n-1} , puis exprimer V_n en fonction de A , de n et de V_0 . On représente dans la suite une matrice carrée $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ par la liste de deux listes

$$[[a,b], [c,d]].$$

- XI.** 1. Écrire en Python une fonction `prodMat(M1,M2)` qui prend en arguments deux matrices 2×2 représentées par des listes comme indiqué ci-dessus, et qui renvoie la liste qui représente la matrice produit $M1 \times M2$.

2. Combien l'appel `prodMat(M1,M2)` effectue-t-il d'additions ? de multiplications ?
 3. On considère la fonction (naïve) d'élévation d'une matrice à une puissance entière suivante.

```

0 def puissanceMat(M,p):
1     R = [[1,0],[0,1]]
2     for i in range(p):
3         R = prodMat(R,M)
4     return R

```

Combien l'appel `puissanceMat(A,p)` réalise-t-il d'appels à `prodMat` en fonction de p ?

- XII.** L'algorithme d'exponentiation rapide repose sur la remarque que

$$a^{2p+1} = (a^p)^2 a = b \times b \times a,$$

où $b = a^p$, quand $a^{2p} = b \times b$. Selon la parité de p , il faut donc 1 ou 2 multiplications de plus pour calculer a^{2p+1} ou a^{2p} connaissant a^p . Ainsi, pour calculer a^{122} , par exemple :

$$\begin{aligned} a^{122} &= (a^{61})^2 \\ &= (a.(a^{30})^2)^2 \\ &= (a.((a^{15})^2)^2)^2 \\ &= (a.((a.(a^7)^2)^2)^2)^2 \\ &= (a.((a.(a.(a^3)^2)^2)^2)^2)^2 \\ &= (a.((a.(a.(a.a^2)^2)^2)^2)^2)^2. \end{aligned}$$

Ainsi, on se contente de 10 multiplications.

1. On propose la fonction suivante qui implémente l'exponentiation rapide pour les matrices 2×2 .

```

0 def puissanceMatRapide(M,n):
1     R = [[1,0], [0,1]]
2     P = M
3     while n > 0:
4         if n%2 == 1:
5             R = prodMat(R,P)
6             P = prodMat(P,P)
7             n = n // 2
8     return R

```

Montrer que par cette méthode, le nombre d'appel à la fonction `prodMat` est majoré par $2 + 2\lfloor \log_2 p \rfloor$.

2. Exprimer en fonction de M et de i la valeur de la matrice P , après la i ème itération de la boucle `while`.
 3. Soit $n = \overline{c_p \dots c_0}$ l'écriture de n en base 2. Montrer que, pour tout entier i compris entre 1 et p , la valeur de l'entier n après la i ème itération de la boucle `while` est donnée par

$$n = \sum_{j=i}^p c_j 2^{j-i}.$$

4. Montrer que pour tout entier i compris entre 1 et p , la valeur de la matrice R après la i ème itération de la boucle `while` est donnée par $R = M^k$, avec

$$k = \sum_{j=0}^{i-1} c_j 2^j.$$

5. Exprimer le nombre d'exécutions de la boucle `while` en fonction de n puis démontrer la correction de l'algorithme proposé, c'est-à-dire que `puissanceMatRapide(M, n)` calcule effectivement la puissance désirée.

- XIII. Proposer le code d'une fonction `lucas5(n)` qui retourne la valeur du nombre de Lucas L_n en utilisant la fonction `puissanceMatRapide`.

Problème n° 2 : emplois du temps et graphes d'intervalles

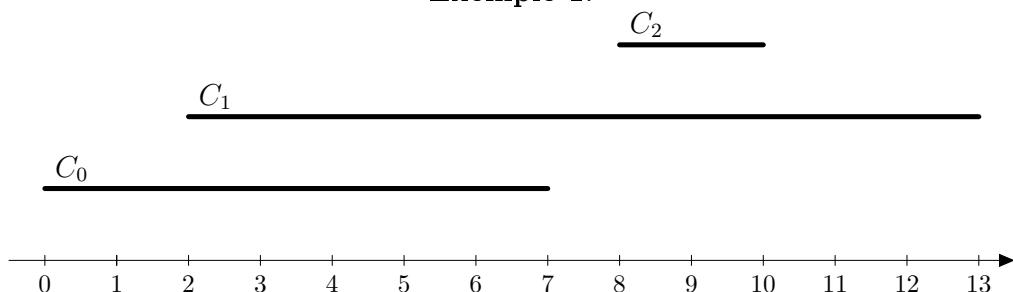
Dans ce problème, on s'intéresse à l'allocation des salles d'un lycée à partir des horaires des cours.

La donnée du problème est une liste de cours. Un cours est simplement représenté par un intervalle $[deb, fin]$, où deb est l'instant de début du cours et fin est l'instant de fin du cours. Les instants peuvent être décomptés en heures ou en minutes au fil de la journée selon les besoins ; dans ce sujet, les instants sont des nombres entiers et l'unité de temps n'est pas précisée.

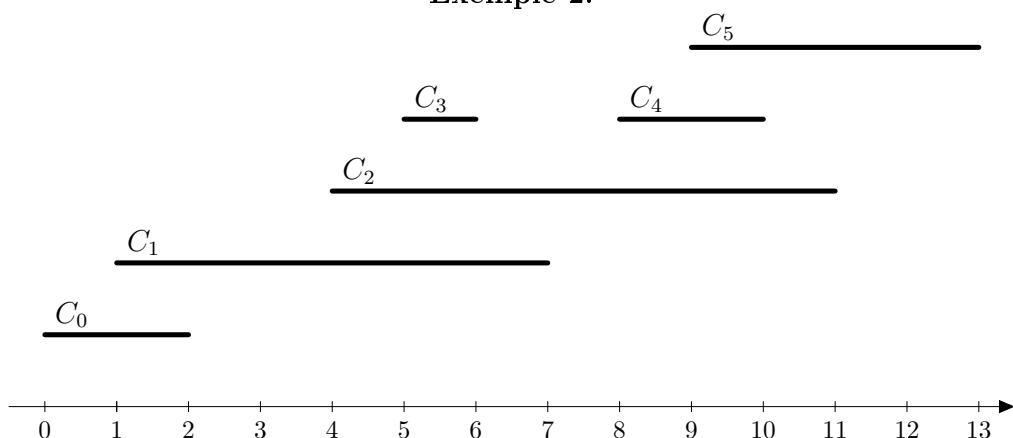
À chaque cours, on doit allouer une salle. Deux cours peuvent se voir allouer la même salle, uniquement si leurs intervalles sont disjoints. En particuliers, deux cours représentés par les intervalles $[4, 6[$ et $[6, 8[$ peuvent se voir allouer la même salle. L'objectif est d'utiliser un minimum de salles.

On présente ci-dessous, deux instances de ce problème.

Exemple 1.



Exemple 2.



Dans l'exemple 1, le cours C_2 est représenté par l'intervalle $[8, 10[$.

- I.** Pour l'allocation des salles, on parcourt simplement les cours par instants de début croissants et on alloue systématiquement la salle disponible avec le numéro le plus petit.
1. Décrire, sans justification, les allocations ainsi obtenues pour chacun des exemples précédents.
 2. Déterminer le nombre minimal de salles nécessaires pour l'allocation dans les deux exemples précédents.

On admet pour la suite que le nombre optimal de salles nécessaire pour une liste de cours est le nombre maximal d'intervalles s'intersectant mutuellement en un instant donné.

- II.** On s'intéresse dans cette question à une première modélisation du problème. Le cours représenté par la liste Python `[d, f]` se déroule sur l'intervalle mathématique $[d, f]$.
1. Déterminer les listes d'intervalles représentant les exemples 1 et 2.
 2. Compléter la définition suivante :

```
def insere(l, elt):
```

où `l` est une liste d'entiers triée dans l'ordre croissant et `elt` est un entier, et qui renvoie une liste triée obtenue par insertion à sa place de `elt` dans `l`. On notera que la liste `l` peut être vide.

On suppose pour la suite que l'on dispose d'une fonction similaire `insereBis` qui opère sur des listes de listes plutôt que sur des listes d'entiers, le critère de tri étant l'ordre des premiers éléments de chaque sous-liste. Plus précisément, la fonction `insereBis` a pour en-tête `def insereBis(LL, li)` : et a pour effet d'insérer une liste `li` à la place adéquate dans la liste de listes `LL`.

- III.** Pour automatiser l'allocation des salles, on va utiliser une autre modélisation. L'intervalle `[deb, fin[` représentant le cours numéro i va être représenté par deux événements, modélisés par des listes de longueur 3 : `[deb, i, 0]` et `[fin, i, 1]`. Une liste d'intervalles pourra ainsi être représentée par une liste d'événements, c'est-à-dire une liste de listes de longueur 3 de la forme `[instant, num, 0 ou 1]`.

1. Compléter la définition

```
def traduit(liste_intervalles):
```

qui prend en argument une liste d'intervalles représentés par des listes de longueur 2 et qui renvoie une liste d'événements (listes de longueur 3) correspondante. Notons que pour n intervalles, on obtient $2n$ événements.

2. Pour l'efficacité des algorithmes de résolution, on va travailler sur une liste d'événements triée par instants croissants. On appellera *agenda* toute liste d'événements triés par instants croissants. Quels sont les agendas correspondant aux exemples 1 et 2 ?
3. Compléter la définition

```
def agenda(liste_evt):
```

qui prend en argument une liste d'événements (listes de longueur 3) obtenue par un appel à la fonction `traduit` et qui renvoie l'agenda correspondant. On pourra utiliser la fonction `insereBis`.

IV. On a demandé à des élèves d'écrire une fonction qui vérifie qu'une liste d'événements donnée contient bien autant de fin que de début, et dans le bon ordre, mais sans tester l'appariement cours par cours, c'est-à -dire sans considérer les numéros d'intervalles.

- Parmi les quatres solutions proposées, déterminer (sans justifier) la ou les réponses correctes.

```

0 def valideA(agenda):
1     c = 0
2     for e in agenda:
3         if e[2] == 0: c += 1
4         else: c -= 1
5         if c < 0: return False
6         else : return True
7
8 def valideB(agenda):
9     n,c,i,b = len(agenda),0,0,True
10    while b and (i < n):
11        if agenda[i][2] == 0: c += 1
12        else: c -= 1
13        i += 1
14        b = (c >= 0)
15    return c == 0
16
17 def valideC(agenda):
18     for i in range(len(agenda)):
19         if agenda[i][2] == 0:
20             b = False
21             for j in range(i+1,len(agenda)):
22                 if agenda[j][2] == 1: b = True
23                 if not b : return(b)
24     return(True)
25
26 def valideD(agenda):
27     c = 0
28     for e in agenda:
29         c += 1 - 2*e[2]
30         if c < 0: return False
31     return c == 0

```

- Adapter une des fonctions précédentes pour écrire une fonction `intersection_max` qui à partir d'un agenda calcule le nombre maximal d'intervalles qui s'intersectent mutuellement. Justifier la correction de l'approche.
- En utilisant les fonctions précédentes, écrire une fonction `nbr_optimal` qui à partir d'une liste d'intervalle (modélisation initiale), calcule le nombre de salles nécessaire.

- V. 1. On a demandé à un élève d'écrire une fonction qui, étant donné une liste de booléens, calcule le plus petit entier i tel que la case d'indice i vaille True. La fonction renverra -1 si un tel indice n'existe pas. Corriger sa fonction.

```

0 def plus_petit_vrai(liste):
1     n = len(liste)
2     while liste[i] and (i < n) : i+= 1
3     if i == n: return -1
4     else: return i

```

2. On utilise à chaque instant une liste de booléens pour indiquer si une salle est disponible ou non. En utilisant les fonctions précédentes, compléter la fonction suivante qui étant donnée une liste d'intervalles (listes de longueur 2), calcule une liste d'allocations des salles, toujours en allouant la salle disponible avec le plus petit numéro. Dans cette liste, la case d'indice le numéro du cours contient le numéro de la salle allouée.

```

0 def allocation(liste_intervalles):
1     nb_cours = ...
2     liste = ...
3     nb_salles = ...
4     salles_dispos = [True]*nb_salles
5     alloc = [-1]*nb_cours
6     for l in liste:
7         if l[2] == 0 :
8             alloc[l[1]] = ...
9             salles_dispos[...] = False
10        else :
11            salles_dispos[...] = ...
12
return(alloc)

```

Sélection internationale
École Normale Supérieure
À Épreuve de culture scientifique - Informatique

Session 2015
 Paris
 Durée : 3 heures

Pour les candidats ayant choisi **l'informatique** comme **spécialité principale**

Si vous ne parvenez pas à répondre à une question, vous pouvez cependant l'utiliser comme hypothèse pour les questions suivantes.

Calculatrices interdites.

Exercice 1. Supposons que nous avons en notre possession n pièces de monnaie dont certaines sont contrefaites. Les pièces réelles ont toutes le même poids w_0 et les pièces contrefaites ont toutes le même poids $w_1 < w_0$. Nous disposons d'une balance de type Roberval très précise qui étant donnés deux ensembles arbitraires de pièces (parmi ces n pièces) nous permet de décider si les ensembles sont de même poids ou quel ensemble est le plus léger. Sans l'aide de cette balance, les pièces sont cependant indistinguables. Nous supposons qu'il y a au moins une pièce contrefaite dans l'ensemble des n pièces.

1. Proposer un algorithme qui permet de trouver (au moins) une pièce contrefaite lorsqu'il y a $n = 8$ pièces et effectue le nombre minimum de pesées (on ne vous demande pas de demander que l'algorithme minimise le nombre de pesées). Combien votre algorithme effectue-t-il de pesées au pire ?
2. Proposer un algorithme qui permet de trouver (au moins) une pièce contrefaite en $\log_2(n) + O(1)$ pesées.
3. Proposer un algorithme en $\log_2(n) + O(1)$ pesées. qui permet de diviser les pièces en trois ensembles S_1 , S_2 et S_3 tels que
 - S_1 et S_2 contiennent autant de pièces réelles ;
 - S_1 et S_2 contiennent autant de pièces contrefaites ;
 - S_3 contient au plus deux pièces.
4. En déduire un algorithme en $\log_2(n) + O(1)$ pesées qui retourne la parité du nombre de pièces contrefaites parmi les n pièces.
5. Proposer un algorithme en $\log_2^2(n) + O(\log_2(n))$ pesées qui retourne le nombre de pièces contrefaites parmi les n pièces.
6. Montrer (par un argument de combinatoire par exemple) qu'un algorithme qui retourne le nombre de pièces contrefaites parmi les n pièces doit effectuer au moins $\log_3(n)$ pesées.

Exercice 2. Supposons que nous avons en notre possession n écrous et n vis de tailles différentes. Chaque vis correspond à un écrou et un seul (et réciproquement). Les écrous et les vis sont presque de la même taille et il est impossible de dire si un écrou est plus grand qu'un autre ou si une vis est plus grande qu'une autre. Cependant, si on essaie de faire correspondre un écrou à une vis, la vis sera soit trop grande, soit trop petite, soit de la bonne taille exactement. On peut donc faire des comparaisons écrou-vis, et chaque comparaison donne un résultat parmi les trois résultats possibles.

1. Proposer un algorithme qui, étant donné un écrou, retrouve la vis correspondante. Donner sa complexité en nombre de comparaisons écrou-vis (en fonction de n).
2. Proposer un algorithme probabiliste inspiré du tri rapide pour associer correctement chaque écrou à sa vis.
3. Soit $X_{i,j}$ la variable aléatoire qui vaut 1 si à un moment lors de l'exécution le écrou i est comparé à la vis j , et $X_{i,j} = 0$ sinon. Exprimer en fonction des $X_{i,j}$ le nombre $T(n)$ de comparaisons écrou-vis effectuées par l'algorithme.
4. Donner une formule pour $\mathbb{E}(X_{i,j})$ l'espérance de $X_{i,j}$.
5. Montrer que $\mathbb{E}(T(n)) = O(n \log n)$ (où $\mathbb{E}(T(n))$ est l'espérance de $T(n)$).

Exercice 3. Vous avez n devoirs à faire mais avez pris du retard. Le i -ième devoir vous réclamera h_i heures de travail et doit être terminé avant le temps t_i , sinon vous sera pénalisé : si vous le terminez au temps t'_i , votre pénalité sera 0 pour $t'_i \leq t_i$ et $t'_i - t_i$ pour $t'_i > t_i$. Le but est de déterminer pour chaque i l'heure s_i de démarrage du devoir i , de façon à minimiser la pénalité totale que vous devrez payer. Bien évidemment, vous ne pouvez travailler que sur un devoir à la fois.

1. Donner un algorithme polynomial qui, étant donné h_1, h_2, \dots, h_n , calcule s_1, s_2, \dots, s_n .
2. Démontrer que votre algorithme est correct, c'est-à-dire qu'il minimise la pénalité que vous devrez payer.
3. Quel est la complexité de l'algorithme en temps ?
4. Finalement, vous n'allez pas y arriver et devez laisser tomber certains des devoirs. Choisir de ne pas rendre le devoir i vous fait encourrir une pénalité de p_i . Donner un algorithme efficace qui, étant donné $(h_1, p_1), (h_2, p_2), \dots, (h_n, p_n)$, calcule $(b_1, s_1), (b_2, s_2), \dots, (b_n, s_n)$, où b_i est une variable booléenne, vraie si vous terminez le devoir i et faux sinon, et s_i n'est définie que si b_i est vrai et est dans ce cas égal à l'heure de démarrage de la tâche i .

Exercice 4. Le but de l'exercice est de concevoir une structure de données pour stocker des mesures de températures. Une mesure est de la forme $(t; d)$ où t , un nombre réel qui peut être négatif, est une température en Celsius, et où d , un entier naturel, est le nombre de jours écoulés entre le 31 décembre 1999 et le jour de la mesure (d est toujours positif). Il peut y avoir plusieurs mesures un même jour (qui peuvent mesurer la même température, ou non - S est donc un multi-ensemble puisqu'un même (t, d) peut être présent plusieurs fois), et il peut également n'y en avoir aucune.

La structure de données pour S doit pouvoir gérer les opérations suivantes :

- insérer($t; d$) qui ajoute à S la mesure $(t; d)$; et
- moyenne($d_1; d_2$) où $d_1 \leq d_2$ sont deux dates. Cette requête donne en résultat la moyenne uniforme des températures de tous les (t, d) de S tels que $d_1 \leq d \leq d_2$, c'est-à-dire la somme de ces températures divisée par le nombre de mesures.

1. Montrer brièvement comment, avec une structure de données naïve, on peut obtenir une complexité de $O(1)$ pour chaque insertion et $O(n)$ pour chaque moyenne, où n est la taille de S .
2. Montrer comment, avec une autre méthode, on peut obtenir une complexité de $O(D)$ pour chaque insertion et $O(1)$ pour chaque requête de moyenne, où D est le nombre total de jours écoulés entre le 31 décembre 1999 et aujourd’hui.
3. Proposer une structure de données avec laquelle chaque insertion a complexité $O(\log n)$ et chaque requête de moyenne a complexité $O(\log n)$.

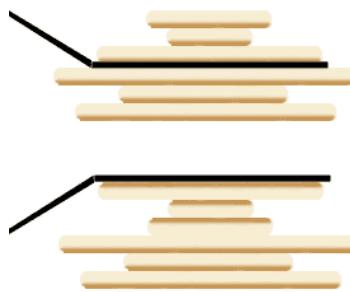
**Sélection internationale
École Normale Supérieure
Épreuve de culture scientifique - Informatique**

Session 2017
Paris
Duration : 2 hours

For candidates who chose **computer science as secondary specialisation**

If you cannot answer a question, you may use it as hypothesis to later questions.
Calculators are not allowed.

Exercice 1. Supposons que nous avons une pile de n crêpes de tailles différentes. Nous voulons trier les crêpes de sorte que chaque crêpe est toujours au dessus d'une crêpe de diamètre supérieur. La seule opération autorisé pour le tri est un retournement qui consiste à insérer une spatule sous les k crêpes du dessus (for k compris entre 1 et n) et à les retourner toutes.



1. Donner une borne inférieure sur le nombre de retournements nécessaires pour un trier un ensemble arbitraire de n crêpes.
2. Décrire un algorithme efficace pour trier un ensemble arbitraire de n crêpes. Combien de retournements effectue exactement votre algorithme dans le pire des cas ?
3. Supposons maintenant que une face de chaque crêpe est brûlée. Combien de retournements doit-on effectuer pour trier les crêpes de sorte que les faces brûlées de toute la crêpe soient en dessous ?

Exercice 2.

1. Supposons que nous avons deux tableaux triés $A[1 \dots m]$ et $B[1 \dots n]$ et un entier k . Décrire un algorithme pour trouver le k -ième plus petit élément de l'union de A et de B en temps $O(\log(m + n))$.

Par exemple, sur l'entrée

$$A[1 \dots 8] = [0, 1, 6, 9, 12, 13, 18, 20], B[1 \dots 5] = [2, 5, 8, 17, 19] \text{ et } k = 6$$

votre algorithme doit retourner 8. Vous pouvez supposer que les tableaux ne contiennent pas de doublon.

2. Supposons que nous avons deux tableaux triés $A[1 \dots n]$ et $B[1 \dots n]$. Donner un algorithme en temps $O(\log^2 n)$ pour trouver le n -ième plus petit des $2n$ éléments.
3. Supposons que nous avons k tableaux triés, chacun de n éléments et que nous voulons les combiner en un seul tableau trié de kn éléments.

- (a) Considérons la stratégie suivante : en utilisant la procédure de fusion (du tri fusion), fusionner les deux premiers tableaux, puis le résultat avec le troisième, puis le résultats avec le quatrième, et ainsi de suite. Quelle est la complexité en temps de cet algorithme en fonction de k et de n ?
- (b) Donner une solution plus efficace à ce problème, en utilisant une stratégie diviser-pour-régner. Quelle est la complexité en temps de cet algorithme en fonction de k et de n ?
4. Soit $M[1 \dots n][1 \dots n]$ une matrice $n \times n$ où chaque ligne et chaque colonne est triée. Une telle matrice est dite *totalement mononote* et nous supposons que tous ses éléments sont distincts.
- (a) Décrire et analyser un algorithme pour résoudre le problème suivant en temps $O(n)$: étant donnés des indices i, j, i', j' , calculer le nombre d'entiers de M plus petit que $M[i][j]$ et plus grand que $M[i'][j']$.
- (b) Décrire et analyser un algorithme pour résoudre le problème suivant en temps $O(n)$: étant donnés des indices i, j, i', j' , retourner un élément de M tiré uniformément aléatoirement parmi les éléments plus petits que $M[i][j]$ et plus grands que $M[i'][j']$. On supposera que l'ensemble demandé est toujours non vide.
- (c) Décrire et analyser un algorithme probabiliste pour calculer l'élément médian de M en temps espéré $O(n \log n)$.
5. Soit $A[1 \dots n]$ un tableau dont les $n\sqrt{n}$ premiers éléments sont triés (mais sur lequel on ne sait rien pour les éléments restants). Écrire un algorithme qui trie A en un temps substantiellement meilleure que $O(n \log n)$ opérations.

Exercice 3. Soit $k, n \in \mathbb{N}$. Le problème (k, n) -œuf est le suivant : considérons un immeuble de n étages, nous savons qu'il existe un étage f tel que si un œuf est lâché du f -ième étage, il se casse alors que si il est jeté du $(f - 1)$ -ième étage, il ne se casse pas (il est possible que $f = 1$ et dans ce cas, l'œuf se casse toujours ou que $f = n + 1$ et dans ce cas, l'œuf ne se casse jamais).

Si les œufs se cassent lorsqu'ils sont lâchés d'un certain étage, alors ils cassent aussi si ils sont lâchés d'un étage supérieur. Si les œufs ne cassent pas lorsqu'ils sont lâchés d'un certain étage, alors ils ne se cassent pas non plus si ils sont lâchés d'un étage inférieur.

Le but de l'exercice est, étant donnés k œufs, de trouver f . La seule opération autorisée est de lancer un œuf à partir d'un étage et de voir ce qu'il se passe. Si un œuf est cassé, il ne peut pas être réutilisé. Le but est de lâcher des œufs le moins de fois possible. Soit $E(k, n)$ le nombre minimal de lâcher à effectuer pour toujours déterminer f .

1. Montrer que $E(1, n) = n$.
2. Montrer que $E(k, n) = \Theta(n^{1/k})$
3. Trouver une relation de récurrence pour $E(k, n)$. Écrire un programme dynamique pour trouver $E(k, n)$. Quelle est sa complexité en temps ?
4. Écrire un programme dynamique qui peut être utilisé pour trouver la stratégie optimale pour trouver l'étage f . Quelle est sa complexité en temps ?

ÉCOLE NORMALE SUPÉRIEURE DE LYON
Concours d'admission session 2017
Filière universitaire : Second concours
COMPOSITION D'INFORMATIQUE

Durée : 3 heures

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

★ ★ *

1 Préliminaires

On s'intéresse à la représentation d'ensembles disjoints, c'est-à-dire de collections $\mathcal{S} = \{S_1, \dots, S_k\}$, où S_1, \dots, S_k sont des ensembles disjoints finis. Ce sujet comporte trois parties qui ne sont pas indépendantes. Il est toutefois possible d'admettre le résultat de toute question pour répondre aux questions suivantes.

L'énoncé de ce sujet est écrit en utilisant PYTHON comme langage de référence. Cependant, lorsque du code est demandé, ce code peut être écrit en PYTHON, en pseudo-code ou dans un langage au choix du candidat, en utilisant les structures de contrôle habituelles. Toutes les réponses devront être justifiées.

Étant données deux fonctions $f, g : \mathbb{N}^k \rightarrow \mathbb{N}$, nous disons que f est en $O(g(n_1, \dots, n_k))$ lorsqu'il existe des constantes $M, N_0, \dots, N_k \in \mathbb{N}$ telles que $f(n_1, \dots, n_k) \leq M \cdot g(n_1, \dots, n_k)$ pour tous n_1, \dots, n_k avec $n_i \geq N_i$ pour tout $i = 1, \dots, k$.

La complexité, ou le temps d'exécution, d'un programme P ou d'une séquence d'instructions o_1, \dots, o_m , est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de P (resp. à l'exécution de la séquence d'instructions o_1, \dots, o_m). Cette complexité peut dépendre de paramètres n_1, \dots, n_k et donc être vue comme une fonction $f : \mathbb{N}^k \rightarrow \mathbb{N}$. Dans ce cas nous dirons que P (resp. o_1, \dots, o_m) a une complexité en $O(g(n_1, \dots, n_k))$ lorsque f est en $O(g(n_1, \dots, n_k))$.

2 Ensembles disjoints par forêts

Un **noeud** est une liste PYTHON x à au moins deux éléments. L'élément $x[0]$ contient la valeur du noeud, et l'élément $x[1]$ désigne la liste représentant le parent du noeud x . On dit que x est une **racine** si $x[1]$ désigne x . La fonction suivante renvoie **True** si le noeud x est une racine et **False** sinon.

```
def est_racine(x) : return (x[1] == x)
```

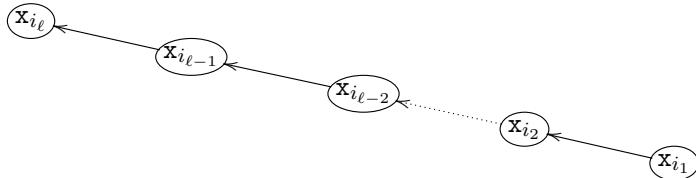
D'autre part, la fonction suivante crée et renvoie un nouveau noeud racine.

```
def cree_racine () :
    x = [None, None]
    x[1] = x
    return x
```

Considérons n noeuds $\mathbf{x}_1, \dots, \mathbf{x}_n$ tels que chaque \mathbf{x}_i a pour parent un \mathbf{x}_j , c'est-à-dire tels que pour tout $1 \leq i \leq n$, il existe un $1 \leq j \leq n$ avec $\mathbf{x}_i[1] = \mathbf{x}_j$. On dit qu'un noeud \mathbf{x}_j est un **ascendant** d'un noeud \mathbf{x}_i (et reciprocement que \mathbf{x}_i est un **descendant** d'un noeud \mathbf{x}_j), notation $\mathbf{x}_i \preceq \mathbf{x}_j$, si on peut atteindre \mathbf{x}_j à partir de \mathbf{x}_i en suivant les parents, c'est-à-dire s'il existe une suite de noeuds $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}$ telle que

$$\mathbf{x}_i = \mathbf{x}_{i_1} \quad \text{et} \quad \mathbf{x}_{i_1}[1] = \mathbf{x}_{i_2} \quad \text{et} \quad \dots \quad \text{et} \quad \mathbf{x}_{i_{\ell-2}}[1] = \mathbf{x}_{i_{\ell-1}} \quad \text{et} \quad \mathbf{x}_{i_{\ell-1}}[1] = \mathbf{x}_{i_\ell} = \mathbf{x}_j \quad (1)$$

ce que l'on représente de la manière suivante :



Remarquons tout d'abord que l'ensemble des descendants d'un noeud donné est linéairement ordonné par \preceq .

Question 2.1 Justifier brièvement l'assertion suivante :

- Si $\mathbf{x}_i \preceq \mathbf{x}_j$ et $\mathbf{x}_i \preceq \mathbf{x}_\ell$, alors soit $\mathbf{x}_j \preceq \mathbf{x}_\ell$, soit $\mathbf{x}_\ell \preceq \mathbf{x}_j$.

On suppose maintenant que la relation \preceq est **acyclique**, au sens où $\mathbf{x}_i = \mathbf{x}_j$ dès lors que $\mathbf{x}_i \preceq \mathbf{x}_j \preceq \mathbf{x}_i$. Dans ce cas, la relation \preceq définit une **forêt**, c'est-à-dire un ensemble d'arbres.

Question 2.2 Justifier brièvement l'assertion suivante :

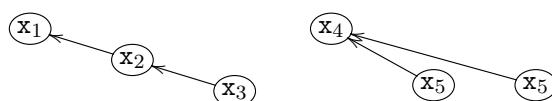
- En supposant que \preceq est acyclique, tout noeud a un descendant qui est une racine.

Un **arbre** de \preceq est par définition l'ensemble des descendants d'un noeud racine. Ainsi, chaque noeud \mathbf{x}_i appartient à un unique arbre de \preceq , et les noeuds $\mathbf{x}_1, \dots, \mathbf{x}_n$ décrivent une collection d'arbres disjoints $\mathcal{S} = \{S_1, \dots, S_k\}$, c'est-à-dire une collection d'ensembles disjoints.

Exemple. Considérons des noeuds $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_6$ avec

$$\mathbf{x}_1[1] = \mathbf{x}_1, \quad \mathbf{x}_2[1] = \mathbf{x}_1, \quad \mathbf{x}_3[1] = \mathbf{x}_2 \quad \text{et} \quad \mathbf{x}_4[1] = \mathbf{x}_5[1] = \mathbf{x}_6[1] = \mathbf{x}_4$$

Ces noeuds représentent la forêt



(dont les racines sont \mathbf{x}_1 et \mathbf{x}_4) et la collection $\{\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}, \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}\}$.

Recherche de représentant. Supposons donnés n noeuds $\mathbf{x}_1, \dots, \mathbf{x}_n$, engendrant une relation \preceq acyclique, et décrivant la collection d'ensembles disjoints $\mathcal{S} = \{S_1, \dots, S_k\}$. En d'autres termes, \mathcal{S} est la collection des arbres de \preceq . Par définition, le **représentant** de l'ensemble S_i est le noeud racine de l'arbre correspondant. Le **représentant** d'un noeud \mathbf{x} est le représentant de l'unique S_i auquel \mathbf{x} appartient.

Question 2.3 Écrire une fonction `repr` qui prend en argument un noeud et qui renvoie son représentant. Donner une borne de complexité de cette fonction en fonction de n .

Il est très simple de savoir si deux noeuds quelconques appartiennent au même ensemble.

Question 2.4 Écrire une fonction qui prend arguments deux noeuds, et qui renvoie `True` si ces deux noeuds appartiennent au même ensemble et `False` sinon.

Union d'ensembles. Nous allons maintenant implémenter une fonction `union` qui permet de réaliser l'union de deux ensembles $S, T \in \mathcal{S}$. Cette opération va en fait être implémentée directement au niveau des noeuds et de leurs représentants. La fonction `union` prend en arguments deux noeuds x et y dont les représentants sont supposés distincts. Si $x \in S$ et $y \in T$, alors l'exécution de `union(x, y)` modifie les représentants respectifs px et py de x et y de telle sorte que px devienne le parent de py , ou bien que py devienne le parent de px . Dans le contexte de ce sujet, il est pratique d'implémenter la fonction `union` en utilisant une fonction auxillière `relie`, et d'écrire `union` de la manière suivante :

```
def union(x,y) : relie(repr(x), repr(y))
```

Question 2.5 Écrire une fonction `relie`, qui prend en arguments deux noeuds x et y et qui modifie le parent de y pour qu'il devienne x . La complexité de cette fonction doit être en $O(1)$.

Coût d'une séquence d'appels à `cree_racine`, `union` et `repr`. Considérons une séquence de m instructions o_1, \dots, o_m , dans laquelle chaque o_i consiste en l'appel à une des fonctions `cree_racine`, `repr` ou `union`. On suppose que cette séquence débute par $n \leq m$ appels à `cree_racine`, de sorte à créer n noeuds racines x_1, \dots, x_n , et que les instructions o_{n+1}, \dots, o_m consistent uniquement en des appels à `repr` ou `union` sur des noeuds parmi x_1, \dots, x_n . On suppose en outre que tous les appels à `union` sont faits sur des noeuds de représentants distincts.

Question 2.6 Montrer que dans une séquence d'instructions o_1, \dots, o_m comme ci-dessus, il y a au plus $n - 1$ appels à `union`.

Question 2.7 Montrer que la complexité d'une séquence d'instructions o_1, \dots, o_m comme ci-dessus est en $O(m \cdot n)$.

L'objet de la question suivante est de montrer que la borne asymptotique de la Question 2.7 peut être atteinte.

Question 2.8 Donner une suite de séquences d'instructions $(o_{k,1}, \dots, o_{k,m_k})_{k \geq 0}$ comme ci-dessus, et qui satisfait les deux propriétés suivantes :

- La suite $(m_k)_{k \geq 0}$ est croissante.
- Soit n_k le nombre d'appels à la fonction `cree_racine` dans la séquence d'instructions $o_{k,1}, \dots, o_{k,m_k}$. Alors il existe des constantes M, K_0 avec $K_0 \in \mathbb{N}$ et $M \in \mathbb{Q}$, $M > 0$, et telles que la complexité de $o_{k,1}, \dots, o_{k,m_k}$ est supérieure ou égale à $M \cdot n_k \cdot m_k$ pour tout $k \geq K_0$.

Compression de chemins. Nous nous intéressons maintenant à une heuristique pour la recherche de représentant, appellée **compression de chemins**, et qui permet, comme nous allons le voir par la suite, d'améliorer la complexité « amortie » de la recherche de représentant dans une suite de m opérations o_1, \dots, o_m comme ci-dessus.

La compression de chemin est une opération réalisée par une fonction de recherche de représentant, et qui permet, pour deux appels successifs à cette fonction sur le même noeud, d'effectuer le second appel en temps constant (par rapport à m). Le principe est qu'un appel à `repr` sur un noeud x de représentant y modifie les parents de tous les descendants de x (y compris le parent de x lui-même) de telle sorte que chacun de ces parents soit y .

Question 2.9 Donner une nouvelle implémentation de la fonction `repr`, qui prend en argument un noeud, qui renvoie le représentant de ce noeud, et qui implémente l'heuristique de compression de chemins. La complexité de `repr(x)` doit être en $O(s)$ où s est tel que si $x_{i_1}, \dots, x_{i_\ell}$ satisfont (1), sont deux à deux distincts, et sont tels que $x = x_{i_1}$ et x_{i_ℓ} est le représentant de x , alors $s = \ell$.

3 Arbres avec noeuds classés

Nous avons au §2 proposé une structure de données pour représenter des ensembles disjoints sous forme d'arbres. Cette structure de donnée était en particulier équipée d'une fonction `union` réalisant l'union de deux ensembles, implémentée à l'aide de la fonction auxillière `relie` de la Question 2.5. Un appel à `relie(x, y)` modifie `y` de sorte que `x` devienne le représentant de `y`. Nous allons maintenant voir une heuristique qui permet d'améliorer la compléxité d'une séquence d'instructions o_1, \dots, o_m , et selon laquelle `relie(x, y)` choisi pour représentant parmi `x` et `y` celui qui est la racine de l'arbre le plus haut.

Noeuds classés. Un **noeud classé** est une liste PYTHON `x` avec au moins **trois** éléments. L'élément `x[0]` contient la valeur du noeud, l'élément `x[1]` désigne le parent du noeud `x`, et l'élément `x[2]` est entier positif ou nul, le **rang** de `x`. Les noeuds classés sont donc en particulier des noeuds au sens du §2. De ce fait les définitions de noeud racine et de \preceq ne sont pas modifiées. De plus `x[1][1]` désigne le parent du parent de `x` et `x[1][2]` désigne le rang du parent de `x`. La fonction `cree_racine` peut être adaptée aux noeuds classés de manière à créer un noeud de rang nul :

```
def cree_racine () :
    x = [None, None, 0]
    x[1] = x
    return x
```

Question 3.1 Réécrire la fonction `relie` de la manière suivante. Si `x` et `y` sont des racines de rangs strictement supérieurs aux rangs de leurs descendants, alors `relie(x, y)` choisi pour racine parmi `x` et `y` celui de plus grand rang, et met à jour le rang du noeud racine de sorte qu'il soit le plus petit entier strictement supérieur aux rangs de ses descendants dans le nouvel arbre, et supérieur ou égal aux rangs de `x` et `y` avant l'appel à `relie(x, y)`. La complexité de cette fonction doit être en $O(1)$.

Coût d'une suite d'opérations. On suppose à partir de maintenant et pour toute la suite de cette partie que la fonction `cree_racine` est celle définie juste avant la Question 3.1, que la fonction `union` utilise la fonction `relie` de la Question 3.1 et que la fonction `repr` est celle de la Question 2.9.

Considérons une séquence d'instructions o_1, \dots, o_m , dans laquelle chaque o_i consiste en l'appel à une des fonctions `cree_racine`, `repr` ou `union` sur des noeuds classés. On suppose en outre que la séquence o_1, \dots, o_n consiste en n appels à `cree_racine` et que tous les appels à `union` sont faits sur des noeuds de représentants distincts. L'objet de la suite de cette partie est de montrer que la complexité de o_1, \dots, o_m est presque linéaire en n et m . Nous allons voir qu'elle est en $O(m \cdot \alpha(n))$, où α est une fonction à croissance très lente (avec $\alpha(n) \leq 4$ pour toute application pratiquement concevable).

La fonction α . Les fonctions $(A_k)_{k \in \mathbb{N}}$, avec $A_k : \mathbb{N} \rightarrow \mathbb{N}$ sont définies de la manière suivante :

$$\begin{aligned} A_0(j) &:= j + 1 \\ A_{k+1}(j) &:= A_k^{(j+1)}(j) \end{aligned}$$

où, pour $f : \mathbb{N} \rightarrow \mathbb{N}$, on définit $f^{(i)} : \mathbb{N} \rightarrow \mathbb{N}$ par $f^{(0)}(j) := j$ et $f^{(i+1)}(j) := f(f^{(i)}(j))$.

Question 3.2

- (i) Montrer que pour tout $k \in \mathbb{N}$, la fonction A_k est strictement croissante, et que pour tout $j \in \mathbb{N}$, on a $j < A_k(j)$.
- (ii) Montrer que si $k < \ell$, alors pour tout $j > 0$ on a $A_k(j) < A_\ell(j)$.
- (iii) Montrer que pour tout $j \in \mathbb{N}$, il existe $k \geq 0$ tel que $j \leq A_k(1)$.

La fonction $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ est définie par

$$\alpha(n) := \min\{k \mid A_k(1) \geq n\}$$

On a donc $n \leq A_{\alpha(n)}(1)$.

Une borne temporelle amortie. Fixons maintenant et pour la suite de cette partie une séquence de m' instructions $o'_1, \dots, o'_{m'}$, dans laquelle chaque o'_i consiste en l'appel à une des fonctions `cree_racine`, `repr` ou `union`. On suppose que cette séquence débute par $n \leq m$ appels à `cree_racine`, de sorte à créer n noeuds classés racines x_1, \dots, x_n , que les instructions $o'_{n+1}, \dots, o'_{m'}$ consistent uniquement en des appels à `repr` ou `union` sur des noeuds parmi x_1, \dots, x_n , et que tous les appels à `union` sont faits sur des noeuds de représentants distincts. Nous allons voir que la complexité de $o'_{n+1}, \dots, o'_{m'}$ est en $O(m' \cdot \alpha(n))$.

Afin de montrer ce résultat, nous décomposons chaque appel à `union` en deux appels à `repr` suivis d'un appel à `relie`. Ceci génère une séquence de m instructions o_1, \dots, o_m , avec $m \leq 3m'$. Notons que les n -premières instructions o_1, \dots, o_n sont des appels à `cree_racine`, créant les noeuds racines x_1, \dots, x_n , et que les instructions o_{n+1}, \dots, o_m consistent uniquement en des appels à `repr` et `relie`. De plus, si la complexité de o_1, \dots, o_m est en $O(m \cdot \alpha(n))$, alors la complexité de $o'_1, \dots, o'_{m'}$ est en $O(m' \cdot \alpha(n))$.

On va maintenant s'intéresser uniquement à la séquence o_{n+1}, \dots, o_m . Notons qu'aucun noeud n'est créé (ni détruit) dans cette séquence. De plus, avant l'instruction o_{n+1} le rang de chaque noeud est nul.

Question 3.3 Soit $k \geq n$ et considérons le noeud x_i (avec $1 \leq i \leq n$) après l'instruction o_k .

- (i) Montrer que $x_i[2] \leq x_i[1][2]$, et que $x_i[2] = x_i[1][2]$ si et seulement si x_i est une racine.
- (ii) Soit r' le rang de x_i après l'instruction o_{k+1} . Montrer que $r' \geq x_i[2]$, et que $r' = x_i[2]$ si x_i n'est pas une racine.
- (iii) Montrer que $x_i[2] \leq n - 1$.

Méthode des potentiels. À chaque noeud x_i pour $1 \leq i \leq n$, nous allons associer des potentiels $\Phi_n(x_i), \Phi_{n+1}(x_i), \dots, \Phi_m(x_i) \in \mathbb{N}$, avec $\Phi_n(x_i) = 0$, et associer à chaque instruction o_k (pour $k = n + 1, \dots, m$) sa **complexité amortie**

$$\hat{c}_k := c_k + \sum_{i=1}^n (\Phi_k(x_i) - \Phi_{k-1}(x_i)) \quad (2)$$

où c_k est la complexité de l'instruction o_k . Dans toute la suite de cette partie, c_k et \hat{c}_k sont supposées être des fonctions de n et m . Les fonctions potentielles Φ_k vont être définies de sorte que le coût amorti \hat{c}_k de l'instruction o_k soit en $O(\alpha(n))$.

Question 3.4 Supposons que \hat{c}_k est en $O(\alpha(n))$ et satisfait (2) pour tout $k = n + 1, \dots, m$, et que $\Phi_n(x_i) = 0$ pour tout $i = 1, \dots, n$. Montrer que la complexité de o_{n+1}, \dots, o_m est en $O(m \cdot \alpha(n))$.

Comme d'autre part la complexité de o_k pour $k \leq n$ est en $O(1)$, il s'en suivra que la complexité de o_1, \dots, o_m est en $O(m \cdot \alpha(n))$.

Potentiel d'un noeud. Considérons un noeud \mathbf{x} parmi $\mathbf{x}_1, \dots, \mathbf{x}_n$, qui n'est pas une racine et qui a un rang supérieur ou égal à 1. On pose

$$\begin{aligned} \text{niv}(\mathbf{x}) &:= \max\{k \mid \mathbf{x}[1][2] \geq A_k(\mathbf{x}[2])\} \\ \text{iter}(\mathbf{x}) &:= \max\{i \mid \mathbf{x}[1][2] \geq A_{\text{niv}(\mathbf{x})}^{(i)}(\mathbf{x}[2])\} \end{aligned}$$

Pour chaque $k \geq n$, la fonction Φ_k est définie par

$$\Phi_k(\mathbf{x}) := \begin{cases} \alpha(n) \cdot \mathbf{x}[2] & \text{si } \mathbf{x} \text{ est un racine ou si } \mathbf{x}[2] = 0 \\ (\alpha(n) - \text{niv}(\mathbf{x})) \cdot \mathbf{x}[2] - \text{iter}(\mathbf{x}) & \text{sinon} \end{cases}$$

où $\mathbf{x}[2]$ désigne le rang de \mathbf{x} juste après l'instruction o_k .

Question 3.5 Montrer que $\Phi_n(\mathbf{x}) = 0$.

Quelques propriétés sur les potentiels. Soit $k \geq n+1$ et considérons le noeud \mathbf{x}_i (avec $1 \leq i \leq n$) après l'instruction o_k .

Question 3.6 Supposons que, après l'instruction o_k , \mathbf{x}_i n'est pas une racine et que $\mathbf{x}_i[2] \geq 1$.

- (i) Montrer que $\mathbf{x}_i[1][2] < A_{\alpha(n)}(\mathbf{x}_i[2])$.
- (ii) Montrer que $\text{niv}(\mathbf{x}_i) < \alpha(n)$.
- (iii) Montrer que $\text{iter}(\mathbf{x}_i) \geq 1$.
- (iv) Montrer que $\text{iter}(\mathbf{x}_i) \leq \mathbf{x}_i[2]$.
- (v) Montrer que $\Phi_k(\mathbf{x}_i) < \alpha(n) \cdot \mathbf{x}_i[2]$.
- (vi) Montrer que $\Phi_k(\mathbf{x}_i) \geq 0$.

Question 3.7 Montrer que $0 \leq \Phi_k(\mathbf{x}_i) \leq \alpha(n) \cdot \mathbf{x}_i[2]$.

Coût amortit des opérations. Soit $k \geq n+1$. Nous allons montrer que \hat{c}_k est en $O(\alpha(n))$.

Question 3.8 Supposons que, avant l'instruction o_k , le noeud \mathbf{x}_i n'est pas une racine. Montrer que $\Phi_k(\mathbf{x}_i) \leq \Phi_{k-1}(\mathbf{x}_i)$.

Question 3.9 Supposons que o_k est un appel à `relie`. Montrer que \hat{c}_k est en $O(\alpha(n))$.

Soit $k \geq n+1$ et supposons que o_k est un appel à `repr`. Nous allons montrer que \hat{c}_k est en $O(\alpha(n))$.

Question 3.10 Montrer que pour tout $i = 1, \dots, n$, on a $\Phi_k(\mathbf{x}_i) \leq \Phi_{k-1}(\mathbf{x}_i)$.

Supposons que o_k est un appel à `repr(z)` et supposons qu'il y a s noeuds entre \mathbf{z} et son représentant, c'est-à-dire que si $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_\ell}$ satisfont (1), sont deux à deux distincts, et sont tels que $\mathbf{z} = \mathbf{x}_{i_1}$ et \mathbf{x}_{i_ℓ} est le représentant de \mathbf{z} , alors $s = \ell$.

Question 3.11 Soit $\mathbf{x}_{i_{j_0}}$ de rang ≥ 1 avant o_k et tel qu'il existe $\mathbf{x}_{i_{j_1}}$ avec $j_0 < j_1 < \ell$ et tel que $\text{niv}(\mathbf{x}_{i_{j_0}}) = \text{niv}(\mathbf{x}_{i_{j_1}})$ avant o_k . Notons $\mathbf{x} = \mathbf{x}_{i_{j_0}}$ et $\mathbf{y} = \mathbf{x}_{i_{j_1}}$.

- (i) Soit i la valeur de $\text{iter}(\mathbf{x})$ avant o_k et q la valeur de $\text{niv}(\mathbf{x}) = \text{niv}(\mathbf{y})$ avant o_k . Montrer que avant o_k , on a $\mathbf{y}[1][2] \geq A_q^{(i+1)}(\mathbf{x}[2])$.
- (ii) Soit i la valeur de $\text{iter}(\mathbf{x})$ avant o_k . Montrer que après o_k , on a $\mathbf{x}[1][2] \geq A_q^{(i+1)}(\mathbf{x}[2])$.
- (iii) Montrer que soit $\text{niv}(\mathbf{x})$, soit $\text{iter}(\mathbf{x})$ croît après o_k .
- (iv) Montrer que $\Phi_k(\mathbf{x}) < \Phi_{k-1}(\mathbf{x})$.

Question 3.12 Supposons que o_k est un appel à `repr`. Montrer que \hat{c}_k est en $O(\alpha(n))$.

4 La fonction d'Ackermann

Les fonctions $(A_k)_{k \in \mathbb{N}}$ définies au §3 correspondent essentiellement à la fonction dite d'Ackermann. Cette fonction est connue pour avoir une croissance très rapide. Dans cette partie, on s'intéresse aux fonctions A_k pour $k \leq 4$, en particulier afin de justifier le fait que dans le contexte du §3, on peut dans les applications concrètes toujours supposer $\alpha(n) \leq 4$.

Question 4.1 Montrer que pour tous $i, j \in \mathbb{N}$, on a $A_1^{(i)}(j) = 2^i(j+1) - 1$.

Question 4.2 Soit $t : \mathbb{N} \rightarrow \mathbb{N}$ la fonction définie par $t(0) = 2$ et $t(i+1) = 2^{t(i)}$. Montrer que $t(i) < A_3(i)$ pour tout $i > 0$.

Question 4.3 Rappelons que le nombre d'atomes dans l'univers est de l'ordre de 10^{80} . Montrer que $A_4(1) > 10^{80}$ et justifier pourquoi, pour toute séquence d'instructions o_1, \dots, o_m pratiquement concevable, si n est le nombre d'appels à `cree_racine` dans o_1, \dots, o_m , alors on a $\alpha(n) \leq 4$.

* *
*

**ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES
ÉCOLE SUPÉRIEURE DE PHYSIQUE ET DE CHIMIE INDUSTRIELLES**

CONCOURS D'ADMISSION 2018

**FILIÈRES MP HORS SPECIALITÉ INFO
PC et PSI**

COMPOSITION D'INFORMATIQUE – B – (XELCR)

(Durée : 2 heures)

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

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

Introduction

Une requête sur une base de données est décrite au moyen d'un langage *déclaratif*. Le langage SQL est le plus connu. Pour évaluer une requête, un système de gestion de base de données (SGBD) établit un plan d'exécution combinant les opérateurs de l'*algèbre relationnelle*. L'objectif de ce sujet est l'étude de ces opérateurs.

Nous étudierons en partie I l'implémentation en Python de ces opérateurs. Nous appliquerons ensuite en partie II ces résultats à des requêtes SQL. Nous verrons en partie III comment il est possible de tirer parti des propriétés des données pour améliorer les performances.

Les parties peuvent être traitées indépendamment. Néanmoins, chaque partie utilise des notations et des fonctions introduites dans les parties précédentes.

Notion de complexité. La complexité, ou le coût, d'un algorithme ou d'une fonction Python est le nombre d'opérations élémentaires nécessaires à son exécution dans le pire cas. Lorsque cette complexité dépend d'un ensemble de paramètres (n, p, \dots) , on pourra donner cette estimation sous forme asymptotique. On rappelle qu'une application $c(n, p, \dots)$ est dans la classe $\mathcal{O}(f)$ s'il existe une constante $\alpha > 0$ telle que $|c(n, p, \dots)| < \alpha \times f(n, p, \dots)$, pour toutes les valeurs de n, p, \dots assez grandes. Nous préciserons plus loin le coût de chacune des opérations utilisées dans ce sujet.

Bases de données, tables, attributs, enregistrements

Nous détaillons ici la représentation des données dans le modèle relationnel. Une *base de données* est un ensemble de *tables*. Chaque table porte un nom et est associée à un vecteur d'*attributs* de longueur au moins 1. Le nombre d'attributs d'une table est appelé l'*arité* de la table. Le vecteur des attributs $\langle a_0, a_1, \dots, a_{k-1} \rangle$ d'une table T d'arité k est noté $\text{attributs}(T)$ et la table est notée $T[[a_0, a_1, \dots, a_{k-1}]]$.

Une table $T[[a_0, a_1, \dots, a_{k-1}]]$ est constituée d'*enregistrements*. La *taille* d'une table est le nombre des enregistrements qu'elle contient. Dans ce sujet, nous considérerons qu'un enregistrement est un vecteur $\langle v_0, v_1, \dots, v_{k-1} \rangle$ de longueur l'arité k de la table. Chaque élément de ce vecteur est la valeur de cet enregistrement par rapport à l'attribut correspondant de la table. La valeur v_i à l'indice i de l'enregistrement est la valeur associée à l'attribut a_i à l'indice i du vecteur d'attributs de cette table. On pourra donc identifier un attribut et son indice et parler de *la valeur d'un enregistrement associée à un indice*. La valeur d'un enregistrement e associée à un indice i est notée $e[i]$.

 Nous considérons dans ce sujet que toutes les valeurs d'attributs sont des chaînes de caractères et que la comparaison entre deux valeurs d'un attribut a un coût unitaire quelles que soient ces valeurs.

Deux enregistrements représentés par des vecteurs contenant les mêmes valeurs aux mêmes indices sont égaux.

 Une table peut contenir des enregistrements égaux. L'élimination des enregistrements égaux est une opération complexe qui est l'objet de l'opérateur SQL appelé DISTINCT que nous étudierons plus loin.

Exemple (Tables et enregistrements). Considérons une agence de voyages qui vend des trajets et des chambres d'hôtel. La table

$\text{Vehicule}[[\text{IdVehicule}, \text{Type}, \text{Compagnie}]]$

contient les données relatives aux divers véhicules disponibles. Pour chaque enregistrement e représentant un véhicule dans la table Vehicule, la valeur de e associée à l'attribut IdVehicule est l'identifiant du véhicule ; la valeur de e associée à l'attribut Type est le type de véhicule ; la valeur associée à l'attribut Compagnie est le nom de la compagnie qui gère ce véhicule.

Cette table contient trois *enregistrements* qui décrivent des véhicules : un bus de la compagnie IBUS, un train de la compagnie SNCF et un avion de la compagnie Hop !.

$\langle 98300, \text{Bus}, \text{IBUS} \rangle$
 $\langle 1562, \text{TGV}, \text{SNCF} \rangle$
 $\langle 30990, \text{A320}, \text{Hop !} \rangle$

Considérons d'autre part la table

$\text{Trajet}[[\text{IdTrajet}, \text{VilleD}, \text{VilleA}, \text{DateD}, \text{HeureD}, \text{IdVehicule}]]$

Cette table contient les trajets élémentaires possibles avec les valeurs des attributs associés : l'identifiant du trajet, la ville de départ, la ville d'arrivée, la date du départ de ce trajet, l'heure de départ du trajet, l'identifiant du véhicule utilisé pour le trajet. On rappelle que toutes ces valeurs sont des chaînes de caractères.

Cette table contient 3 trajets possibles le 5 octobre 2016 pour aller de Lille à Rennes. Ils partent respectivement à 9h00, à 10h00 et à 14h00.

$\langle \text{Trajet1}, \text{Lille}, \text{Rennes}, 5 \text{ oct. } 2016, 09h00, 30990 \rangle$
 $\langle \text{Trajet2}, \text{Lille}, \text{Rennes}, 5 \text{ oct. } 2016, 10h00, 98300 \rangle$
 $\langle \text{Trajet3}, \text{Lille}, \text{Rennes}, 5 \text{ oct. } 2016, 14h00, 1562 \rangle$

Représentation des tables et des enregistrements en Python

Dans ce sujet, nous représentons un enregistrement d'une table d'arité k par une *liste Python* de longueur k . L'élément d'indice i de cette liste représente la valeur de l'enregistrement pour l'attribut d'indice i de la table.

Nous représentons une table d'arité k par une liste d'enregistrements. Une table vide est représentée par une liste vide.

Voici par exemple une représentation en Python de la table Véhicule d'arité 3.

```
>>> Vehicule
[['98300', 'Bus', 'IBUS'], ['1562', 'TGV', 'SNCF'], ['30990', 'A320', 'Hop!']]
```

Notez qu'une table peut être représentée par plusieurs listes différentes. Voici une autre représentation possible de cette table.

```
>>> Vehicule
[['1562', 'TGV', 'SNCF'], ['98300', 'Bus', 'IBUS'], ['30990', 'A320', 'Hop!']]
```

Rappel sur les listes Python

Nous rappelons brièvement les opérateurs sur les listes en Python.

 Il est attendu que les candidats rédigent leurs réponses à l'aide de ces fonctions seulement. En particulier, l'opérateur d'égalité entre listes ne doit pas être utilisé.

Longueur. L'opération `len(1)` renvoie la longueur de la liste 1. On considérera que cette opération a un coût unitaire.

Ajout. L'opération `1.append(x)` ajoute l'élément `x` à la fin de la liste 1. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'élément.

Extraction. L'opération `1.pop()` enlève le dernier élément de la liste 1 et renvoie cet élément. Une erreur est signalée si la liste est vide. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'élément.

Accès. L'opération `1[i]` renvoie l'élément d'indice i de la liste 1 de longueur n . Cette opération ne peut être utilisée dans ce cadre qu'avec un indice compris entre 0 et $n - 1$. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'indice.

Concaténation. L'opération `11 + 12` renvoie la concaténation des deux listes 11 et 12. Les listes ne sont pas modifiées. On considérera que cette opération a un coût unitaire.

Itération. Il est possible de parcourir une liste 1 par la commande d'itération

```
for x in 1: ...
```

Ce parcours respecte l'ordre des éléments apparaissant dans la liste. On considérera que le coût d'un parcours est la somme des coûts des opérations effectuées, sans surcoût additionnel.

I Implémentation des opérateurs de l'algèbre relationnelle en Python

 Dans toute la suite, on supposera que les arguments des fonctions Python à rédiger sont bien formés : toutes les listes représentant les enregistrements d'une table ont la même longueur qui est l'arité de cette table, les entiers représentant des indices d'attributs appartiennent bien à l'intervalle attendu, etc.

Sélection avec test d'égalité à une constante

L'opérateur $\sigma_{\text{Constante}}$ prend en arguments une table T d'enregistrements, un attribut de cette table identifié à son indice i dans le vecteur $\text{attributs}(T)$ et une valeur c .

Il renvoie une table T' associée aux mêmes attributs que T . Elle est constituée des enregistrements de T tels que la valeur de l'attribut d'indice i est égale à la valeur c . Cette table peut être vide.

Exemple. $\sigma_{\text{Constante}}(\text{Trajet}, 1, \text{Lille})$ renvoie une table T' avec les mêmes attributs que la table Trajet. Elle contient tous les voyages dont la ville de départ est Lille. Dans notre exemple, c'est le cas de tous les voyages. La table T' contient donc les mêmes enregistrements que la table Trajet.

Question I.1. Implémentez la fonction

```
SelectionConstante(table, indice, constante)
```

qui prend en arguments une table table représentée par une liste d'enregistrements, un entier indice associé à un attribut de cette table table et une valeur constante . Elle renvoie une nouvelle liste représentant la table $\sigma_{\text{Constante}}(\text{table}, \text{indice}, \text{constante})$.

Question I.2. Donnez la complexité de votre implémentation de la fonction `SelectionConstante` par rapport à la taille de la table table . Justifiez votre réponse en vous appuyant sur la structure du programme.

Sélection avec test d'égalité entre deux attributs

L'opérateur $\sigma_{\text{Égalité}}$ prend en arguments une table T d'enregistrements et deux attributs de T identifiés à leurs indices respectifs i et j dans $\text{attributs}(T)$. Notez qu'il est possible que $i = j$.

Il renvoie une table T' associée aux mêmes attributs que T . Elle est constituée des enregistrements de T tels que la valeur pour l'attribut d'indice i est égale à la valeur pour l'attribut d'indice j . Cette table peut être vide.

Exemple. $\sigma_{\text{Égalité}}(\text{Trajet}, 1, 2)$ renvoie une table avec les mêmes attributs que la table Trajet. Elle contient tous les voyages dont la ville de départ est la même que la ville d'arrivée. Le résultat est une table vide.

Question I.3. Implémentez la fonction

```
SelectionEgalite(table, indice1, indice2)
```

qui prend en arguments une table table d'enregistrements et deux attributs identifiés à leurs indices indice1 et indice2. Elle renvoie une nouvelle liste représentant la table $\sigma_{\text{Égalité}}(\text{table}, \text{indice1}, \text{indice2})$.

Projection sur des indices

L'opérateur Π prend en arguments une table d'enregistrements $T[a_0, a_1, \dots, a_{k-1}]$ d'arité k et un vecteur $L = \langle l_0, \dots, l_{k'-1} \rangle$ tel que $0 < k' \leq k$ d'indices identifiant des attributs $\langle a_{l_0}, \dots, a_{l_{k'-1}} \rangle$ de la table T .

-  On se restreint au cas où la liste L est ordonnée dans le sens croissant, sans répétition.
- On supposera que les valeurs du vecteur L sont bien comprises entre 0 et $k - 1$.

L'opérateur Π renvoie la table T' d'arité k' associée au vecteur d'attributs $\langle a_{l_0}, \dots, a_{l_{k'-1}} \rangle$. Les enregistrements de T' sont obtenus à partir des enregistrements de T en conservant uniquement les valeurs de ces enregistrements pour les attributs de T' . Deux enregistrements distincts et différents de T peuvent ainsi créer deux enregistrements égaux dans T' .

Exemple. $\Pi(\text{Trajet}, \langle 1, 2 \rangle)$ renvoie une table associée aux attributs $\langle \text{VilleD}, \text{VilleA} \rangle$.

```
 $\langle \text{Lille}, \text{Rennes} \rangle$ 
 $\langle \text{Lille}, \text{Rennes} \rangle$ 
 $\langle \text{Lille}, \text{Rennes} \rangle$ 
```

Il se trouve dans ce cas précis que tous ces enregistrements sont égaux. La table n'en est pas moins constituée de 3 enregistrements. Sa taille est 3.

Question I.4. Considérons une table d'enregistrements $T[a_0, a_1, \dots, a_{k-1}]$ d'arité k . Implémentez la fonction `ProjectionEnregistrement(enregistrement, listeIndices)` qui prend en arguments un enregistrement enregistrement de cette table et une liste listeIndices $\langle l_0, \dots, l_{k'-1} \rangle$ telle que $0 < k' \leq k$ d'indices identifiant des attributs de cette table. Elle renvoie une nouvelle liste représentant l'enregistrement $\langle a_{l_0}, \dots, a_{l_{k'-1}} \rangle$.

-  On se restreint au cas où la liste listeIndices d'indices est ordonnée dans le sens croissant, sans répétition. On supposera également que tous les indices l_i de listeIndices sont compris entre 0 et $k - 1$.

Question I.5. Implémentez la fonction `Projection(table, listeIndices)` qui prend en arguments une table table d'enregistrements d'arité k et une liste listeIndices $\langle l_0, \dots, l_{k'-1} \rangle$ telle que $0 < k' \leq k$ d'indices identifiant des attributs de cette table dans attributs(table). Cette fonction renvoie une nouvelle liste représentant la table $\Pi(\text{table}, \text{listeIndices})$.

Produit cartésien

L'opérateur \times prend en arguments deux tables T_1 et T_2 d'enregistrements. La table T_1 , d'arité k_1 , est constituée de n_1 enregistrements. La table T_2 , d'arité k_2 , est constituée de n_2 enregistrements. La table T' résultante est d'arité $k_1 + k_2$. Son vecteur d'attributs attributs(T') est la concaténation des vecteurs d'attributs attributs(T_1) et attributs(T_2).

La table T' est constituée de $n_1 \times n_2$ enregistrements. Ces enregistrements sont créés par concaténation de chaque enregistrement de T_1 avec chaque enregistrement de T_2 . Les n_1 premiers attributs sont ceux de T_1 dans l'ordre de T_1 , les n_2 suivants sont ceux de T_2 , dans l'ordre de T_2 . L'ordre des enregistrements ainsi synthétisés dans T' est arbitraire.

Exemple. $\times(\text{Vehicule}, \text{Trajet})$ renvoie une table T' . Les enregistrements de T' sont formés par la concaténation deux à deux des enregistrements de la table Vehicule et de ceux de la table Trajet.

```

⟨98300, Bus, IBUS, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨98300, Bus, IBUS, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨98300, Bus, IBUS, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩
⟨1562, TGV, SNCF, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨1562, TGV, SNCF, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨1562, TGV, SNCF, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩
⟨30990, A320, Hop !, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨30990, A320, Hop !, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨30990, A320, Hop !, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩

```

Question I.6. Implémentez la fonction ProduitCartesien(table1, table2) qui prend en arguments deux tables table1 et table2 d'enregistrements. Elle renvoie une nouvelle liste représentant la table $X(\text{table1}, \text{table2})$.

Jointure

L'opérateur \bowtie prend en arguments deux tables, T_1 d'arité k_1 et de taille n_1 , et T_2 d'arité k_2 et de taille n_2 . Il prend aussi en arguments un attribut de T_1 identifié par son indice i_1 tel que $0 \leq i_1 < k_1$ dans le vecteur attributs(T_1) noté A_1 et un attribut de T_2 identifié par son indice i_2 tel que $0 \leq i_2 < k_2$ dans le vecteur attributs(T_2) noté A_2 . Posons $A_2 = \langle a_0, \dots, a_{i_2}, \dots, a_{k_2-1} \rangle$.

La table T' résultante est d'arité $k_1 + k_2 - 1$. Son vecteur d'attributs attributs(T') est la concaténation du vecteur A_1 et du vecteur A'_2 défini par $\langle a_0, \dots, a_{i_2-1}, a_{i_2+1}, \dots, a_{k_2-1} \rangle$, obtenu en effaçant la coordonnée i_2 de A_2 .

La table T' est constituée d'au plus $n_1 \times n_2$ enregistrements. Les enregistrements de T' sont créés par concaténation des enregistrements e_1 de T_1 et e_2 de T_2 tels que $e_1[i_1] = e_2[i_2]$, en supprimant la valeur d'indice $k_1 + i_2$ pour éviter la répétition avec celle d'indice i_1 . L'enregistrement résultant de cette opération est appelé *jointure* des deux enregistrements e_1 et e_2 . Notez qu'il est possible que plusieurs couples (e_1, e_2) produisent des jointures égales dans T' .

Exemple. $\bowtie(\text{Vehicule}, \text{Trajet}, 0, 5)$ renvoie les enregistrements qui décrivent les voyages de chaque véhicule suivi des informations le concernant.

```

⟨98300, Bus, IBUS, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00⟩
⟨1562, TGV, SNCF, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00⟩
⟨30990, A320, Hop !, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00⟩

```

Question I.7. Implémentez la fonction

```
Jointure(table1, table2, indice1, indice2)
```

qui prend en arguments deux tables `table1` et `table2` et deux entiers représentant respectivement la position d'un attribut de `table1` et celle d'un attribut de `table2`. Elle renvoie une nouvelle liste représentant la table $\bowtie(\text{table1}, \text{table2}, \text{indice1}, \text{indice2})$.

 On pourra commencer par implémenter une fonction qui prend en arguments deux enregistrements e_1 et e_2 et deux indices i_1 et i_2 tels que $e_1[i_1] = e_2[i_2]$ et qui renvoie leur jointure au sens ci-dessus.

Question I.8. Donnez la complexité de votre implémentation de

```
Jointure(table1, table2, indice1, indice2)
```

par rapport aux tailles et arités respectives des tables `table1` et `table2`. Justifiez votre réponse en vous appuyant sur la structure du programme.

Distinct

Nous ajoutons aux opérateurs précédents un nouvel opérateur `Distinct` qui n'appartient pas à l'algèbre relationnelle classique. Cet opérateur permet de supprimer les répétitions d'enregistrements égaux dans une table T . Il renvoie une table T' qui est associée aux mêmes attributs que T . Cette table contient exactement un représentant pour chaque classe d'enregistrements égaux de T .

Exemple. `Distinct($\Pi(\text{Trajet}, \langle 1, 2 \rangle)$)` renvoie une table avec un unique représentant de chaque couple possible de villes de départ et d'arrivée. Cette table ne contient qu'un seul enregistrement : $\langle \text{Lille}, \text{Rennes} \rangle$.

Question I.9. Implémentez la fonction

```
SupprimerDoublons(table)
```

qui prend en argument une table `table`. Elle renvoie une nouvelle liste représentant la table `Distinct(table)`.

 On rappelle que l'opérateur Python d'égalité entre listes ne doit pas être utilisé dans ce sujet. Il est seulement possible de tester l'égalité de deux valeurs associées à un même attribut.

Question I.10. Donnez la complexité de votre implémentation de

```
SupprimerDoublons(table)
```

par rapport à la taille et l'arité de la table `table`. Justifiez votre réponse en vous appuyant sur la structure du programme.

II Implémentation de requêtes SQL en Python

Les données de notre agence de voyage sont enregistrées par les tables suivantes.

Vehicule(IdVehicule, Type, Compagnie) : enregistre les véhicules disponibles — l'identifiant du véhicule, son type et sa compagnie.

Trajet(IdTrajet, VilleD, VilleA, IdVehicule) : enregistre les trajets élémentaires possibles — l'identifiant du trajet, la ville de départ, la ville d'arrivée ainsi que le véhicule utilisé.

Ticket(IdTicket, IdTrajet, Place, Date, Heure, Prix) : enregistre les tickets disponibles — l'identifiant du ticket, l'identifiant du trajet auquel ce ticket donne accès, le numéro de la place, la date, l'horaire et le prix.

Hotel(IdHotel, Classe, Ville) : enregistre les hôtels connus — l'identifiant de l'hôtel, sa classe et sa ville.

Chambre(IdReservation, IdHotel, Date, Prix) : enregistre les chambres d'hôtel qui sont disponibles — l'identifiant de réservation à utiliser, l'identifiant de l'hôtel où se trouve la chambre, la date et le prix.

L'objectif de cette partie est d'étudier l'implémentation de requêtes SQL en combinant les fonctions de l'algèbre relationnelle présentées dans la partie I. Tout commentaire expliquant et justifiant la traduction sera apprécié.

Par convention, la liste Python représentant une table aura le même nom que cette table. On représentera un attribut avec sa position. Par exemple, l'attribut `IdTrajet` de la table `Trajet` est représenté par l'entier 0.

Dans chaque cas, le résultat de la requête sera affecté à une variable nommée `resultat`. Par exemple, la requête SQL

```
SELECT Vehicule.Compagnie FROM Vehicule
```

pourra être implémentée par

```
resultat = Projection(Vehicule, [2])
```

en supposant que la variable Python `Vehicule` représente la table `Vehicule`. Dans des cas plus complexes, on pourra simplifier l'expression en utilisant des variables auxiliaires pour stocker la valeur de certaines sous-expressions, comme dans l'exemple suivant.

```
r1 = Vehicule
resultat = Projection(r1, [2])
```

 Il est attendu que les candidats rédigent leurs réponses en combinant uniquement les fonctions de l'algèbre relationnelle présentées dans la partie I, à l'exclusion de toute autre fonction ou structure de contrôle Python.

Question II.1. Proposez une implémentation pour la requête suivante.

```
SELECT *
  FROM Trajet
 WHERE Trajet.VilleD = Rennes
```

Question II.2. Proposez une implémentation pour la requête suivante.

```
SELECT *
  FROM Trajet, Vehicule
```

Question II.3. Proposez une implémentation pour la requête suivante.

```
SELECT *
  FROM Trajet, Vehicule
 WHERE Trajet.IdVehicule = Vehicule.IdVehicule
```

Question II.4. Proposez une implémentation pour la requête suivante.

```
SELECT Classe, Ville, Date, Prix
  FROM Hotel JOIN Chambre
    ON Hotel.IdHotel = Chambre.IdHotel
```

Question II.5. Proposez une implémentation pour la requête suivante.

```
SELECT Hotel.IdHotel
  FROM Hotel, Trajet, Ticket
 WHERE Hotel.Ville = Trajet.VilleA
   AND Trajet.IdTrajet = Ticket.IdTrajet
   AND Ticket.Prix = '50'
```

Question II.6. Proposez une implémentation pour la requête suivante.

```
SELECT *
  FROM Chambre
 WHERE Chambre.Prix = '100'
   AND Chambre.IdHotel IN
    (SELECT Hotel.IdHotel
      FROM Hotel, Trajet, Ticket
     WHERE Hotel.Ville = Trajet.VilleA
       AND Trajet.IdTrajet = Ticket.IdTrajet
       AND Ticket.Prix = '50')
```

III Amélioration des performances

Il est possible dans certains cas d'améliorer l'implémentation d'une requête en tenant compte de propriétés particulières de la représentation des données ou en utilisant des structures de données supplémentaires. Dans cette partie, nous allons montrer que l'on peut amé-

liorer les performances en triant les données avant de les traiter ou en utilisant des tables associatives (dictionnaires) auxiliaires.

Tables triées par rapport à un indice

Une table d'arité k est représentée par une liste d'enregistrements, eux-mêmes représentés par des listes à k éléments. Supposons tout d'abord avoir à disposition une fonction

```
TrieTableIndice(table, indice)
```

qui trie par ordre croissant suivant l'ordre lexicographique les enregistrements de la liste `table` d'arité k par rapport à la valeur de l'attribut d'indice `indice` dans le vecteur des attributs de cette table. On suppose que la valeur `indice` est strictement inférieure à k .

Par exemple, la liste `Trajet` ci-dessous est triée par rapport à l'attribut d'indice 1 pour l'ordre lexicographique $<$ sur les chaînes de caractères de Python.

```
>>> Trajet
[['30990', 'A320', 'Hop!'], ['98300', 'Bus', 'IBUS'], ['1562', 'TGV', 'SNCF']]
```

Question III.1. *Implémentez la fonction `VerifieTrie(table, indice)` qui renvoie True si la table `table` est triée pour l'indice `indice` et False sinon.*

Question III.2. *Considérez la fonction de la question I.1.*

```
SelectionConstante(table, indice, constante)
```

Hypothèse. *On suppose que la liste représentant la table `table` est triée selon l'indice `indice`.*

Proposez une implémentation de cette fonction qui utilise cette hypothèse pour améliorer les performances. Elle sera nommée comme suit :

```
SelectionConstanteTrie(table, indice, constante)
```

Question III.3. *Considérez la fonction de la question I.7.*

```
Jointure(table1, table2, indice1, indice2)
```

Hypothèse. *On suppose dans cette question que les enregistrements de la table `table1` ont des valeurs deux à deux distinctes pour l'attribut d'indice `indice1`, et de même pour les enregistrements de la table `table2` avec l'indice `indice2`.*

On suppose de plus que la liste représentant la table `table1` est triée selon l'indice `indice1` et que celle représentant la table `table2` est triée selon l'indice `indice2`.

Proposez une implémentation de cette fonction qui utilise cette hypothèse pour améliorer les performances. Elle sera nommée comme suit :

```
JointureTrie(table1, table2, indice1, indice2)
```

Question III.4. Donnez la complexité de votre implémentation en vous appuyant sur la structure du programme. Donnez des exemples pour lesquels cette nouvelle approche est plus performante. Y a-t-il des cas où elle n'est pas plus performante ?

Utilisation d'un dictionnaire (index)

Dictionnaires Python

Un dictionnaire est une structure de données de Python qui permet d'associer à une clé c une valeur v . On parle aussi de *table d'association*. Dans notre cas, les clés sont des chaînes de caractères et les valeurs sont des listes d'entiers.

Nous donnons ici quelques opérations permettant de manipuler les dictionnaires en Python.

 Il est attendu que les candidats rédigent leurs réponses en utilisant exclusivement ces opérations pour manipuler les dictionnaires.

Création d'un dictionnaire. L'opération `dico = {}` crée le dictionnaire `dico` et l'initialise à vide. Cette opération a un coût unitaire.

Ajout d'une association. L'opération `dico[c] = liste` ajoute au dictionnaire une association entre la clé c et la liste `liste`. Si une association existait déjà pour la clé dans le dictionnaire, celle-ci est perdue. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et des valeurs de la clé et de la liste.

Extraction d'une clé. L'opération `dico[c]` renvoie la liste associée à la clé c dans le dictionnaire `dico`. Cette opération n'est autorisée que si la clé c est effectivement associée à une valeur dans le dictionnaire `dico`. Si aucune association n'existe pour la clé, une erreur se produit. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et de la valeur de la clé.

Test de présence. L'opération `c in dico` renvoie `True` si la clé c est associée à une valeur dans le dictionnaire `dico` et `False` sinon. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et de la valeur de la clé.

On peut itérer sur les clés présentes dans un dictionnaire `dico` par la commande

```
for c in dico: ...
```

Voici un exemple d'utilisation.

```
>>> dico = {}
>>> dico['aaa'] = [1]
>>> dico['bbb'] = [2]
>>> dico['ccc'] = [3]
>>> dico['aaa'].append(4)
>>> dico['ccc'] = [5]
>>> for c in dico: print c, '→', dico[c]
aaa → [1, 4]
bbb → [2]
ccc → [5]
>>> dico['ddd']
KeyError: 'ddd'
```

Utilisation de dictionnaires pour indexer les bases de données

Une autre idée pour explorer les tables efficacement est d'utiliser des dictionnaires.

Considérons une table T d'arité k et de taille n représentée par une liste Python d'enregistrements $[e_0, \dots, e_{n-1}]$. Soit L cette liste. Considérons un indice i d'attribut de T tel que $0 \leq i < k$.

On peut associer à T et i un dictionnaire Python $Dico_{T,i}$ de la manière suivante. Les clés de ce dictionnaire sont les valeurs possibles pour les valeurs v qui apparaissent pour l'attribut d'indice i dans la table T . L'image associée à une clé v est la liste des positions dans L des enregistrements e tels que $e[i] = v$.

Si l'attribut d'indice i de T ne prend la valeur v pour aucun enregistrement, cette clé n'est pas enregistrée dans le dictionnaire. L'image d'une clé est donc une liste non vide.

Exemple (Dictionnaire associé à une table). Considérons la table

```
Vehicule[[IdVehicule, Type, Compagnie]]
```

avec les enregistrements suivants.

```
<98300, Bus, IBUS>
<1562, TGV, SNCF>
<30990, A320, Hop !>
<1789, TGV, SNCF>
```

Soit $dico$ le dictionnaire $Dico_{Vehicule,1}$ associé à l'attribut Type de position 1.

Nous avons

```
>>> for c in dico: print c, '→', dico[c]
Bus → [0]
A320 → [2]
TGV → [1, 3]
>>> dico['Ariane6']
KeyError: 'Ariane6'
```

Application à la sélection

Question III.5. Implémentez la fonction `CreerDictionnaire(table, indice)` qui prend en arguments une table $table$ et un indice indice d'attribut de table. Elle renvoie un dictionnaire de la table $table$ pour l'attribut d'indice $indice$.

Question III.6. Considérez la fonction de la question I.1.

```
SelectionConstante(table, indice, constante)
```

Implémentez la fonction

```
SelectionConstanteDictionnaire(table, indice, constante, dico)
```

qui a la même fonctionnalité que `SelectionConstante`, mais qui prend en plus en argument un dictionnaire $dico$ de la table $table$ pour l'indice $indice$.

Question III.7. Comparez la complexité de la fonction SelectionConstanteDictionnaire avec celle de la fonction SelectionConstante. Donnez des exemples pour lesquels cette nouvelle approche est plus performante. Y a-t-il des cas où elle n'est pas plus performante ?

Application à la jointure

Question III.8. Considérez la fonction de la question I.7.

```
Jointure(table1, table2, indice1, indice2)
```

Implémentez la fonction

```
JointureDictionnaire(table1, table2, indice1, indice2, dico2)
```

qui a la même fonctionnalité que Jointure, mais qui prend en plus en argument un dictionnaire dico2 pour la table table2 par rapport à l'indice indice2.

Question III.9. Donnez la complexité de votre implémentation par rapport aux tailles et arités respectives des tables table1 et table2 ainsi que par rapport à la longueur maximale d'une liste renvoyée par le dictionnaire dico2, qui sera notée k_2 . Justifiez votre réponse en vous appuyant sur la structure du programme.

Question III.10. L'opérateur de jointure prend en arguments deux tables qui jouent des rôles analogues. Il serait donc possible d'utiliser un dictionnaire pour table1 au lieu d'un dictionnaire pour table2. Comment pourrait-on choisir la table à indexer pour obtenir les meilleures performances ?

A2018 – INFO

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

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

CONCOURS 2018**ÉPREUVE D'INFORMATIQUE COMMUNE**

Durée de l'épreuve : 1 heure 30 minutes

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

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

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

INFORMATIQUE COMMUNE

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

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

MESURES DE HOULE

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

On s'intéresse à des mesures de niveau de la surface libre de la mer effectuées par une bouée (représentée sur la Figure 1)¹. Cette bouée contient un ensemble de capteurs incluant un accéléromètre vertical qui fournit, après un traitement approprié, des mesures à étudier².

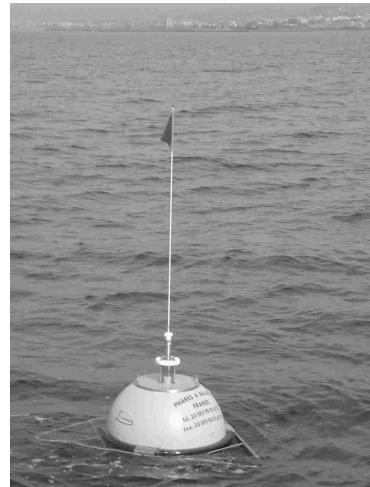


FIGURE 1 : bouée de mesure de houle

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

Partie I. Stockage interne des données

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

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

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

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

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

2. L'ensemble des paramètres des états de mer présent dans la base CANDHIS est calculé par les logiciels :

- Houle5 (CETMEF) : analyse vague par vague (temporelle) ;
- PADINES (EDF/LNHE) : analyse spectrale et directionnelle (fréquentielle).

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

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

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

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

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

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

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

Partie II. Analyse "vague par vague"

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

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

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

On suppose $\eta(0) > m$, et $\frac{d\eta}{dt}(0) > 0$.

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

Les hauteurs de vagues H_i sont définies par les différences :

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

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

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

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

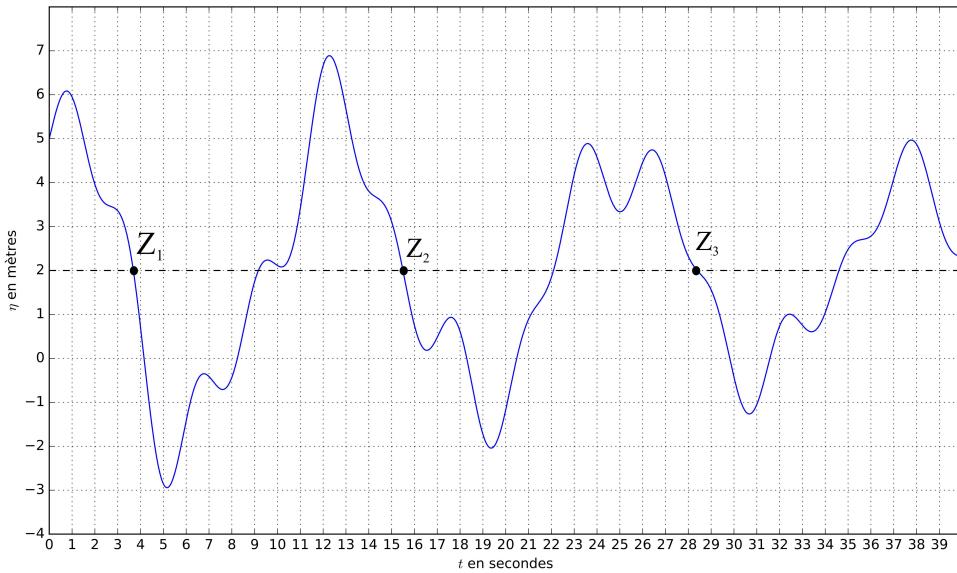


FIGURE 2 : Passages par le Niveau moyen en Descente (PND). Ici la moyenne m vaut 2

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

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

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

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

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

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

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

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

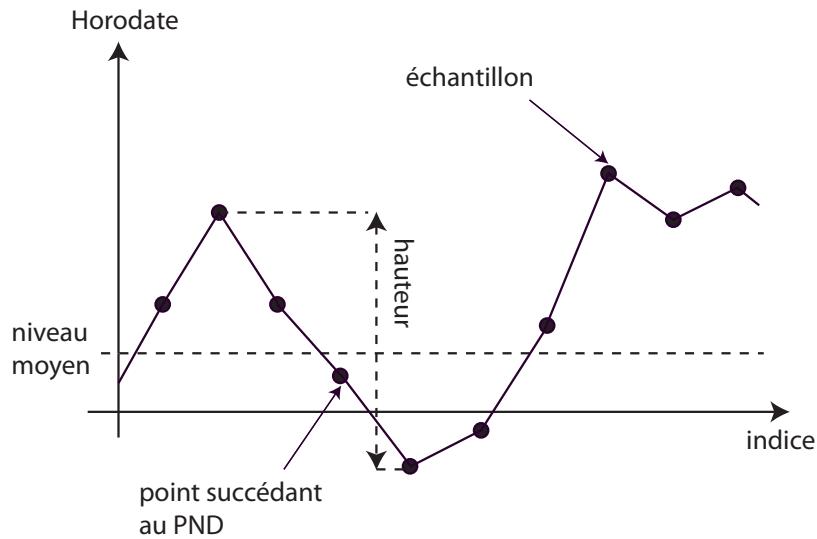


FIGURE 3 : propriétés d'une vague

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

On désire maintenant caractériser les vagues.

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

- H_i , sa hauteur en mètres (m) (voir Figure 3),
- T_i , sa période en secondes (s).

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

Partie III. Contrôle des données

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

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

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

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

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

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

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

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

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

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

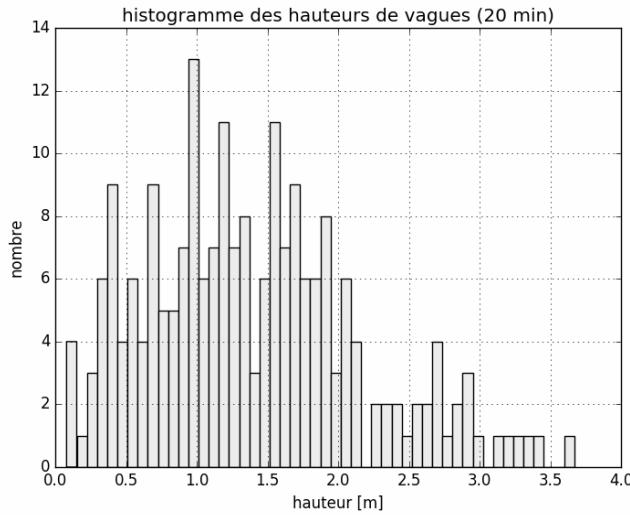


FIGURE 4 : histogramme des hauteurs de vague

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

On définit alors

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

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

Le test suivant est appliqué :

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

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

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

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

Partie IV. Base de données relationnelle

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

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

	<code>idBouee</code>	<code>nomSite</code>	<code>localisation</code>	<code>typeCapteur</code>	<code>frequence</code>
<code>Bouee</code>	831	Porquerolles	Mediterranee	Datawell non directionnelle	2.00
	291	Les pierres noires	Mer d'iroise	Datawell directionnelle	1.28

La seconde table est `Campagne`.

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

	<code>idCampagne</code>	<code>idBouee</code>	<code>debutCampagne</code>	<code>finCampagne</code>
<code>Campagne</code>	08301	831	01/01/2010 00h00	15/01/2010 00h00
	02911	291	15/10/2005 18h30	18/10/2005 08h00

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

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

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

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

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

□ Q19 – Formuler les requêtes SQL permettant de répondre aux questions suivantes :

- “Quels sont le numéro d'identification et le nom de site des bouées localisées en Méditerranée ?”

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

Partie V. Analyse “spectrale”

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

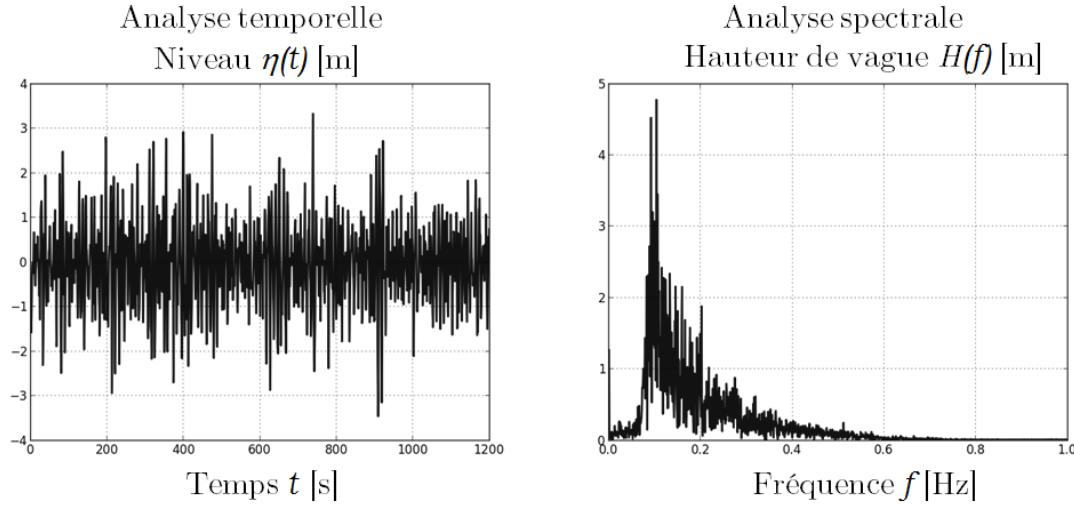


FIGURE 5 : analyses temporelle et spectrale

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

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

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

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

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

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

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

On pose

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

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

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

On montre alors que pour $0 \leq k < \frac{N}{2}$,

$$\begin{cases} X_k = P_k + w^k I_k \\ X_{k+N/2} = P_k - w^k I_k \end{cases}$$

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

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

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

Annexe

Algorithm 1

```

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

```

Algorithm 2

```

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

```

Algorithm 3

```

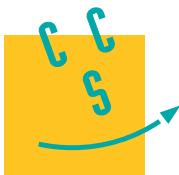
1 def triInsertion(liste,g,d):
2     for i in range(g+1,d+1):
3         j=i-1
4         tmp = liste[i]
5         while                      # A compléter
6
7
8
9
10
11
12
13         liste[j+1]=tmp

```

Algorithm 4

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

Fin de l'épreuve.



CONCOURS CENTRALE-SUPÉLEC

Informatique

MP, PC, PSI, TSI

2018

3 heures

Calculatrices autorisées

Simulation de la cinétique d'un gaz parfait

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

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

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

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

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

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

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

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

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

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

Représentation en Python

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

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

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

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

Figure 1 Exemples de particules

I Initialisation

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

I.A – Placement en dimension 1

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



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

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

```

1  def placement1D(N:int, R:float, L:float) -> [np.ndarray]:
2
3      def possible(c:np.ndarray) -> bool:
4          if c[0] < R or c[0] > L - R: return False
5          for p in res:
6              if abs(c[0] - p[0]) < 2*R: return False
7          return True
8
9      res = []
10     while len(res) < N:
11         p = L * np.random.rand(1)
12         if possible(p): res.append(p)
13
14     return res

```

- Q 1.** Détails l'action de la ligne 9.
- Q 2.** Quelle est la signification du paramètre `c` de la fonction `possible` (ligne 2) ?
- Q 3.** Expliquer le rôle de la ligne 3.
- Q 4.** Expliquer le rôle des lignes 4 et 5.
- Q 5.** Donner en une phrase le rôle de la fonction `possible`.
- Q 6.** Proposer une nouvelle version de la ligne 9 permettant d'éviter certains rejets de la part de la fonction `possible`.
- Q 7.** On considère l'appel `placement1D(4, 0.5, 5)` et on suppose que les trois premières particules ont été placées aux points d'abscisses 1, 2,5 et 4 (figure 3). Quelle sera la suite du déroulement de la fonction `placement1D` ?

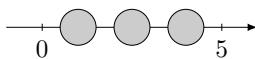


Figure 3

- Q 8.** Quelle est la complexité temporelle de la fonction `placement1D` dans le cas où $N \ll N_{\max}$, nombre maximal de particules de rayon R pouvant être placées sur un segment de longueur L ?
- Q 9.** Pour remédier de manière simple (mais non optimale) à la situation de la question 7, on décide de recommencer à zéro le placement des particules dès qu'une particule est rejetée par la fonction `possible`. Réécrire les lignes 7 à 11 de la fonction `placement1D` pour mettre en œuvre cette décision.

I.B – Optimisation du placement en dimension 1

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

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

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

Q 10. Écrire la fonction d'entête

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

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

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

I.C – Analyse statistique

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

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

I.D – Dimension quelconque

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

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

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

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

II Mouvement des particules

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

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

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

II.A – Analyse physique

Q 14. Comment évolue une particule entre deux événements ?

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

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

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

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

II.B – Évolution des particules

Q 17. Écrire la fonction d'entête

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

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

Q 18. Écrire la fonction d'entête

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

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

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

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

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

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

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

III Inventaire des événements

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

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

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

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

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

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

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

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

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

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

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

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

III.B – Catalogue d'évènements

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

Q 22. Écrire la fonction d'entête

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

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

Q 23. Écrire la fonction d'entête

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

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

Q 24. Écrire la fonction d'entête

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

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

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

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

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

IV Simulation

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

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

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

Q 29. Écrire la fonction d'entête

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

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

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

Q 30. Écrire la fonction d'entête

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

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

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

On dispose de la fonction d'entête

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

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

Q 31. Écrire la fonction d'entête

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

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

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

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

V Exploitation des résultats

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

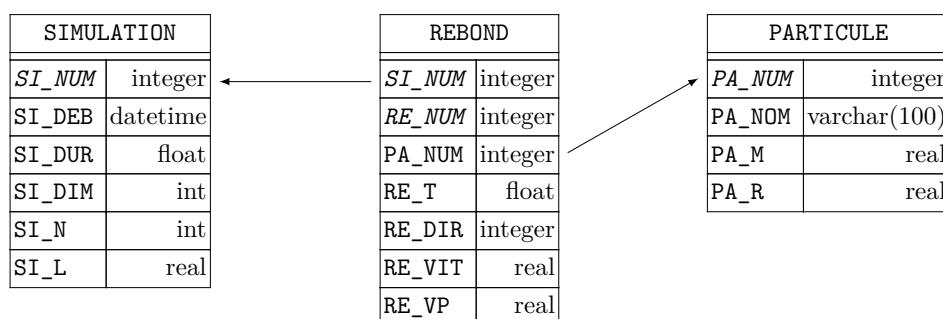


Figure 4 Structure physique de la base de données des résultats de simulation

Cette base comporte les trois tables suivantes :

- la table `SIMULATION`, de clef primaire `SI_NUM`, donne les caractéristiques de chaque simulation effectuée. Elle contient les colonnes
 - `SI_NUM` numéro d'ordre de la simulation (clef primaire)
 - `SI_DEB` date et heure du lancement du programme de simulation
 - `SI_DUR` durée (en secondes) de la simulation (il ne s'agit pas du temps d'exécution du programme, mais du temps simulé)
 - `SI_DIM` nombre de dimensions de l'espace de simulation
 - `SI_N` nombre de particules pour cette simulation
 - `SI_L` (en mètres) taille du récipient utilisé pour la simulation
- la table `PARTICULE`, de clef primaire `PA_NUM`, des types de particules considérées. Elle contient les colonnes
 - `PA_NUM` numéro (entier) identifiant le type de particule (clef primaire)
 - `PA_NOM` nom de ce type de particule
 - `PA_M` masse de la particule (en grammes)
 - `PA_R` rayon (en mètres) de la particule

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

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

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

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

Opérations et fonctions Python disponibles

Fonctions

- `range(n)` renvoie la séquence des n premiers entiers ($0 \rightarrow n - 1$)
- `list(range(n))` renvoie une liste contenant les n premiers entiers dans l'ordre croissant :


```
list(range(5)) → [0, 1, 2, 3, 4]
```
- `random.randrange(a, b)` renvoie un entier aléatoire compris entre a et $b-1$ inclus (a et b entiers)
- `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1]$ suivant une distribution uniforme
- `random.shuffle(u)` permute aléatoirement les éléments de la liste u (modifie u)
- `random.sample(u, n)` renvoie une liste de n éléments distincts de la liste u choisis aléatoirement, si $n > len(u)$, déclenche l'exception `ValueError`
- `math.sqrt(x)` calcule la racine carrée du nombre x
- `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à x
- `math.floor(x)` renvoie le plus grand entier inférieur ou égal à x
- `sorted(u)` renvoie une nouvelle liste contenant les éléments de la liste u triés dans l'ordre « naturel » de ses éléments (si les éléments de u sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les listes

- `len(u)` donne le nombre d'éléments de la liste u :


```
len([1, 2, 3]) → 3 ; len([[1,2], [3,4]]) → 2
```
- $u + v$ construit une liste constituée de la concaténation des listes u et v :


```
[1, 2] + [3, 4, 5] → [1, 2, 3, 4, 5]
```
- $n * u$ construit une liste 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

```
2 in [1, 2, 3] → True ; 2 not in [1, 2, 3] → False
```
- `u.append(e)` ajoute l'élément e à la fin de la liste u (similaire à $u = u + [e]$)
- `u.pop()` renvoie le dernier élément de la liste u ($u[-1]$) et le supprime (`del u[-1]`)
- `del u[i]` supprime de la liste u son élément d'indice i
- `del u[i:j]` supprime de la liste u tous ses éléments dont les indices sont compris dans l'intervalle $[i, j[$
- `u.remove(e)` supprime de la liste u le premier élément qui a pour valeur e , déclenche l'exception `ValueError` si e ne figure pas dans u
- `u.insert(i, e)` insère l'élément e à la position d'indice i dans la liste u (en décalant les éléments suivants) ; si $i \geq len(u)$, e est ajouté en fin de liste

- `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`
- `u.sort()` trie la liste `u` en place, dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

Opérations sur les tableaux (`np.ndarray`)

- `np.array(u)` crée un nouveau tableau contenant les éléments de la liste `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`
- `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou une matrice à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique numpy (`np.int16`, `np.float32`, ...). Si le paramètre `dtype` n'est pas précisé, les éléments seront de type `float`
- `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens
- `np.random.rand(n)`, `np.random.rand(n, m)` crée un tableau de la forme indiquée (`n` lignes, `m` colonnes) en initialisant chaque élément avec une valeur aléatoire issue d'une distribution uniforme sur $[0, 1[$
- `a.ndim` nombre de dimensions du tableau `a` (1 pour un vecteur, 2 pour une matrice, etc.)
- `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions
- `len(a)` taille du tableau `a` dans sa première dimension (nombre d'éléments d'un vecteur, nombre de lignes d'une matrice, etc.) équivalent à `a.shape[0]`
- `a.size` nombre total d'éléments du tableau `a`
- `a.flat` itérateur sur tous les éléments du tableau `a`
- `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O(a.size)$
- `b in a` détermine si `b` est un élément du tableau `a` ; si `b` est un scalaire, vérifie si `b` est un élément de `a` ; si `b` est un vecteur ou une liste et `a` une matrice, détermine si `b` est une ligne de `a`
- `np.concatenate((a1, a2))` construit un nouveau tableau en concaténant deux tableaux ; `a1` et `a2` doivent avoir le même nombre de dimensions et la même taille à l'exception de leur taille dans la première dimension (deux matrices doivent avoir le même nombre de colonnes pour pouvoir être concaténées)
- `a.sort(d)` trie le tableau `a` en place suivant sa dimension d'indice `d` (par défaut, la dernière du tableau) : `a.sort(0)` trie les éléments du vecteur `a` ou les lignes de la matrice `a` ; `a.sort(1)` trie les colonnes de la matrice `a`
- `np.sort(a, d)` renvoie une copie triée du tableau `a` suivant sa dimension d'indice `d` (voir `a.sort(d)` pour la signification exacte du paramètre `d`)

• • • FIN • • •

**ÉPREUVE SPÉCIFIQUE - FILIÈRE PC****MODÉLISATION DE SYSTÈMES PHYSIQUES OU CHIMIQUES****Jeudi 3 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 deux parties (pages 1 à 14)
et d'une annexe (pages 15 à 18).**

PROBLÈME

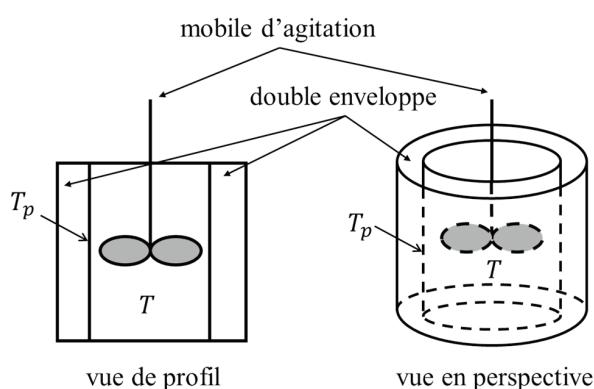
Étude d'une réaction exothermique : stabilité thermique en réacteur fermé

Présentation du problème

De nombreux procédés industriels font intervenir des réacteurs fermés pour la synthèse de molécules à haute valeur ajoutée. Pour optimiser le rendement de la synthèse, il est nécessaire de bien comprendre l'influence des paramètres physiques (comme la température de la réaction,...) sur la marche du réacteur. La maîtrise des échanges thermiques est cruciale dans le cas des réactions exothermiques car la chaleur dégagée par la réaction provoque une augmentation de la température du mélange réactionnel. Selon les conditions opératoires, cette augmentation de température peut entraîner un emballement thermique du réacteur et conduire à des dégâts irréversibles.

L'accident de Seveso le 10 juillet 1976 illustre les problèmes liés à l'emballement thermique des réacteurs. Il s'agissait d'un procédé de production de 2,4,5-trichloro-phénol à partir de 1,2,4,5-tétrachloro-benzène et de soude dans un solvant (l'éthylène glycol) à une température voisine de 150 °C et à pression atmosphérique en réacteur fermé. La mauvaise maîtrise de la température de la réaction a entraîné le déroulement de réactions secondaires conduisant d'une part à une augmentation de la température et de la pression et d'autre part à la formation de produits secondaires toxiques : les dioxines. La rupture de la soupape de sécurité due à l'augmentation de la pression a conduit au rejet de dioxines dans l'atmosphère.

L'étude de l'influence des paramètres physiques sur la marche d'un réacteur se fait la plupart du temps à l'échelle du laboratoire dans des dispositifs de dimensions beaucoup plus petites que celles des réacteurs utilisés dans les procédés industriels. La particularité du réacteur utilisé pour la présente étude est qu'il possède une géométrie cylindrique et qu'il est équipé d'une double enveloppe externe dans laquelle circule un fluide permettant de refroidir la paroi du réacteur et d'empêcher un emballement thermique (**figure 1**). L'emballement thermique survient lorsque la chaleur dégagée par la réaction excède la capacité du réacteur à évacuer l'énergie.



T et T_p représentent respectivement la température dans le réacteur et la température à la paroi (côté refroidissement).

Figure 1 – Schéma simplifié d'un réacteur fermé parfaitement agité avec une double enveloppe pour son refroidissement

Pour caractériser le comportement thermique du réacteur, on commence la plupart du temps par une étude en l'absence de réaction. Cette étude permet dans un premier temps de caractériser la capacité du réacteur à évacuer l'énergie avec la détermination du coefficient de transfert thermique à la paroi. Dans un deuxième temps, on met en œuvre dans ce réacteur une réaction exothermique $R \rightarrow \text{produits}$. Ces études permettent de déterminer les valeurs de paramètres clefs intervenant dans les équations décrivant le comportement du réacteur (modèle théorique). L'utilisation de ce modèle théorique permet de prédire la stabilité thermique du réacteur. L'établissement du modèle théorique repose sur l'écriture de bilans de matière et de chaleur. Une fois que l'influence des conditions physiques sur la marche du réacteur est déterminée, on peut en déduire les conditions de stabilité du réacteur industriel.

Ce sujet est constitué de deux parties. La première partie porte sur la modélisation du réacteur fermé parfaitement agité avec double enveloppe. Elle permet l'établissement du système d'équations différentielles régissant les variations de la conversion du réactif et de la température en fonction du temps, ainsi que la détermination de la valeur de paramètres physico-chimiques intervenant dans ces équations. La deuxième partie porte sur le traitement numérique des données expérimentales avec la détermination des paramètres d'un modèle par régression linéaire, puis la prédiction du comportement thermique du réacteur par résolution d'un système d'équations différentielles par la méthode d'Euler implicite.

Partie I – Modélisation du réacteur fermé parfaitement agité avec double enveloppe

I.1 – Etalonnage thermique du réacteur

Dans cette partie, on souhaite caractériser les transferts de chaleur entre un liquide contenu à l'intérieur du réacteur et la paroi en l'absence de toute réaction chimique. On supposera que la capacité thermique massique et la masse volumique de ce liquide sont constantes quelle que soit la température. Pour caractériser ces transferts de chaleur, on réalise deux expériences successives avec la température de la paroi, T_p , maintenue constante dans les deux cas.

- La première expérience consiste à chauffer le liquide (initialement à une température identique à celle de la paroi) avec un dispositif annexe (une résistance chauffante) dissipant une puissance thermique P_{th} de 96,0 W. On constate que la température de la phase liquide augmente, puis atteint une valeur asymptotique en régime permanent (**figure 2a**).
- Après avoir atteint le régime permanent lors de la phase de chauffe, on réalise une seconde expérience en coupant le chauffage. La température du liquide décroît jusqu'à ce qu'elle tende vers la température de la paroi (**figure 2b**).

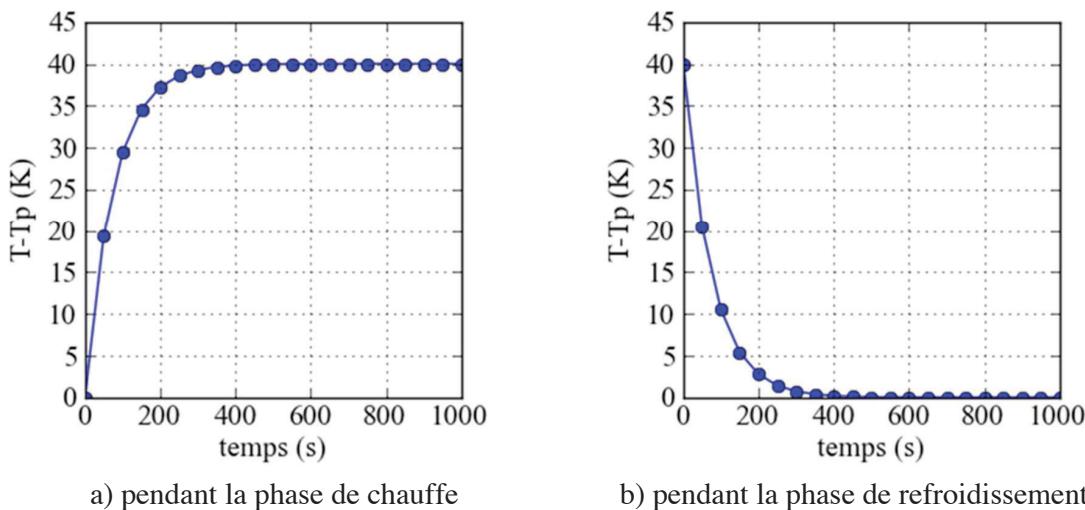


Figure 2 – Évolution de la température du liquide dans le réacteur

On exploite d'abord la courbe obtenue lors de la phase de chauffe (**figure 2a**) pour déterminer le coefficient de transfert de chaleur à la paroi, noté U (unité : $\text{W} \cdot \text{m}^{-2} \cdot \text{K}^{-1}$). Ce coefficient rend compte des échanges de chaleur entre la phase réactionnelle et le fluide caloporteur dans la double enveloppe à travers la paroi du réacteur. Dans le cas présent, la température T_p étant la température de paroi du côté du fluide caloporteur, il s'agit d'un coefficient de transfert thermique global qui tient compte du transfert convectif côté réaction et du transfert par conduction dans la paroi qui sépare les deux fluides.

Pour obtenir la valeur de U , on commence par établir l'équation différentielle qui régit l'évolution de la température T en fonction du temps en réalisant un bilan d'énergie.

Le bilan d'énergie, conséquence directe du premier principe de la thermodynamique, appliqué au système constitué par la phase réactionnelle lors de la phase de chauffe, conduit à l'équation différentielle suivante (équation (1))

$$(\rho \times V \times Cp) \frac{dT}{dt} = P_{th} - U \times A \times (T - T_p), \quad (1)$$

où T est la température du fluide à l'intérieur du réacteur, ρ , V et C_p sont respectivement la masse volumique, le volume et la capacité thermique massique du fluide, P_{th} est la puissance thermique cédée par la résistance chauffante au milieu réactionnel, T_p est la température à la paroi, maintenue constante ($T_p = 320,0$ K) et A la surface latérale du réacteur sur laquelle le fluide à l'intérieur du réacteur est en contact avec la double enveloppe.

Q1. Interpréter concrètement chacun des trois termes du bilan d'énergie en précisant leur signification physique et vérifier qu'ils sont homogènes à des puissances.

Q2. Donner l'expression de $T - T_p$ en régime permanent. Il est rappelé que la température de la paroi, T_p , est maintenue constante tout au long des essais. Il est précisé que la température de la phase liquide à l'instant initial est égale à T_p .

Q3. D'après les résultats obtenus lors de la première expérience (**figure 2a**), donner la valeur de la différence de température $T - T_p$ lorsqu'on atteint le régime permanent. Calculer la valeur du coefficient de transfert de chaleur à la paroi dans les unités SI. On donne $T_p = 320,0$ K, $\rho = 1\ 000,0\ \text{kg}\cdot\text{m}^{-3}$, $V = 0,1 \times 10^{-3}\ \text{m}^3$ et $C_p = 1\ 800,0\ \text{J}\cdot\text{kg}^{-1}\cdot\text{K}^{-1}$, $P_{th} = 96,0\ \text{W}$ et $A = 8,0 \times 10^{-3}\ \text{m}^2$.

Q4. On souhaite faire apparaître un temps caractéristique d'échange thermique τ_c du système. Montrer que le bilan d'énergie peut se mettre sous la forme suivante (équation (2)) :

$$\frac{dT}{dt} = s + \frac{T_p - T}{\tau_c}. \quad (2)$$

Donner les expressions de s et τ_c . Vérifier que τ_c est homogène à un temps.

On souhaite maintenant exploiter les résultats obtenus lors de la phase de refroidissement (**figure 2b**) pour confirmer la valeur du temps caractéristique d'échange thermique déterminé précédemment.

Q5. Le bilan d'énergie établi à la question **Q4** est-il modifié ? Si oui, donner la nouvelle expression de $\frac{dT}{dt}$.

Q6. Donner l'expression de $T - T_p$ en fonction du temps t par résolution de l'équation différentielle. On notera T_{max} la température initiale lors de la phase de refroidissement.

Q7. Le tracé de $\ln(T - T_p)$ (avec $T - T_p$ en K) en fonction du temps t (**figure 3**) donne une droite d'équation $y = -1,33 \times 10^{-2} \times x + 3,68$ (avec x en secondes). En déduire la valeur du temps caractéristique d'échange thermique τ_c . Calculer la valeur du coefficient de transfert de chaleur à la paroi et vérifier que cette valeur correspond à celle obtenue avec la première expérience.

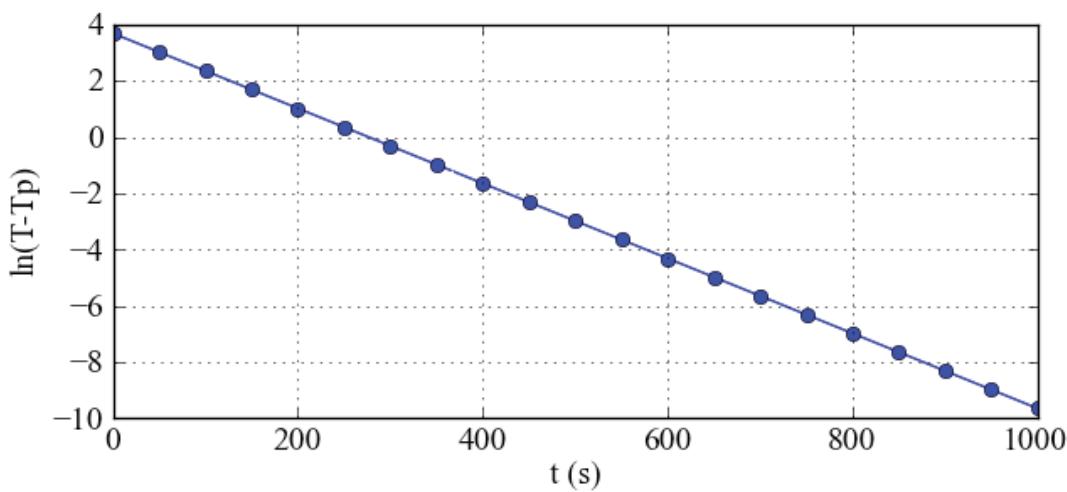


Figure 3 – Tracé de $\ln(T - T_p)$ en fonction de t à l'aide des points enregistrés lors de la phase de refroidissement (**figure 2b**)

I.2 – Etude d'une réaction exothermique en réacteur fermé à double enveloppe

Dans cette sous-partie, on considère que l'on met en œuvre une réaction chimique exothermique $R \rightarrow \text{produits}$ (R est dissous dans un solvant) dans le même réacteur que celui dont on a étudié les échanges thermiques dans la sous-partie précédente. Il s'agit ici de caractériser le comportement thermique du réacteur en présence d'une réaction exothermique.

Le comportement du réacteur fermé parfaitement agité avec double enveloppe peut être représenté par un système constitué de deux équations différentielles ordinaires. La première équation différentielle ordinaire représente l'évolution temporelle de la conversion du réactif R ; la deuxième permet de caractériser l'évolution de la température de la réaction en fonction du temps.

Le réactif R étant dissout dans un solvant, on considère que le volume du mélange réactionnel V reste constant au cours du temps. On considère également que la réaction est homogène et qu'elle a lieu dans tout le volume du mélange réactionnel.

On commence par déterminer l'équation différentielle qui régit l'évolution de la conversion du réactif R en fonction du temps.

Q8. On considère que la réaction est d'ordre 1 par rapport au réactif R . Donner l'expression de la vitesse de la réaction (exprimée par unité de volume de mélange réactionnel) que l'on notera r (on notera C_R la concentration molaire du réactif R et k la constante cinétique). Préciser la dimension et l'unité de r dans le Système International.

Q9. Rappeler la loi d'Arrhenius donnant les variations de la constante de réaction en fonction de la température. On notera k_0 le facteur pré-exponentiel et E_a l'énergie d'activation. Préciser les dimensions et les unités SI de k , k_0 et E_a .

Q10. Écrire le bilan de matière sous la forme $\frac{dC_R}{dt} = f(C_R, T)$. Préciser l'expression de $f(C_R, T)$. Il est rappelé que le volume de la phase réactionnelle reste constant au cours du temps.

Q11. Dans le cas où le réacteur fonctionnerait en marche isotherme, résoudre l'équation différentielle et donner l'expression de la concentration de R en fonction du temps sous la forme $C_R = g(t)$. On notera C_R^0 la concentration initiale en R .

Q12. Pour simplifier les bilans, on introduit le taux de conversion de R , noté X_R , défini par la relation suivante : $X_R = (C_R^0 - C_R)/C_R^0$. Exprimer l'évolution de taux de conversion X_R en fonction du temps pour le cas de la marche isotherme.

Q13. Donner l'expression de l'équation différentielle qui régit l'évolution du taux de conversion X_R en fonction du temps dans le cas général (marche quelconque, c'est-à-dire non isotherme), sans chercher à la résoudre.

L'évolution de la température en fonction du temps est régie par une seconde équation différentielle obtenue en réalisant un bilan d'énergie sur le réacteur, conséquence directe du premier principe de la thermodynamique.

La réaction chimique qui se déroule dans le réacteur produit par unité de temps une variation de l'enthalpie du système réactionnel donnée par $S(t, X, T)$ (équation (3)) qui correspond à la chaleur dégagée par la réaction. Ce paramètre est appelé « terme source » dans la suite.

$$S(t, X_R, T) = -\Delta_r H^0(T) \times r(t, X_R, T) \times V, \quad (3)$$

où V est le volume du mélange réactionnel, r est la vitesse de la réaction et $\Delta_r H^0(T)$ est l'enthalpie molaire standard de la réaction. Dans la suite, on suppose que $\Delta_r H^0(T)$ ne dépend pas de la température. On prendra $\Delta_r H^0(T) = \Delta_r H^0(T_p)$ que l'on notera $\Delta_r H^0$ pour simplifier.

Q14. Donner la dimension du terme source $S(t, X_R, T)$.

Q15. Montrer qu'un bilan enthalpique appliqué à un système que l'on précisera avec soin permet d'aboutir à la relation suivante (équation (4))

$$\frac{dT}{dt} = J \frac{dX_R}{dt} - \frac{T - T_p}{\tau_c}, \quad (4)$$

où l'on exprimera J et τ_c en fonction de $\Delta_r H$, C_R^0 , ρ , C_p , V , U et A . On admettra qu'il est légitime de négliger la contribution des réactifs, des produits et des accessoires situés à l'intérieur du réacteur au travers de leur capacités thermiques devant celle du solvant.

Q16. Donner l'expression du paramètre J ainsi que sa dimension.

Q17. Calculer la valeur du paramètre J en unité SI. On donne $\Delta_r H^0 = -360,0 \text{ kJ}\cdot\text{mol}^{-1}$, $\rho = 1\,000,0 \text{ kg}\cdot\text{m}^{-3}$, $C_p = 1\,800,0 \text{ J}\cdot\text{kg}^{-1}\cdot\text{K}^{-1}$ et $C_R^0 = 500,0 \text{ mol}\cdot\text{m}^{-3}$.

I.3 – Stabilité thermique du réacteur

Une première condition de stabilité, valable pour le cas d'une marche adiabatique, est que la température finale T_f de la réaction soit inférieure à une température maximale T_{max} , telle que $T_{max} = 1,25 \times T_p$.

Q18. Exprimer l'équation différentielle (équation (4)) dans le cas d'un fonctionnement adiabatique.

Q19. Déterminer alors l'expression de la température T en fonction du taux de conversion X_R , du paramètre J et de T_0 la température initiale du mélange réactionnel.

Q20. Donner l'expression de la température T_f atteinte en fin de réaction dans le cas d'une marche adiabatique sachant que $T_0 = T_p = 320,0 \text{ K}$. Conclure quant à la stabilité du réacteur dans le cas de cette étude. Donner la signification physique du paramètre J .

Partie II – Traitement numérique des données expérimentales

Les algorithmes demandés au candidat **devront être réalisés dans le langage Python**. On supposera les bibliothèques « numpy » et « matplotlib.pyplot » chargées. Une **annexe** présentant les fonctions usuelles de Python est disponible pages 15 à 18. Les commentaires suffisants à la compréhension du programme devront être apportés et des noms de variables explicites devront être utilisés lorsque ceux-ci ne sont pas imposés.

II.1 – Détermination des paramètres d'un modèle par régression linéaire

Pour calculer la valeur du temps caractéristique d'échange thermique du réacteur à la question **Q7**, un modèle de régression linéaire simple a été estimé à partir des points expérimentaux enregistrés lors de la phase de refroidissement.

Le modèle de régression linéaire simple (fonction affine) est un modèle de régression d'une variable expliquée ($\ln(T - T_p)$ dans notre cas) sur une variable explicative (le temps t dans notre cas) dans lequel on fait l'hypothèse que la fonction qui relie la variable explicative à la variable expliquée est linéaire dans ses paramètres.

Soit n le nombre de points expérimentaux. Le modèle linéaire simple s'écrit de la manière suivante pour un point i ($1 \leq i \leq n$)

$$y_i = \widehat{\beta}_1 \times x_i + \widehat{\beta}_0, \quad (5)$$

où $\widehat{\beta}_0$ et $\widehat{\beta}_1$ sont les paramètres du modèle, y_i est la variable expliquée et x_i est la variable explicative.

On propose de déterminer les paramètres du modèle par deux méthodes directes.

La méthode consiste à écrire le modèle (équation (5)) sous la forme matricielle $Y = L \times \widehat{B}$. \widehat{B} est un vecteur colonne contenant les paramètres du modèle $\widehat{\beta}_0$ et $\widehat{\beta}_1$, Y est un vecteur colonne contenant les n valeurs y_i et L une matrice à n lignes et 2 colonnes, telle que $L(i, 1) = \frac{\partial y_i}{\partial \widehat{\beta}_0}$ et $L(i, 2) = \frac{\partial y_i}{\partial \widehat{\beta}_1}$. Rappelons que $\widehat{B} = (L^t \times L)^{-1} \times L^t Y$ où L^t est la matrice transposée de L .

Q21. Donner les expressions de $\frac{\partial y_i}{\partial \widehat{\beta}_0}$ et $\frac{\partial y_i}{\partial \widehat{\beta}_1}$. En déduire la valeur des coefficients de la matrice L .

Q22. On suppose que les vecteurs colonnes Y et X , qui contiennent les valeurs y_i et x_i ($1 \leq i \leq n$), sont déjà créés. Donner le code permettant de créer la matrice L .

Q23. Donner le code permettant de déterminer les coefficients de la matrice $P = L^t \times L$. Préciser les dimensions de la matrice P .

Q24. Donner le code permettant de déterminer les coefficients de la matrice $Q = L^t \times Y$. Préciser les dimensions de la matrice Q .

Q25. Donner le code permettant de créer une fonction **inv_mat(M)** qui renvoie la matrice inverse de la matrice M de dimension (2×2) donnée comme argument d'entrée.

Q26. On note N la matrice inverse de M . Donner le code permettant de déterminer les coefficients $\widehat{\beta}_0$ et $\widehat{\beta}_1$ de la matrice \widehat{B} .

II.2 – Prédiction du comportement thermique du réacteur

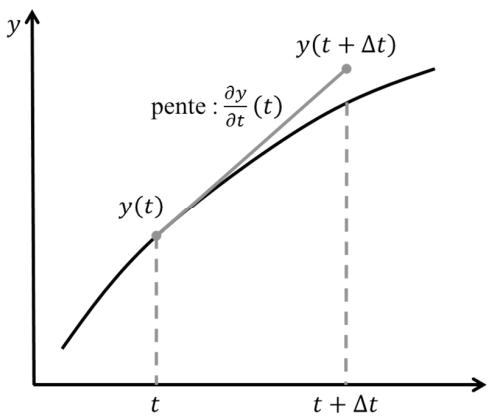
Dans cette sous-partie, on souhaite utiliser le modèle constitué du système d'équations différentielles établies dans la **Partie I** qui décrit l'évolution du taux de conversion du réactif X_R et de la température de réaction T en fonction du temps pour prédire le comportement thermique du réacteur en présence d'une réaction exothermique.

Pour simplifier les notations, on met le système d'équations différentielles sous la forme suivante (équation (6)) :

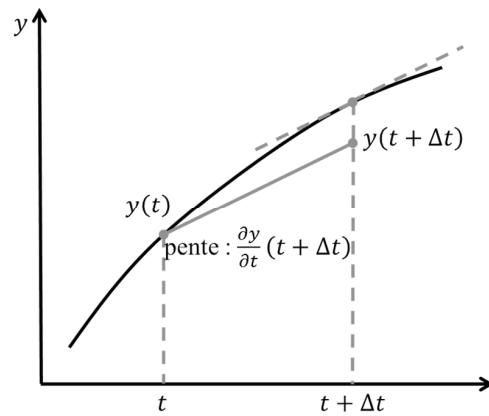
$$\begin{cases} \frac{dX_R}{dt} = f_1(t, X_R, T), \\ \frac{dT}{dt} = f_2(t, X_R, T). \end{cases} \quad (6)$$

La méthode de résolution proposée pour résoudre le système d'équations différentielles est la méthode d'Euler implicite à pas fixe. Cette méthode est préférée car elle donne de meilleurs résultats que la méthode explicite pour les systèmes dits raides (un système raide est un système qui est caractérisé par une évolution rapide des phénomènes en fonction du temps, ce qui est le cas ici pour la température de réaction).

Soit une variable y qui dépend du temps t . Comme la méthode d'Euler explicite, la méthode d'Euler implicite consiste à évaluer la valeur de $y(t + \Delta t)$ à partir de celle de $y(t)$ et de la dérivée $\frac{\partial y}{\partial t}$ (**figure 4**, page suivante). La différence entre les deux méthodes réside dans le choix de l'abscisse à laquelle est évaluée la dérivée $\frac{\partial y}{\partial t}$. Dans le cas de la méthode explicite, elle est évaluée en $t : \frac{\partial y}{\partial t}(t)$ comme le montre le schéma de la **figure 4a**, page suivante. Pour la méthode implicite, elle est évaluée en $t + \Delta t : \frac{\partial y}{\partial t}(t + \Delta t)$ (**figure 4b**, page suivante).



a) cas de la méthode explicite



b) cas de la méthode implicite

Figure 4 – Approximation de $y(t + \Delta t)$ par la méthode d'Euler

Dans le cas d'un schéma implicite (**figure 4b**), l'expression de $y(t + \Delta t)$ en fonction de $y(t)$ et de la dérivée $\frac{\partial y}{\partial t}$ ($t + \Delta t$) évaluée en $t + \Delta t$ est obtenue en réalisant un développement limité dit rétrograde : $y(t) = y(t + \Delta t) - \Delta t \times \frac{\partial y}{\partial t}(t + \Delta t) + o(\Delta t)$.

Q27. À l'aide d'un développement limité rétrograde de la fonction $X_R(t)$, donner l'expression de $X_R(t + \Delta t)$ à l'ordre 1 en fonction de $X_R(t)$ et de sa dérivée partielle par rapport à t , $\frac{dX_R}{dt}(t + \Delta t)$ évaluée en $t + \Delta t$.

Q28. En déduire une valeur approchée de $\frac{dX_R}{dt}(t + \Delta t)$ à l'ordre 0 en fonction de $X_R(t)$, $X_R(t + \Delta t)$ et Δt .

Q29. À l'aide d'un développement limité rétrograde de la fonction $T(t)$, donner l'expression de $T(t + \Delta t)$ à l'ordre 1 en fonction de $T(t)$ et de sa dérivée partielle par rapport à t , $\frac{dT}{dt}(t + \Delta t)$.

Q30. En déduire une valeur approchée de $\frac{dT}{dt}(t + \Delta t)$ à l'ordre 0 en fonction de $T(t)$, $T(t + \Delta t)$ et Δt .

On procède à la discrétisation des équations. On note X_{Ri} la conversion évaluée au temps t_i , $X_{R,i+1}$ la conversion évaluée au temps t_{i+1} et $\left. \frac{dX_R}{dt} \right|_{t_{i+1}}$ la dérivée de X_R évaluée à l'instant t_{i+1} . De même, on note T_i la température évaluée au temps t_i , T_{i+1} la température évaluée au temps t_{i+1} et $\left. \frac{dT}{dt} \right|_{t_{i+1}}$ la dérivée de T évaluée à l'instant t_{i+1} .

Q31. Donner l'expression de $\frac{dX_R}{dt}\Big|_{t_{i+1}}$ en fonction de X_{Ri} , X_{Ri+1} et Δt .

Q32. Donner l'expression approchée de X_{Ri+1} en fonction de X_{Ri} , Δt et de la fonction $f_1(t_{i+1}, X_{Ri+1}, T_{i+1})$, évaluée en t_{i+1} .

Q33. Donner l'expression de $\frac{dT}{dt}\Big|_{t_{i+1}}$ en fonction de T_i , T_{i+1} et Δt .

Q34. Donner l'expression approchée de T_{i+1} en fonction de T_i , Δt et de la fonction $f_2(t_{i+1}, X_{Ri+1}, T_{i+1})$, évaluée en t_{i+1} .

On constate que les expressions obtenues aux questions **Q32** et **Q34** constituent un système non linéaire dont les inconnues sont X_{Ri+1} et T_{i+1} . On propose d'utiliser la méthode de Newton-Raphson pour trouver les valeurs de X_{Ri+1} et T_{i+1} à chaque itération de la méthode d'Euler.

La méthode de Newton-Raphson pour la résolution d'un système de n équations non linéaires à n inconnues $x = (x_1, \dots, x_n)$, mis sous la forme de l'équation (7) suivante,

$$g(x) = \begin{bmatrix} g_1(x_1, \dots, x_n) \\ \vdots \\ g_n(x_1, \dots, x_n) \end{bmatrix} = 0, \quad (7)$$

est une extension de la méthode de Newton permettant de trouver la racine d'une fonction d'une variable.

On peut démontrer la formule de récurrence suivante (équation (8)) :

$$x^{j+1} = x^j - (Dg(x^j))^{-1} g(x^j), \quad (8)$$

où x^{j+1} est la valeur du vecteur x à l'itération $j + 1$, x^j est la valeur du vecteur x à l'itération j , $g(x^j)$ est la valeur de $g(x)$ à l'itération j et $Dg(x^j)$ est la matrice Jacobienne évaluée en x^j (équation (9)) :

$$Dg(x^j) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix}_{x=x^j}. \quad (9)$$

La formule de récurrence s'accompagne du choix d'une valeur initiale, notée x^0 , et d'un critère d'arrêt, par exemple $\|x^{j+1} - x^j\| \leq \varepsilon$.

Q35. Transformer les expressions obtenues aux questions **Q32** et **Q34** pour les mettre sous la forme $g_1(X_{Ri+1}, T_{i+1}) = 0$ et $g_2(X_{Ri+1}, T_{i+1}) = 0$.

Q36. Donner les expressions de $\frac{\partial g_1}{\partial X_{Ri+1}}$, $\frac{\partial g_1}{\partial T_{i+1}}$, $\frac{\partial g_2}{\partial X_{Ri+1}}$ et $\frac{\partial g_2}{\partial T_{i+1}}$ permettant de construire la matrice Jacobienne $Dg(X_{Ri+1}, T_{i+1})$.

Q37. Écrire une fonction **mat_Dg(x)** qui a pour argument d'entrée un vecteur x contenant les valeurs de X_{Ri+1} et T_{i+1} à l'itération j et qui retourne la matrice Jacobienne $Dg(x^j)$. On supposera que les paramètres suivants ont été au préalable déclarés comme variables globales : $\Delta t = 0,01$ s, $k_0 = 5,0$ s $^{-1}$, $E_a = 20\,000,0$ J.mol $^{-1}$, $J = 100,0$ K, $\tau_c = 75$ s et $R = 8,314$ J.K $^{-1}.\text{mol}^{-1}$.

Q38. Écrire le code permettant de calculer les valeurs du vecteur x à l'itération $j + 1$ à l'aide de l'équation (8) lors d'une boucle de l'algorithme de Newton-Raphson. On notera **x_old** la valeur de x à l'itération j et **x_new** la valeur de x à l'itération $j + 1$. De même, on notera **x_euleri** et **T_euleri** les vecteurs contenant les valeurs de X_{Ri} et T_i à l'itération i de la méthode d'Euler implicite. Pour l'inversion de matrice, on utilisera la fonction **inv_mat(M)** écrite à la question **Q25**.

Q39. Pour obtenir la valeur de **x_new** par la méthode de Newton-Raphson, on souhaite créer une boucle itérative avec condition. La condition d'arrêt porte sur la valeur absolue de la différence des températures T_{i+1}^j et T_{i+1}^{j+1} que l'on souhaite inférieure à 10^{-5} K. Pour les valeurs initiales de X_{ri+1}^0 et T_{i+1}^0 on prendra respectivement 0,5 et $T_p + J/2$. Écrire le code correspondant. Il est inutile de recopier l'intégralité du code écrit à la question précédente ; on indiquera néanmoins sa place dans le code de cette question.

Maintenant que le code permettant de trouver les valeurs de X_{Ri+1} et T_{i+1} par la méthode de Newton-Raphson lors d'une itération de la méthode d'Euler implicite a été établi, on souhaite calculer les valeurs pour l'ensemble des itérations de la méthode d'Euler implicite. On rappelle que $X_R(t = 0) = 0$ et $T(t = 0) = T_p = 320,0$ K. En plus de noter **x_euleri** et **T_euleri** les vecteurs contenant les valeurs de X_{Ri} et T_i pour chaque itération i de la méthode d'Euler implicite, on notera **t_euleri** le vecteur contenant les valeurs de t_i à chaque itération. L'intégration sera réalisée sur l'intervalle $[t_0, t_f]$ avec $t_0 = 0$ s et $t_f = 1\,000$ s. Le pas de temps Δt est de 0,01 s.

Q40. Donner le code permettant de calculer le nombre m d'intervalles Δt compris dans l'intervalle $[t_0, t_f]$ (m est un entier).

Q41. Écrire le code permettant de calculer les valeurs des éléments des vecteurs **x_euleri**, **T_euleri** et **t_euleri** à chaque itération de la méthode d'Euler implicite.

Q42. Donner le code permettant de tracer les graphes de la **figure 5** montrant l'évolution de la conversion et de la température en fonction du temps que l'on obtiendrait en réalisant la résolution numérique du système d'équations différentielles (simulation réalisée avec $k_0 = 5,0 \text{ s}^{-1}$, $E_a = 20\,000,0 \text{ J}\cdot\text{mol}^{-1}$, $J = 100,0 \text{ K}$, $\tau_c = 75 \text{ s}$).

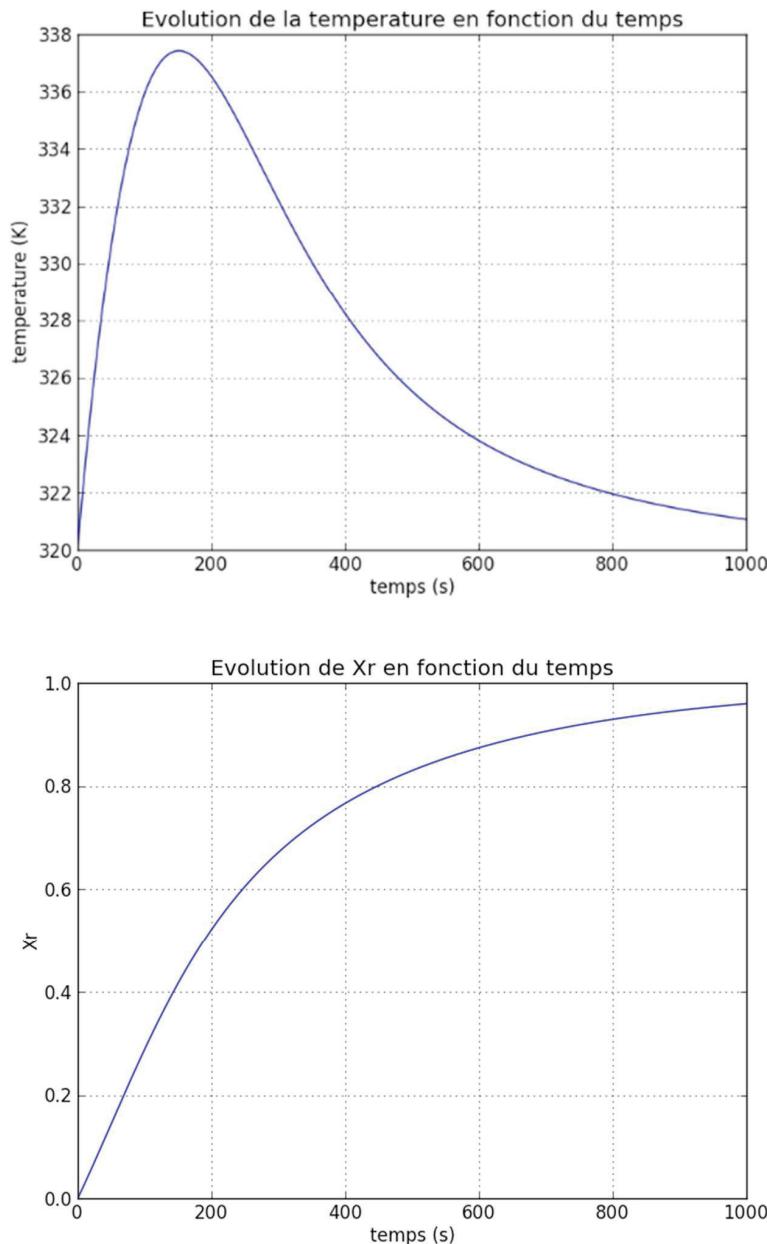


Figure 5 – Température et conversion calculées à partir du modèle constitué du système d'équations différentielles déterminées dans la **Partie I**

FIN

Annexe : Fonctions de Python

1. Bibliothèque numpy de Python

Dans les exemples ci-dessous, la bibliothèque numpy a préalablement été importée à l'aide de la commande : **import numpy as np**

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

np.array(liste)

Description : fonction permettant de créer une matrice (de type tableau) à partir d'une liste.

Argument d'entrée : une liste définissant un tableau à 1 dimension (vecteur) ou 2 dimensions (matrice).

Argument de sortie : un tableau (matrice).

Exemples : `np.array([4,3,2])`
 ⇒ [4 3 2]

`np.array([[5],[7],[1]])`
 ⇒ [[5]
 [7]
 [1]]]

`np.array([[3,4,10],[1,8,7]])`
 ⇒ [[3 4 10]
 [1 8 7]]]

$A[i,j]$.

Description : fonction qui retourne l'élément $(i + 1, j + 1)$ de la matrice A . Pour accéder à l'intégralité de la ligne $i+1$ de la matrice A , on écrit $A[i,:]$. De même, pour obtenir toute la colonne $j+1$ de la matrice A , on utilise la syntaxe $A[:,j]$.

Arguments d'entrée : une liste contenant les coordonnées de l'élément dans le tableau A .

Argument de sortie : l'élément $(i + 1, j + 1)$ de la matrice A .

ATTENTION : en langage Python, les lignes d'un tableau A de dimension $n \times m$ sont numérotées de 0 à $n - 1$ et les colonnes sont numérotées de 0 à $m - 1$

Exemple : `A=np.array([[3,4,10],[1,8,7]])`

`A[0,2]`
 ⇒ 10

`A[1,:]`
 ⇒ [1 8 7]

`A[:,2]`
 ⇒ [10 7]

np.zeros((n,m))

Description : fonction créant une matrice (tableau) de dimensions $n \times m$ dont tous les éléments sont nuls.

Arguments d'entrée : un tuple de deux entiers correspondant aux dimensions de la matrice à créer.

Argument de sortie : un tableau (matrice) d'éléments nuls.

Exemple : np.zeros((3,4))

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

np.linspace(Min,Max,nbElements)

Description : fonction créant un vecteur (tableau) de $nbElements$ nombres espacés régulièrement entre Min et Max . Le 1^{er} élément est égal à Min , le dernier est égal à Max et les éléments sont espacés de $(Max - Min)/(nbElements - 1)$:

Arguments d'entrée : un tuple de 3 entiers.

Argument de sortie : un tableau (vecteur).

Exemple : np.linspace(3,25,5)

$$\Rightarrow [3 \ 8.5 \ 14 \ 19.5 \ 25]$$

np.loadtxt('nom_fichier',delimiter='string',usecols=[n])

Description : fonction permettant de lire les données sous forme de matrice dans un fichier texte et de les stocker sous forme de vecteurs.

Arguments d'entrée : le nom du fichier qui contient les données à charger, le type de caractère utilisé dans ce fichier pour séparer les données (par exemple un espace ou une virgule) et le numéro de la colonne à charger (ATTENTION, la première colonne porte le numéro 0).

Argument de sortie : un tableau.

Exemple : data=np.loadtxt('fichier.txt',delimiter=' ',usecols=[0])
#dans cet exemple data est un vecteur qui correspond à la première #colonne de la matrice contenue dans le fichier 'fichier.txt'.

2. Bibliothèque matplotlib.pyplot de Python

Cette bibliothèque permet de tracer des graphiques. Dans les exemples ci-dessous, la bibliothèque matplotlib.pyplot a préalablement été importée à l'aide de la commande :

import matplotlib.pyplot as plt

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

plt.plot(x,y)

Description : fonction permettant de tracer un graphique de n points dont les abscisses sont contenues dans le vecteur x et les ordonnées dans le vecteur y . Cette fonction doit être suivie de la fonction **plt.show()** pour que le graphique soit affiché.

Arguments d'entrée : un vecteur d'abscisses x (tableau de dimension n) et un vecteur d'ordonnées y (tableau de dimension n).

Argument de sortie : un graphique.

Exemple :

```
x= np.linspace(3,25,5)
y=sin(x)
plt.plot(x,y)
plt.title('titre_graphique')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

plt.title('titre')

Description : fonction permettant d'afficher le titre d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.xlabel('nom')

Description : fonction permettant d'afficher le contenu de nom en abscisse d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.ylabel('nom')

Description : fonction permettant d'afficher le contenu de nom en ordonnée d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.show()

Description : fonction réalisant l'affichage d'un graphe préalablement créé par la commande **plt.plot(x,y)**. Elle doit être appelée après la fonction **plt.plot** et après les fonctions **plt.xlabel** et **plt.ylabel**.

3. Fonction intrinsèque de Python

sum(x)

Description : fonction permettant de faire la somme des éléments d'un vecteur ou tableau.

Arguments d'entrée : un vecteur ou un tableau de réel, entier ou complexe.

Argument de sortie : un scalaire y qui est la somme des éléments de x.

Exemple : $y = \text{sum}(x)$
 //y retourne la somme des éléments de x.



ÉPREUVE SPÉCIFIQUE - FILIÈRE PSI

INFORMATIQUE

Vendredi 4 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 4 parties, toutes 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 16.

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**. 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 15

Annexe : page 16

SYSTÈME D'AIDE L'ARBITRAGE : LE HAWK-EYE

Partie I - Présentation générale du système

Le système Hawk-eye est un système d'aide à la décision arbitrale utilisé dans le sport de haut niveau, notamment le tennis, un des plus connus. Il a été développé par le Paul Hawkins en 1999 avec pour objectif d'augmenter la qualité des décisions arbitrales et fournir un résultat rapide, clair et précis lors des moments décisifs de rencontres sportives (**figure 1**). Il a été utilisé pour la première fois au tennis lors du Masters de Miami en 2006. Chaque joueur peut faire appel trois fois par set de cette décision.

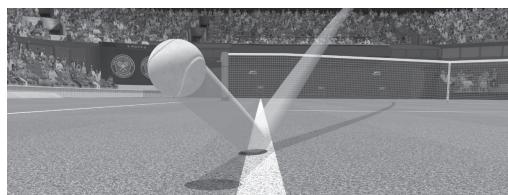


Figure 1 – Décision officielle du système Hawk-eye

Pour analyser la trajectoire et la position de la balle, le système Hawk-Eye se compose de dix caméras, réparties à égale distance autour du court de tennis dans les tribunes (**figure 2**). Ces caméras peuvent photographier "en rafale" des objets se déplaçant à une grande vitesse grâce à un capteur photographique. Elles peuvent enregistrer jusqu'à 1 000 images par seconde. Ces images sont ensuite envoyées au poste de commande du système.

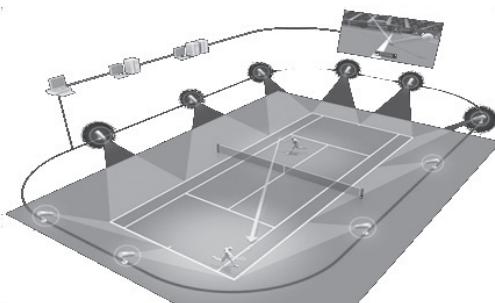


Figure 2 – Position des caméras haute vitesse autour du terrain de tennis

L'objectif de l'étude proposée est de réaliser le programme de suivi (tracking) de la trajectoire de la balle de tennis par le système Hawk-eye, la reconstruction de la trajectoire et enfin l'identification de la position de l'impact de la balle avec le sol pour savoir si la balle est dans les limites du terrain ou non. Afin de mieux appréhender le problème, nous commencerons par la modélisation et l'étude théorique de la trajectoire d'une balle de tennis et montrerons en quoi cette seule modélisation est insuffisante pour l'aide à l'arbitrage.

Dans tout le sujet, il sera supposé que les bibliothèques sont déjà importées dans le programme. Attention, l'utilisation des fonctions min et max ne sera pas acceptée.

Partie II - Modélisation de la trajectoire de la balle

L'objectif de cette partie est de déterminer la trajectoire théorique d'une balle de tennis et de montrer les limites du résultat obtenu.

La trajectoire d'une balle de tennis dépend de plusieurs paramètres :

- la vitesse de frappe par la raquette du joueur ;
- l'angle de frappe ;
- les frottements dans l'air ;
- la vitesse de rotation donnée à la balle par la raquette du joueur.

Le schéma de la **figure 3** représente un terrain de tennis pour une partie en simple et la trajectoire de la balle dans l'espace de coordonnées (x, y, z) . Le repère est choisi de sorte que le plan $(0, \vec{x}, \vec{z})$ soit horizontal.

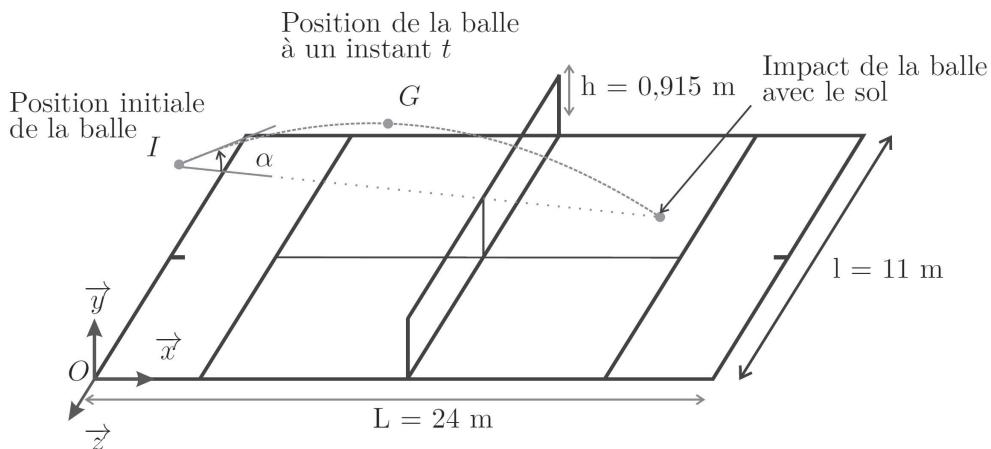


Figure 3 – Paramétrage du terrain de tennis et trajectoire d'une balle

Paramétrage

- La balle a une masse m et un rayon R .
- Le mouvement de la balle est étudié dans le référentiel \mathcal{R} lié au terrain et supposé galiléen. On lui associe le repère $(O, \vec{x}, \vec{y}, \vec{z})$, avec O le point de coordonnées $(0, 0, 0)$.
- La position initiale de la balle est définie dans le référentiel supposé galiléen lié au terrain au point I par (x_0, y_0, z_0) .
- La balle est repérée par la position de son centre d'inertie G auquel est associé le vecteur $\overrightarrow{OG} = (x + x_0)\vec{x} + (y + y_0)\vec{y} + (z + z_0)\vec{z}$.
- La vitesse de la balle est notée $\vec{V} = v_x\vec{x} + v_y\vec{y} + v_z\vec{z}$ et sa vitesse initiale \vec{V}_0 .
- L'angle de frappe entre le plan (O, \vec{x}, \vec{z}) et \vec{V}_0 est α .
- La vitesse de rotation sur elle-même de la balle est notée $\vec{\Omega} = \Omega\vec{z}$.
- $\vec{g} = -g\vec{y}$ est l'accélération de la pesanteur.

Hypothèses

- On supposera **dans toute la suite de cette partie** que la balle ne se déplace que dans le plan (O, \vec{x}, \vec{y}) . Par conséquent, à chaque instant, $v_z = 0$.
- Les différents efforts s'exerçant sur la balle de tennis sont représentés sur la **figure 4**, page 4. \vec{t} est un vecteur unitaire tangent à la trajectoire et dirigé suivant le sens de la trajectoire. \vec{n} est un vecteur unitaire normal à la trajectoire.

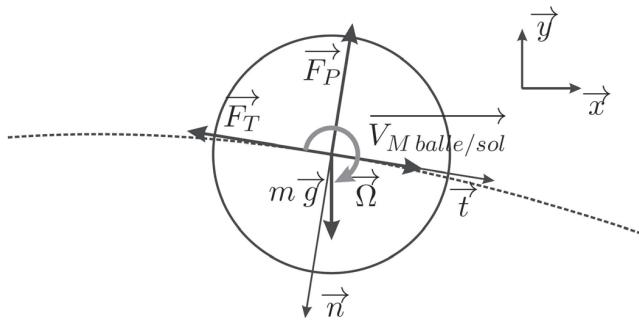


Figure 4 – Efforts de l'air sur la balle

Équation du mouvement de la balle

Le mouvement de la balle de tennis est défini par l'équation suivante

$$m \frac{d^2 \overrightarrow{OG}}{dt^2} = m \cdot \vec{g} + \vec{F}_T + \vec{F}_P \quad (1)$$

avec $\vec{F}_T = -\frac{1}{2} \cdot \pi \cdot R^2 \cdot \rho_{air} \cdot \vec{V}_1 \cdot V^2 \cdot \vec{t}$ et $\vec{F}_P = \rho_{air} \cdot R^3 \cdot \vec{V}_2 \cdot V \cdot \vec{\Omega} \cdot \vec{n}$ où ρ_{air} est la masse volumique de l'air (environ 1 kg/m^3 à température ambiante), V la vitesse de la balle et \vec{V}_1 et \vec{V}_2 deux coefficients sans dimension qui dépendent du nombre de Reynolds.

L'équation (1) une fois projetée dans le plan (O, \vec{x}, \vec{y}) devient :

$$m \frac{d^2 x(t)}{dt^2} = -\frac{1}{2} \cdot \pi \cdot R^2 \cdot \rho_{air} \cdot \vec{V}_1 \cdot V^2 \cdot \cos(\theta) - \rho_{air} \cdot R^3 \cdot \vec{V}_2 \cdot V \cdot \vec{\Omega} \cdot \sin(\theta) \quad (2)$$

$$m \frac{d^2 y(t)}{dt^2} = -m \cdot g - \frac{1}{2} \cdot \pi \cdot R^2 \cdot \rho_{air} \cdot \vec{V}_1 \cdot V^2 \cdot \sin(\theta) + \rho_{air} \cdot R^3 \cdot \vec{V}_2 \cdot V \cdot \vec{\Omega} \cdot \cos(\theta). \quad (3)$$

On pose $Y = \begin{bmatrix} u \\ v \\ x \\ y \end{bmatrix}$, avec $u(t) = \frac{dx(t)}{dt}$ et $v(t) = \frac{dy(t)}{dt}$ et dont la condition initiale est $Y = \begin{bmatrix} u_0 \\ v_0 \\ x_0 \\ y_0 \end{bmatrix}$.

Q1. Mettre les équations (2) et (3) sous la forme d'un problème de Cauchy du type : $\frac{dY}{dt} = F(Y, t)$.

La résolution numérique des équations différentielles (2) et (3) repose sur leur discrétisation temporelle et conduit à déterminer à différents instants t_i une approximation de la solution.

On note u_i et v_i les approximations des composantes du vecteur vitesse $(u(t), v(t))$ et x_i et y_i les approximations des composantes du vecteur position $(x(t), y(t))$ à l'instant t_i . On note $\Delta t = t_{i+1} - t_i$, le pas de temps constant.

Le schéma d'Euler conduit à la relation de récurrence suivante :

$$Y_{i+1} = Y_i + \Delta t F(t_i, Y_i).$$

Q2. Compléter sur votre copie la fonction `euler(T,N,F,Y0)` ci-dessous permettant de construire le schéma d'Euler.

```
def euler (T,N,F,Y0) :
    Y=zeros ((len(Y0),N))
    t=arange (0,T,T/N)
    #conditions initiales à compléter
    for i in range(1,N):
        #zone à compléter
    return (t,Y)
```

La reconstruction de la trajectoire d'une balle de tennis à partir des lois physiques et des conditions initiales ne se révèle pas satisfaisante en terme de précision pour l'aide à la décision arbitrale. En effet, les paramètres climatiques (vent, pluie, pression atmosphérique...), de déformation de la balle ou tout autre aléa ne sont pas pris en compte *a posteriori*. Il convient donc d'adopter une autre méthode vérifiant à tout instant la position de la balle ; c'est ce que permet le système Hawk-eye. Nous allons nous intéresser dans la suite à une partie de l'algorithme permettant de vérifier si une balle est bonne et de reconstruire la trajectoire de la balle.

Partie III - Tracking et reconstruction de la trajectoire de la balle de tennis

Comme nous l'avons vu dans la partie présentation et plus particulièrement sur la **figure 2**, page 2, le système Hawk-eye est composé de 10 caméras réparties autour du terrain de tennis.

Pour pouvoir obtenir la trajectoire de la balle et déterminer le point d'impact de cette dernière avec le terrain, il est nécessaire de faire appel à une unité de traitement des données de différentes caméras. Cette unité de traitement des données fait appel à un algorithme de calcul qui se décompose en 7 étapes :

- Étape ① : on détermine les limites du terrain ;
- Étape ② : on réalise le calibrage de la caméra (détermination de sa position, son orientation et ses paramètres intrinsèques) ;
- Étape ③ : on identifie les pixels représentant la balle et on calcule la position 2D dans chaque image de chaque caméra ;
- Étape ④ : on calcule la position 3D de la balle par triangulation ;
- Étape ⑤ : on assemble les images obtenues ;
- Étape ⑥ : on détermine l'impact de la balle avec le sol sur le terrain ;
- Étape ⑦ : on reconstruit la trajectoire 3D de la balle que l'on affichera pour les spectateurs.

Nous nous proposons dans la suite du sujet de nous intéresser plus spécifiquement aux étapes ③, ④, ⑤, ⑥ et ⑦.

III.1 - Détermination de la taille des fichiers récupérés

L'image acquise par la caméra est une image en couleur constituée de trois couches de pixels (Red - Green - Blue). La résolution de l'écran est de 1 024 pixels horizontalement et 768 verticalement. Les données de l'image sont stockées dans un tableau de type `list` à trois dimensions : la première dimension correspond à la coordonnée selon *x*, la seconde à la coordonnée selon *y* et la troisième correspond à la couleur (rouge, vert ou bleu, indice 0, 1 ou 2).

Ainsi, les dimensions du tableau sont : $n \times m \times 3$ où $n = 1\,024$ et $m = 768$. La valeur associée à chaque pixel est un entier compris entre 0 et 255.

Un point joué désigne l'ensemble des coups exécutés par les joueurs, service compris, jusqu'à ce que l'un d'entre eux commette une faute ou un coup gagnant. Un échange désigne un couple de coups joués par les joueurs.

Q3. On suppose que l'on stocke dans la variable `image1` une image enregistrée par la caméra.

Indiquer la valeur des expressions suivantes :

- `len(image1)`
- `len(image1[12][244])`

Indiquer le type de l'expression suivante :

- `len(image1[12][244][2])`

Q4. Déterminer, en justifiant succinctement votre calcul, la taille de l'espace mémoire occupé par le tableau représentant l'image émise par la caméra.

Q5. Estimer un ordre de grandeur de l'espace de stockage à prévoir pour sauvegarder toutes les images des 10 caméras rapides (1 000 images/s) correspondant à un point joué, puis à un set. Un set comporte en moyenne 100 points joués. On pourra estimer à 10 s la durée moyenne d'un point joué. Indiquer quel moyen de stockage pourra convenir à la sauvegarde des données.

Q6. Définir ce qu'on appelle la mémoire vive d'un ordinateur. Indiquer les inconvénients liés à l'utilisation des données décrites à la question précédente.

III.2 - Calcul de la position 2D de la balle (étape ③)

L'objectif de l'algorithme que nous allons étudier ici est d'identifier la position de la balle en prenant en compte sa taille et sa forme sur chaque image enregistrée à l'aide d'une caméra rapide.

L'algorithme doit renvoyer les coordonnées x et y de la balle dans le repère $(O, \vec{x}, \vec{y}, \vec{z})$ et l'instant correspondant t lorsque celle-ci est détectée. Si la balle n'est pas détectée dans le champ de vision de la caméra, l'information renournée sera `None`.

L'algorithme proposé se déroule en plusieurs étapes :

- première étape : détecter tous les objets en mouvement grâce à l'analyse de deux images successives,
- deuxième étape : éliminer les objets en mouvement qui ne peuvent pas correspondre à une balle de tennis selon des critères géométriques,
- troisième étape : éliminer les "balles fausses" détectées en prenant en compte la trajectoire de la balle.

La première étape est réalisée en comparant deux images successives (donc séparées d'un laps de temps très bref). Ces 2 images (`image1` et `image2`) sont quasiment identiques exceptées les zones en mouvement. L'objectif est de créer un nouveau tableau de dimension $n \times m$ (avec $n = 1\,024$ et $m = 768$) dans lequel l'élément correspondant à un pixel est un entier compris entre 0 et 255. Cet entier correspond à un niveau de gris (le noir correspondant à l'entier 0 et le blanc à l'entier 255). Les zones blanches ou claires seront considérées comme des objets en mouvement.

Les images sont celles décrites en début de sous-partie III.2 et correspondent à des tableaux à 3 dimensions de type `list : n × m × 3`. Pour détecter les zones en mouvement, l'algorithme propose de

calculer pour chaque pixel la somme des différences au carré associées à chaque couleur

$$= (r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 \quad (4)$$

où r_i , g_i et b_i représentent respectivement les niveaux de rouge, vert et bleu d'un pixel de coordonnées (x, y) de l'image1 et de l'image2.

En cas de variation importante de couleur, peut atteindre une valeur supérieure à 255 ce qui pose des difficultés dans la suite de l'algorithme. On veut garantir $0 \leq \leq 255$ et on impose donc la valeur 255 dans le cas où la formule précédente fournirait une valeur supérieure.

- Q7.** Écrire une fonction `detection(image1,image2)` qui prend en argument 2 images décrites à la sous-partie **III.2** (page 5) et qui renvoie un tableau de dimension $n \times m$ d'entiers compris entre 0 et 255 dans lequel chaque élément correspond à . Écrire l'instruction qui permet d'affecter à la variable `image_gray` le résultat de cette fonction appliquée à deux images `im1` et `im2`.

Le tableau ainsi obtenu est ensuite filtré pour éliminer les pixels trop clairs qui correspondent à des parties figées de l'image. Le seuil doit pouvoir être ajusté pour chaque caméra suivant la luminosité et les conditions météorologiques.

- Q8.** Écrire une fonction `filtre(image,seuil)` qui prend en argument un tableau à deux dimensions (de type `list` de `list`) et qui renvoie un nouveau tableau de même dimension que celui passé en argument. Chaque élément de ce nouveau tableau prend la valeur 0 si l'élément correspondant du tableau `image` est inférieur à la valeur `seuil` et prend la valeur 1 sinon.

Nous avons alors obtenu un tableau qui permet de repérer tous les éléments en mouvement dans l'image. Il reste maintenant à détecter les éléments en mouvement qui peuvent être une balle de tennis.

Compte-tenu de la position des caméras latérales et de leur résolution, une balle de tennis occupe au minimum 4 pixels.

Nous allons commencer par éliminer tous les pixels en mouvement qui sont isolés dans le tableau : aucun des pixels adjacents parmi les 8 possibles n'est considéré en mouvement.

- Q9.** La fonction `filtre_pixel(image)` prend en argument un tableau `image` à deux dimensions, rempli de 0 et de 1. Cette fonction doit permettre de balayer tous les éléments du tableau `image` et de tester s'ils correspondent à un pixel isolé.

Proposer deux schémas permettant d'illustrer le cas d'un pixel isolé et donc à éliminer et celui d'un pixel à conserver.

On considère un pixel de coordonnées (p,c) tel que ce pixel ne soit pas situé sur le bord de l'image. Parmi les 4 tests suivants, choisir celui qui permet de détecter si ce pixel est isolé et, si c'est le cas, de l'éliminer.

Test 1

```
| if image[c-1][p-1]==0 or image[c-1][p+1]==0 or image[c+1][p-1]==0
| or image[c+1][p+1] == 0:
|   image[p][c]=0
```

Test 2

```
| if image[c-1][p-1]==0 and image[c-1][p+1]==0 and image[c+1][p-1]==0
| and image[c+1][p+1] == 0:
|   image[p][c]=0
```

Test 3

```

if image[p][c]==1:
    nb_pixel=-1
    for k in [p-1,p,p+1]:
        for j in [c-1,c,c+1]:
            nb_pixel += image[k][j]
    if nb_pixel==0:
        image[p][c]=0

```

Test 4

```

if image[p][c]==1:
    nb_pixel=-1
    for k in range(p-1,p+1):
        for j in range(c-1,c+1):
            nb_pixel += image[k][j]
    if nb_pixel==0:
        image[p][c]=0

```

Une fonction est ensuite implémentée pour éliminer les objets en mouvement trop grands et qui ne peuvent correspondre à une balle de tennis. Ces objets sont en général représentés par des chaînes de pixels à 1 représentant le contour d'un objet en mouvement. Cette fonction ne sera pas étudiée ici.

Le script présenté ci-dessous permet de réaliser le prétraitement à partir des images extraites des vidéos enregistrées par une caméra. Les images issues d'un même point sont stockées dans un dossier `sequence_####` où `####` correspond au numéro du point joué. Ces images sont enregistrées au format `.bmp` et sont nommées `image_#####.bmp` où `#####` est le numéro de l'image dans la séquence vidéo.

```

1      point = input("Rentrer le numéro du point litigieux à
2      analyser")
3      dirs = listdir("sequence_" + point)
4      for image in dirs:
5          index_max = 0
6          index = int(image[6:11])
7          if index >= index_max:
8              index_max = index
9
10     seq_balle = [[x0,y0]]# [x0,y0] position initiale de la balle
11     # donnée par un autre traitement
12     for image in dirs:
13         index_init = index_max - 3000 # indice de la première
14         image de la séquence analysée
15         if int(image[6:11]) == index_init :
16             image2 = imread("sequence_"+ point + "/" + image)
17             elif int(image[6:11]) > index_init :
18                 image1 = deepcopy(image2)
19                 image2 = imread("sequence_"+ point + "/" + image)
20                 liste_balles = traitement(image1,image2)
21                 seq_balle.append(liste_balles)

```

Éléments de documentation

- la commande `listdir` s'applique à un dossier et renvoie une liste de tous les noms de fichiers contenus dans ce dossier,
- la commande `imread` construit un tableau $n \times m \times 3$ à partir d'une image bitmap (`.bmp`),
- la fonction `traitement` renvoie une liste des positions des balles éventuelles détectées dans une image. Cette fonction fait appel aux fonctions précédemment programmées. (Il n'est pas demandé de détailler cette fonction).

Q10. Commenter les lignes 2 et 3, puis indiquer l'intérêt de chaque boucle `for`. Préciser l'intérêt de ce programme.

Après les différents traitements définis précédemment, nous avons pour chaque image une liste des coordonnées de la balle. Malgré ces traitements, une image peut contenir plusieurs points pouvant être assimilés à la balle de tennis. Par exemple, dans le cas d'une image i qui contient 2 balles possibles, on stocke les coordonnées dans une liste :

$$\text{image } i : [x_{i1}, y_{i1}, x_{i2}, y_{i2}].$$

Chaque liste associée à une image est placée dans une liste qui regroupe ainsi toutes les images de la séquence à analyser :

$$\text{seq_balle} = \left[[x_0, y_0], \dots, \underbrace{[x_{i1}, y_{i1}, x_{i2}, y_{i2}]}_{\text{image } i}, \underbrace{[x_{(i+1)1}, y_{(i+1)1}, x_{(i+1)2}, y_{(i+1)2}], \dots}_{\text{image } i+1} \right]$$

partir de la liste `seq_balle` et de la vitesse initiale, nous allons chercher à ne conserver que les "balles bonnes" en prenant en compte la trajectoire. On suppose qu'entre deux images successives, la vitesse de la balle varie peu. Ainsi, à partir de la balle détectée à l'image $i - 1$ (de coordonnées (x_{i-1}, y_{i-1})), nous allons chercher la balle suivante dans une zone rectangulaire dont le centre (x_c, y_c) sera calculé à partir de la position de la balle (x_{i-1}, y_{i-1}) et du vecteur déplacement (d_{xi-1}, d_{yi-1}) :

$$x_c = x_{i-1} + d_{xi-1} \quad \text{et} \quad y_c = y_{i-1} + d_{yi-1}.$$

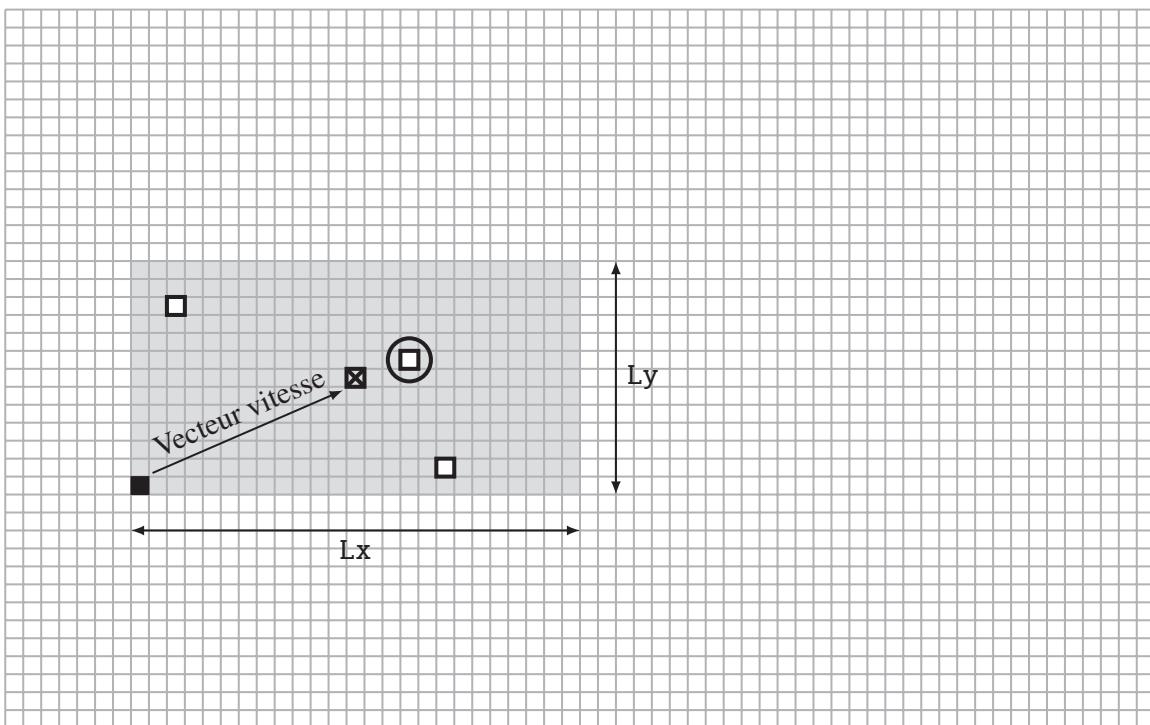
Le rectangle aura pour dimension $Lx = 2d_{xi-1} + 1$ suivant x et $Ly = 2d_{yi-1} + 1$ suivant y avec au minimum 3 pixels suivant x et 3 pixels suivant y , c'est-à-dire au minimum $Lx = 3$ et $Ly = 3$.

- Si plusieurs balles sont présentes dans cette zone (balles candidates), on choisira celle dont la position sera la plus proche du centre. En cas d'égalité, on conservera arbitrairement l'une des 2 positions.
- Si aucune balle n'est présente dans la zone, on renverra pour l'image $i + 1$ la liste `[None, None]` et on cherchera directement une balle dans l'image suivante $i + 2$ en doublant les dimensions de la zone de recherche $2Lx - 1$ et $2Ly - 1$ et en prenant comme pixel central : $x_c = x_i + 2d_{xi-1}$ et $y_c = y_i + 2d_{yi-1}$.

Les images successives étant prises à intervalles de temps égaux, on définit le vecteur déplacement à l'instant i (ou à l'image i) comme une liste à deux éléments :

$$\text{dep_i} = [d_{xi}, d_{yi}] = [x_i - x_{i-1}, y_i - y_{i-1}].$$

L'algorithme proposé dans cette sous-partie est illustré **figure 5**, page 10.



- Balle détectée à l'image précédente
- Centre de la zone de recherche
- Balles candidates
- Balle détectée

Figure 5 – Détection des "balles bonnes"

Q11.

- Écrire une fonction `deplacement(pos1, pos2)` qui prend en argument 2 listes à 2 éléments (`pos1` et `pos2`) et qui renvoie le vecteur déplacement associé (sous la forme d'une liste à deux éléments). On précisera le type des éléments constitutants les listes.
Écrire la ou les instructions permettant d'affecter aux variables `Lx` et `Ly` les dimensions de la zone de recherche.
- Écrire une fonction `distance_quad(xc, yc, liste_balle_i)` où `xc, yc` sont les coordonnées du pixel central et `liste_balle_i` la liste des balles possibles. Cette fonction doit renvoyer une liste des carrés des distances séparant chaque balle possible du pixel central.
- Écrire une fonction `cherche_balle(xc, yc, Lx, Ly, liste_balle_i)` où `xc, yc` sont les coordonnées du pixel central, `Lx, Ly` les dimensions de la zone de recherche et `liste_balle_i` la liste des balles possibles. Cette fonction doit renvoyer une liste contenant les coordonnées de la balle détectée ou la liste `[None, None]` si aucune balle n'est détectée. Cette fonction fera appel à la fonction `distance_quad`.

- d) Écrire une fonction `traj_balle` qui prend en argument la liste `seq_balle` définie précédemment et la vitesse initiale `vit_init` (liste à 2 éléments). Cette fonction doit renvoyer la liste des "balles bonnes" présentes dans la zone de recherche :

$$[[x_0, y_0], [x_1, y_1], \dots, [x_i, y_i], \dots].$$

On ne traitera pas le cas où dans 2 images successives aucune balle ne se trouve dans la zone de recherche.

Q12. Indiquer les limites ou défauts de l'algorithme étudié dans cette sous-partie.

III.3 - Calcul de la position 3D de la balle par triangulation (étape ④)

L'objectif de cette sous-partie est de déterminer la position de la balle dans le repère global défini par rapport au terrain à partir de la position de la balle dans deux images de deux caméras différentes.

Une fois que la configuration des caméras a été calibrée, la position de la balle peut être reconstruite dans le repère 3D par stéréo-triangulation. Cela consiste à déterminer la localisation du centre de la balle (noté $M(x, y, z)$ de coordonnées (x, y, z) dans le repère global) à partir de la connaissance de la position du centre de la balle dans les images obtenues par deux caméras. La position de la balle dépend de l'espacement e des caméras, également appelé "longueur de la ligne de base stéréo des caméras", de la distance focale f des caméras et de la disparité d qui est la distance entre les centres de la balle dans les images de chaque caméra, ainsi que des coordonnées du centre de la balle dans chacune des images (**figure 6**).

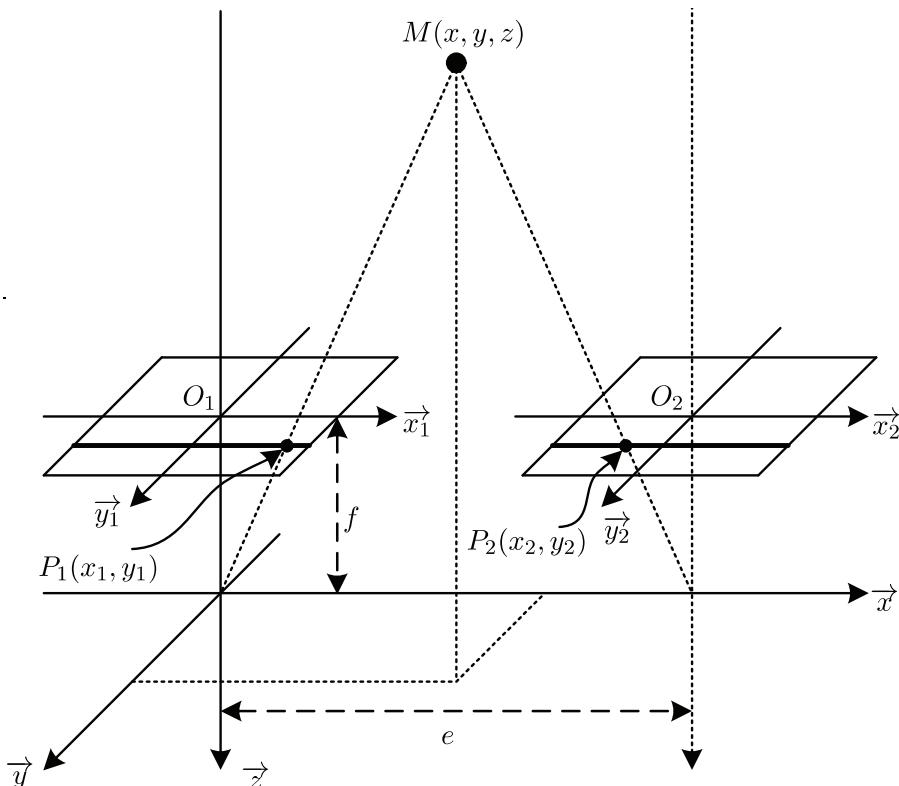


Figure 6 – Position de la balle à l'instant i dans les images des caméras 1 et 2

Hypothèses

- On s'intéressera ici à la triangulation sur deux caméras : les caméras 1 et 2.
- On suppose que la balle est bien détectée par les deux caméras et que les coordonnées relevées correspondent au centre de la balle. Les coordonnées du centre de la balle à l'instant i sont notées respectivement : P_1 de coordonnées (x_1, y_1) pour la caméra 1 et P_2 de coordonnées (x_2, y_2) pour la caméra 2 dans leurs repères respectifs.
- On suppose que les deux caméras sont disposées à la même hauteur autour du terrain et que leur plan focal est identique.
- On suppose que le temps est bien synchronisé sur les deux caméras.
- On suppose que tous les paramètres de calibration des deux caméras sont connus (matrices intrinsèques et extrinsèques, distance focale, position...).

On calcule les coordonnées du centre de la balle à l'instant i dans le repère de la caméra 1 (x_{1i}, y_{1i}, z_{1i}) grâce aux formules suivantes : $x_{1i} = \frac{e}{d}x_1$, $y_{1i} = \frac{e}{d}y_1$ et $z_{1i} = \frac{e}{d}f$, avec $d = x_1 - x_2$. Ces coordonnées sont placées dans une liste telle que : $\text{pos1i}=[x_{1i}, y_{1i}, z_{1i}]$.

Q13. Écrire une fonction $\text{pos_loc}(e, f, x1, x2, y1, y2)$ qui prend en argument les positions de la balle dans la caméra 1 (x_1, y_1) et la caméra 2 (x_2, y_2) à un instant i puis calcule et renvoie une liste des coordonnées (x_{1i}, y_{1i}, z_{1i}) de la balle dans le repère local de la caméra 1 à un instant i $\text{pos1i}=[x_{1i}, y_{1i}, z_{1i}]$.

On calcule ensuite les coordonnées x , y et z dans le repère global à l'aide des matrices de passage de la caméra 1 par rapport au repère global. La position et l'orientation de la caméra 1 par rapport au repère global du système sont définies par trois matrices : T sa matrice de translation suivant \vec{x} , \vec{y} et \vec{z} , R_x sa matrice de rotation d'angle θ suivant \vec{x} et R_y sa matrice de rotation d'angle γ suivant \vec{y} . On obtient la même chose pour la caméra 2 en remplaçant les 1 par des 2. Ces matrices sont supposées connues à chaque instant (résultats de l'étape ② non abordée). La **figure 7** illustre la position de la caméra 1 à l'instant t .

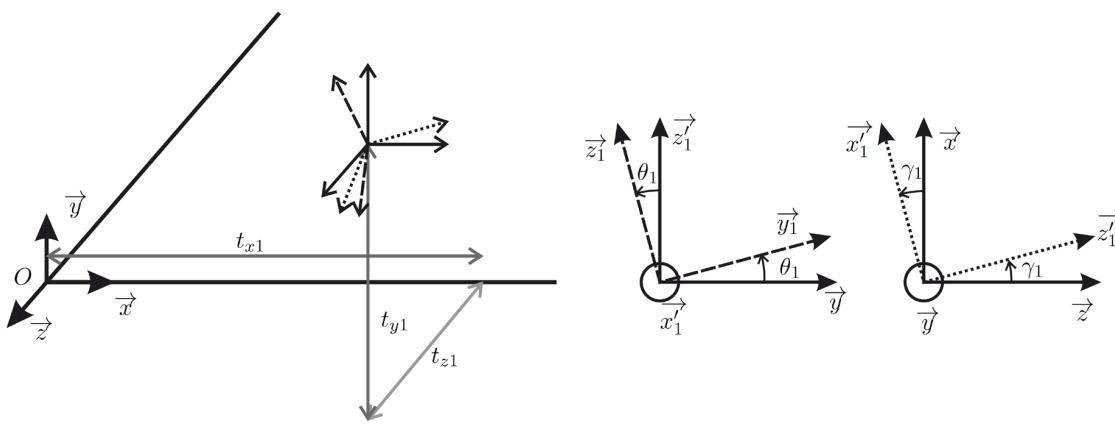


Figure 7 – Position de la caméra 1 à l'instant t

$$T_1 = \begin{pmatrix} t_{x1} \\ t_{y1} \\ t_{z1} \end{pmatrix}; R_{x1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{pmatrix} \text{ et } R_{y1} = \begin{pmatrix} \cos \gamma_1 & 0 & \sin \gamma_1 \\ 0 & 1 & 0 \\ -\sin \gamma_1 & 0 & \cos \gamma_1 \end{pmatrix}.$$

On rappelle que si R_1 est la matrice de rotation entre les bases 0 et 1, alors $R_1 \cdot U_1 = U_0$, avec U_0 un

vecteur défini dans la base 0 et U_1 un vecteur défini dans la base 1.

Q14. Écrire une fonction `pos_glo(pos1i, T1, Rx1, Ry1)` qui prend en argument la liste `pos1i` et les matrices de translation et rotations `T1`, `Rx1` et `Ry1`, qui calcule et renvoie une liste des coordonnées de la balle dans le repère global à l'instant i `posgi=[xgi, ygi, zgi]`. On pourra utiliser les fonctions de numpy pour effectuer les produits matrice-vecteur qui sont rappelées en annexe.

III.4 - Assemblage des positions 3D (étape ⑤)

L'objectif de cette partie est d'assembler les positions 3D de la balle dans le repère global obtenues à l'étape ④.

Les hypothèses sont les mêmes que dans la sous-partie III.3. Les coordonnées locales successives de la balle dans le repère de la caméra 1 sont stockées dans une liste `coord_loc=[[x10, y10, z10], [x11, y11, z11], ..., [x1i, y1i, z1i], ..., [x1N, y1N, z1N]]`. La trajectoire complète est obtenue en assemblant les positions successives de la balle dans la repère global.

Q15. Écrire une fonction `traj3D(coord_loc, T1, Rx1, Ry1)` qui prend en argument la liste `coord_loc` et renvoie une liste de l'assemblage dans l'ordre chronologique des N coordonnées de la balle dans le repère global `coord_glo=[[xg0, yg0, zg0], [xg1, yg1, zg1], ..., [xgi, ygi, zgi], ..., [xgN, ygN, zgN]]`. On pourra utiliser des fonctions de la sous-partie III.3.

III.5 - Détermination de l'impact de la balle avec le sol sur le terrain (étape ⑥)

L'objectif de cette partie est de déterminer le point d'impact de la balle avec le sol afin de savoir si la balle est restée dans les limites du terrain ou non.

Les hypothèses sont les mêmes que dans la sous-partie III.3.

Cette partie de l'algorithme est primordiale, en effet, c'est à partir du résultat obtenu par cette partie de l'algorithme que l'arbitre rendra sa décision. Le système réel permet de déterminer la forme de l'impact entre la balle et le sol en fonction de la déformation de la balle. Afin de faciliter la résolution du problème, la déformation de la balle ne sera pas prise en compte.

Hypothèses

- On suppose que la balle est indéformable (et donc que le point d'impact est ponctuel) et de rayon 0,033 m.
- La position du terrain (et donc les lignes) est parfaitement connue dans le repère global.
- On considère dans notre cas que le point étudié se fait durant l'échange de balle entre les joueurs et non sur un service.
- On suppose que le terrain est parfaitement plat.

Les coordonnées du point d'impact de la balle avec le sol sont déterminées de la manière suivante : on vient détecter quand la coordonnée suivant y est égale à R et on relève les positions correspondantes suivant x et z .

Si la position y comprise entre R et $R+\varepsilon$ ne se trouve dans aucune image, on détermine les coordonnées moyennes suivant x et z des deux images successives où il y a eu inversion du sens de déplacement de la balle selon y (descente, puis remontée). La donnée ε sera notée `eps` dans le programme.

- Q16.** Écrire une fonction `det_impact(coord_glo, eps)` qui prend en argument la liste `coord_glo` et le flottant `eps` puis calcule et renvoie la position de l'impact de la balle dans le repère global : une liste de dimension 2 `impact=[x_imp, z_imp]`.

Le terrain comme présenté sur la **figure 3** fait $L = 24$ m de long sur $l = 11$ m de large. On considère que le point origine du repère global est le point O .

- Q17.** Écrire une fonction `res_final(impact, L, l)` qui prend en argument la liste `impact`, les dimensions du terrain `L` et `l` et qui renvoie IN si l'impact de la balle a eu lieu dans les limites du terrain et OUT sinon.

III.6 - Reconstruction de la trajectoire 3D de la balle (étape 7)

L'objectif de cette sous-partie est d'afficher la trajectoire 3D de la balle reconstruite à l'étape 5 pour que les spectateurs puissent la visualiser.

- Q18.** partir de la documentation sur le tracé 3D en Python, fournie en annexe, écrire une fonction `vis_traj3D(coord_glo)` qui prend en argument la liste `coord_glo` et permet d'afficher la trajectoire 3D de la balle.

Partie IV - Exploitation des données fournies par le Hawk-eye

L'objectif de cette partie est de montrer comment utiliser les informations fournies par le système Hawk-eye à des fins d'entraînement ou de visualisation à la télévision.

Le système Hawk-eye est une mine d'informations pour analyser les statistiques des matchs de tennis. Les données sont ainsi utilisées par les chaînes de télévisions lors des retransmissions des matchs mais également par les entraîneurs après les matchs en vue d'adapter leurs programmes d'entraînement.

Ces données d'un tournoi sont stockées dans une base de données constituée de deux tables.

La table MATCHS contient les différents matchs d'un tournoi avec les attributs :

- `id` : identifiant de type entier, clé primaire ;
- `nom` : nom du match de type texte ;
- `numero` : numéro du match dans la planification du tournoi ;
- `date` : date où le match s'est déroulé ;
- `joueur1` : nom du premier joueur ;
- `joueur2` : nom du deuxième joueur (les joueurs 1 et 2 sont rangés par ordre alphabétique) ;
- autres attributs non détaillés...

La table POINTS contient des entités correspondant aux points joués lors d'un match. Elle contient les attributs :

- `id` : identifiant de type entier, clé primaire ;
- `mid` : identifiant du match correspondant à la définition de type entier ;
- `nombre` : nombre d'échanges de type entier ;
- `fichier` : nom du fichier image de la trajectoire (stockée) correspondant au point ;
- autres attributs non détaillés...

- Q19.** Rappeler la définition et l'intérêt d'une clé primaire.

- Q20.** Écrire une requête SQL permettant d'afficher les identifiants des matchs joués par Federer, joueur pris pour exemple.

Q21. Écrire une requête SQL permettant d'afficher le nombre d'échanges maximum lors du match dont l'identifiant du match est `mid=4`.

Q22. Écrire une requête SQL permettant de récupérer le nom des fichiers de toutes les images qui correspondent au match dont le nom est "Federer-Murray".

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
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] ou M[1][2] donne 5
extraire une portion de tableau (2 premières colonnes)	M[:,0:2]
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 (est identique pour le produit matrice-matrice)	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 principal sur une figure	title(texte)

SESSION 2018**TSIIN07****ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI****INFORMATIQUE****Jeudi 3 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 est composé de 6 parties qui peuvent être traitées indépendamment.
Il est néanmoins recommandé de les aborder dans l'ordre.**

Le sujet comporte :

- le texte du sujet, 13 pages,
- l'annexe, 3 pages,
- le document-réponse à rendre, 12 pages.

Dans tout le sujet, on considérera à chaque étape que les fonctions définies lors des questions précédentes sont utilisables pour la suite du sujet.

Vous devez répondre directement sur le document réponse, soit à l'emplacement prévu pour la réponse lorsque celle-ci implique une rédaction, soit en complétant les différents programmes en langage Python. Si vous manquez de place dans les cadres prévus, vous pouvez compléter sur la dernière page en précisant la question.

OPTIMISATION D'UN CORRECTEUR P.I.D. PAR UN ALGORITHME GÉNÉTIQUE

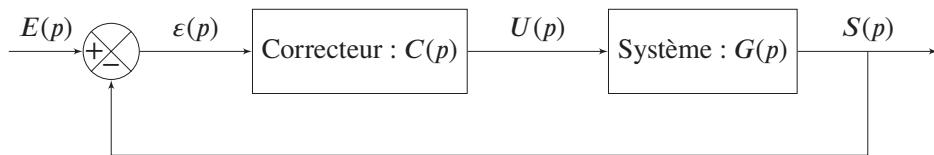
Partie I - Problématique

La correction d'un système asservi linéaire, en vue de répondre aux exigences de précision, rapidité et d'oscillations, n'est jamais aisée et est souvent réalisée à partir de méthodes pratiques plus ou moins empiriques.

Le correcteur Proportionnel, Dérivateur, Intégrateur (P.I.D.) est fréquemment employé afin d'améliorer le comportement des systèmes asservis. On se propose de mettre en œuvre une procédure d'optimisation du correcteur P.I.D. basée sur les algorithmes génétiques.

I.1 - Présentation du problème

On considère le système asservi à retour unitaire décrit par le schéma-bloc ci-dessous :



- on note p la variable de Laplace, les grandeurs seront notées en minuscule dans le domaine temporel et en majuscule dans le domaine de Laplace tel que $\mathcal{L}(e(t)) = E(p)$;
- le correcteur est un correcteur P.I.D. (parfait) de fonction de transfert $C(p)$;
- le système à réguler est connu par sa fonction de transfert $G(p) = \frac{N_G(p)}{D_G(p)}$ où $N_G(p)$ et $D_G(p)$ sont deux polynômes avec $\deg(N_G(p)) = k < \deg(D_G(p)) = \ell$. On pose :
 - $N_G(p) = n_0 + n_1 \cdot p + n_2 \cdot p^2 + \dots + n_k \cdot p^k$,
 - $D_G(p) = d_0 + d_1 \cdot p + d_2 \cdot p^2 + \dots + d_\ell \cdot p^\ell$.

Afin de déterminer la réponse temporelle du système corrigé, nous avons besoin de définir la fonction de transfert en boucle ouverte $BO(p)$, puis la fonction de transfert en boucle fermée $BF(p)$.

On rappelle que dans le cadre d'un système asservi à retour unitaire, on a :

$$BO(p) = C(p) \cdot G(p) = \frac{N_O(p)}{D_O(p)} \quad \text{et} \quad BF(p) = \frac{BO(p)}{1 + BO(p)} = \frac{N_F(p)}{D_F(p)} = \frac{N_O(p)}{N_O(p) + D_O(p)}.$$

On a alors

$$S(p) = BF(p) \cdot E(p)$$

et la réponse temporelle $s(t)$ s'obtient en recherchant la transformée inverse de Laplace de $S(p)$.

Pour réaliser cette étude, nous allons utiliser la librairie **scipy.signal** qui permet, à partir de la fonction de transfert d'un système :

- d'obtenir la réponse temporelle à un échelon unitaire ;
- d'obtenir les racines d'un polynôme.

L'**annexe A.1** des pages 14 et 15, montre un exemple d'utilisation et précise les quelques commandes utiles de cette librairie dans le cadre de ce sujet.

I.2 - Système à réguler

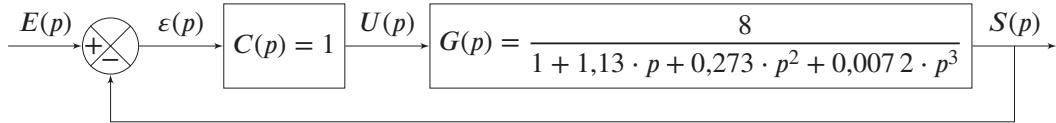
Le système que nous allons étudier est décrit par la fonction de transfert :

$$G(p) = \frac{8}{1 + 1,13 \cdot p + 0,273 \cdot p^2 + 0,0072 \cdot p^3}.$$

On note **numG** et **denG** les listes associées au numérateur $N_G(p)$ et au dénominateur $D_G(p)$ de la fonction de transfert $G(p)$.

- Q1.** À l'aide de l'**annexe A.1**, page 14 et de l'expression de $G(p)$, compléter les lignes 1 et 3 du **listing de la page DR 1 du document réponse**.

Afin d'optimiser la réponse temporelle on réalise un asservissement. Le schéma-bloc ci-dessous représente cet asservissement.



Pour ce premier essai, on considère $C(p) = 1$. La réponse temporelle $s(t)$ à un échelon unitaire $e(t)$ de ce système est tracée sur la **figure 1**.

On constate que la réponse temporelle du système asservi (**figure 1**) présente à la fois un défaut de précision (la valeur de consigne n'est pas atteinte), un temps de réponse d'environ 1,8 s, des oscillations et un premier dépassement relativement important.

Il semble nécessaire de choisir un correcteur qui améliore notamment la réponse du système.

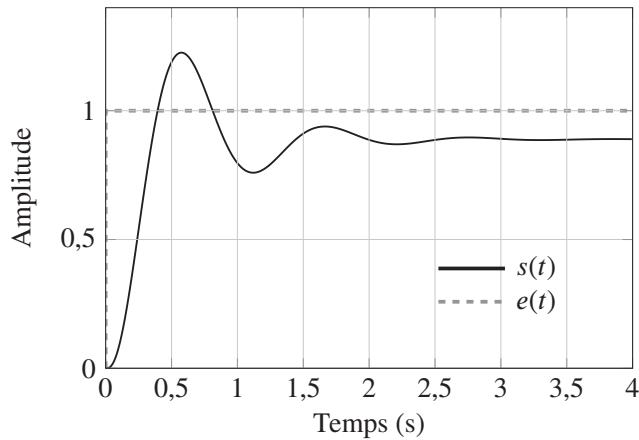


Figure 1 – Réponse temporelle à un échelon unitaire du système corrigé avec $C(p) = 1$

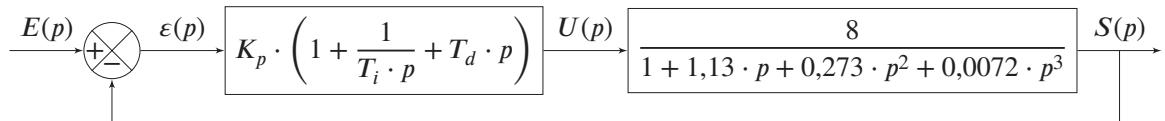
Partie II - Correction du système

II.1 - Définition du correcteur P.I.D.

On installe dans le système un correcteur P.I.D. :

$$C(p) = K_p \cdot \left(1 + \frac{1}{T_i \cdot p} + T_d \cdot p \right).$$

Le système devient :



- Q2.** Mettre le correcteur sous la forme $C(p) = \frac{N_C(p)}{D_C(p)}$ avec $N_C(p)$ et $D_C(p)$ deux polynômes à définir. Puis, sur le **listing page DR 1**, écrire la fonction **correcteur(Kp,Ti,Td)** qui renvoie les listes **numC** et **denC** associées au numérateur et au dénominateur du correcteur P.I.D.

Le numérateur et le dénominateur du correcteur seront nommés respectivement **numC** et **denC** pour la suite.

II.2 - Détermination des fonctions de transfert en boucle ouverte et en boucle fermée

La détermination de la fonction de transfert en boucle ouverte $BO(p)$ et de la fonction de transfert en boucle fermée $BF(p)$ nécessite de développer deux fonctions : une fonction qui réalise le produit de deux polynômes et une autre qui en fait la somme.

Soit la fonction **multi_listes(P,Q)** (**listing page DR 2**) qui prend comme argument deux listes **P** et **Q**.

- Q3.** Décrire l'exécution de la fonction **multi_listes(P,Q)** lorsque les deux listes sont **P=[a,b,c]** et **Q=[r,s]** en précisant sur le **document réponse page DR 2** dans le cadre correspondant à cette question, le contenu des variables suivantes :

- de la liste Q1 avant la boucle for ;
- de la liste R pour $k = 2$ et $i = 1$;
- de la liste R à la fin de l'exécution.

Que fait la fonction **multi_listes(P,Q)** ?

Afin que le résultat du produit des deux listes corresponde au produit de deux polynômes, tels qu'ils sont définis par la librairie **scipy.signal**, il est nécessaire de ranger les termes de ce résultat dans l'ordre inverse.

Pour la suite, chaque fois qu'une liste sera l'image d'un polynôme, on considérera que les coefficients sont rangés dans l'ordre décroissant des puissances de p (voir la définition de la fonction **lti()** en **annexe A.1**).

- Q4.** Écrire la fonction **inverse(liste)** qui prend comme argument une liste **liste** et renvoie une liste **liste_r** en inversant les coefficients sur le **listing page DR 2** dans le cadre correspondant à cette question.

Soient $F_1(p) = \frac{N_1(p)}{D_1(p)}$ et $F_2(p) = \frac{N_2(p)}{D_2(p)}$ deux fonctions de transfert avec $N_i(p)$ et $D_i(p)$ des polynômes en p . On cherche à déterminer $F_1(p) \times F_2(p)$.

On note **num1**, **num2**, **den1**, **den2** les quatre listes représentant les polynômes $N_1(p)$, $N_2(p)$, $D_1(p)$ et $D_2(p)$.

- Q5.** Écrire la fonction **multi_FT(num1,den1,num2,den2)** du **listing page DR 3** dans le cadre correspondant à cette question qui prend comme argument quatre listes représentant les polynômes du numérateur et du dénominateur de deux fonctions de transfert et renvoie deux listes représentant le produit de ces deux fonctions de transfert rangées comme des polynômes. Pour cela, vous utiliserez les deux fonctions **multi_listes(P,Q)** et **inverse(liste)**.

Il reste à déterminer la fonction qui réalise la somme de deux polynômes.

- Q6.** Écrire la fonction **somme_poly(P,Q)** qui prend en argument deux listes représentant les coefficients de deux polynômes et renvoie une liste des coefficients de la somme des deux polynômes. Ces deux polynômes sont de dimensions différentes.

II.3 - Tracé de la réponse temporelle

Finalement, pour obtenir la réponse temporelle, il est nécessaire de compléter les fonctions définies précédemment par deux fonctions :

- **FTBF(num,den)**, fonction qui renvoie le numérateur et le dénominateur de la fonction de transfert en boucle fermée $\frac{S(p)}{E(p)}$
- **rep_Temp(Kp,Ti,Td)**, fonction qui permet d'obtenir les deux listes **t** (le temps) et **s** l'amplitude de la sortie pour un échelon unitaire.

Ces deux fonctions sont définies ci-dessous.

Le listing du programme de la page 6 montre une utilisation de ces fonctions permettant de tracer la réponse temporelle pour un échelon unitaire en prenant comme paramètres du correcteur **Kp=5**, **Ti=1,1** et **Td=0,218**. La figure 2 représente le tracé de la réponse temporelle correspondante.

Sur cette figure, on retrouve la consigne d'entrée $e(t)$ en trait pointillé, la sortie corrigée $s(t)$ en trait fort et les deux limites à $\pm 5\%$ autour de la valeur finale en tracés fins.

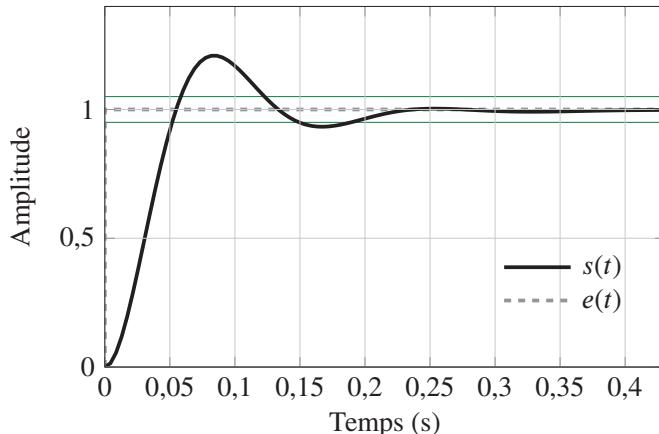


Figure 2 – Tracé de la réponse temporelle corrigée

Définition des fonctions FTBF() et rep_Temp()

```

1 def FTBF(num,den) :
2     num_bf=num
3     den_bf=somme_poly(num,den)
4     return num_bf,den_bf
5
6 def rep_Temp(Kp,Ti,Td) :
7     numC,denC=correcteur(Kp,Ti,Td)
8     num_BO, den_BO=multi_FT(numG,denG,numC,denC)
9     num_BF,den_BF=FTBF(num_BO,den_BO)
10    BF=lti(num_BF, den_BF)
11    temps, sortie = step(BF)
12    temps, sortie = step(BF, T = linspace(min(temps), temps[-1], 500))
13    return temps,sortie

```

Programme de tracé de la réponse temporelle

```

1 numG = [ .... ]
2 denG = [.....] #Question 1
3 #définition du correcteur (valeurs quelconques)
4 Kp,Ti,Td=5,1,1,0.218
5 t,s= rep_Temp(Kp,Ti,Td)
6 plt.plot(t[:60], s[:60])
7 plt.title('Réponse temporelle à un échelon')
8 plt.xlabel('Temps (s)')
9 plt.ylabel('Amplitude')
10 plt.hlines(1, min(t), t[60], colors='r')
11 plt.hlines(0.95, min(t), t[60], colors='g')
12 plt.hlines(1.05, min(t), t[60], colors='g')
13 plt.hlines(0, min(t), t[60])
14 plt.grid()
15 plt.show()

```

Partie III - Détermination des critères d'optimisation

III.1 - Réglage optimal pratique

Dans le cas qui nous intéresse, la présence d'une intégration garantit que l'erreur indicelle sera nulle à la condition que le système reste stable. Une méthode de réglage pratique (méthode de compensation des pôles) a permis de déterminer les valeurs suivantes pour le correcteur :

$$\begin{aligned}K_{p0} &= 2,33, \\T_{i0} &= 1,1 \text{ s}, \\T_{d0} &= 0,218 \text{ s}.\end{aligned}$$

Le tracé (**figure 3**) montre une réponse temporelle qui semble optimisée.

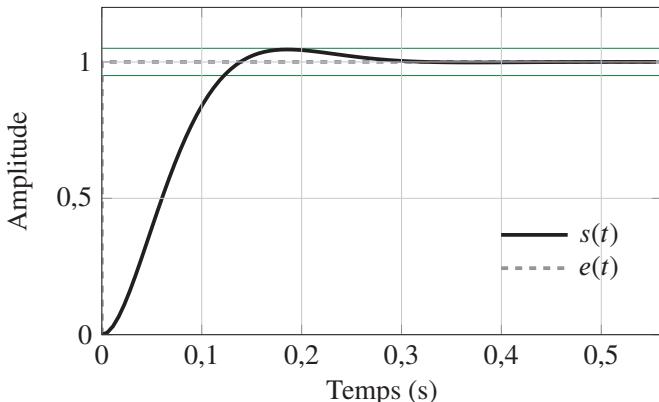


Figure 3 – Réponse temporelle optimisée obtenue par la méthode de compensation des pôles

L'objet de l'étude qui suit est de déterminer un réglage du correcteur P.I.D. qui améliore encore la réponse temporelle.

III.2 - Stabilité

Avant de définir les différents critères d'optimisation de la réponse temporelle, il est nécessaire de vérifier que la réponse temporelle est convergente. La réponse est convergente si le système est stable. On rappelle la définition de la **stabilité** : une fonction de transfert est stable si toutes les racines du dénominateur (les pôles) sont à parties réelles strictement négatives.

On cherche donc à vérifier que la fonction de transfert obtenue est stable et ce, pour différentes valeurs de K_p , T_i et T_d .

- Q7.** Écrire la fonction **stabilite(P)** qui prend comme argument une liste représentant un polynôme et renvoie la valeur binaire **True** si la fonction de transfert, dont le dénominateur est $P(p)$, est stable et **False** dans l'autre cas. Vous utiliserez les fonctions **roots(...)** et **real(...)** de la librairie **numpy** (voir l'**annexe A.2**, page 15).

III.3 - Critères usuels d'optimisation

Les critères usuels de réglage et d'optimisation d'un système asservi généralement retenus sont le temps de réponse à 5 %, l'amplitude des dépassements et une erreur indicielle nulle. On tente souvent d'avoir une réponse temporelle analogue à la réponse d'un système du second ordre avec un temps de réponse minimal et un dépassement limité à 5 %.

On rappelle ici les définitions :

Temps de réponse à 5 % : le temps de réponse à 5 % est le temps mis par le système pour que la réponse temporelle atteigne la valeur finale à 5 % près ;

Dépassement relatif : le dépassement relatif est défini par $D_1 = \frac{s_{max} - s_\infty}{s_\infty}$ avec s_{max} la valeur maximale de la fonction et s_∞ la valeur finale en régime permanent.

- Q8.** Justifier que la fonction **Temps_reponse(s,t)** (listing page DR 4) qui prend comme argument deux listes, la réponse temporelle et le temps, renvoie une valeur approchée majorant le temps de réponse à 5 %.

- Q9.** Proposer une ré-écriture de la fonction **Temps_reponse(s,t)** en utilisant une seule boucle *while* à la place de la boucle *for* et du test.

- Q10.** Écrire la fonction **dépassemment(s,t)** qui prend en argument les listes **s** et **t** et qui renvoie le dépassement relatif.

III.4 - Critère de la valeur absolue de l'intégrale de l'erreur

À ces critères, on peut en rajouter un qui va caractériser la réponse transitoire : il s'agit de l'intégrale de la valeur absolue de l'erreur (IAE) :

$$IAE = \int_0^{+\infty} |\varepsilon(t)| dt$$

avec $s(t)$ la réponse temporelle de la sortie, $e(t) = 1$ la consigne d'entrée et $\varepsilon(t) = e(t) - s(t)$.

Ce critère permet d'évaluer l'aire de la surface entre la courbe de réponse $s(t)$ et la consigne d'entrée $e(t) = 1$ (figure 4).

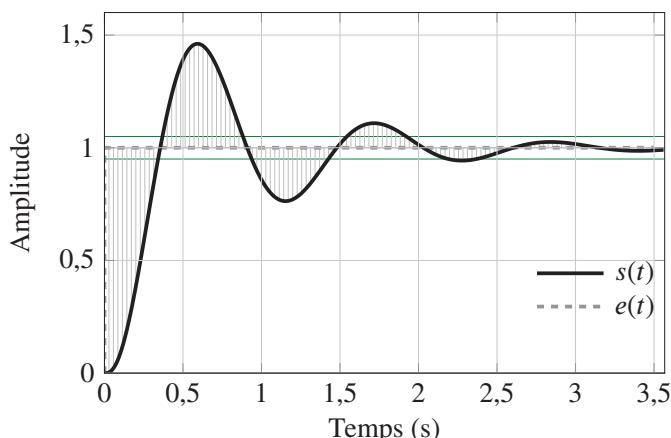


Figure 4 – Critère de la valeur absolue de l'erreur

- Q11.** Écrire la fonction **critere_IAE(s,t)** qui prend comme argument les listes **s** et **t** et qui renvoie l'intégrale de la valeur absolue de l'erreur. Vous préciserez la méthode utilisée.

III.5 - Fonction coût

Si on applique ces trois critères à la fonction réglée par la méthode de compensation des pôles (**figure 3**) avec $K_{p0} = 2,33$, $T_{i0} = 1,1$ s et $T_{d0} = 0,218$ s, on obtient respectivement pour les critères de temps de réponse ($T_{5\%}$), de dépassement relatif (D_1) et d'écart (IAE) :

$$\begin{aligned}T_{50} &= 0,123 \text{ s}, \\D_{10} &= 5 \% = 0,05, \\IAE_0 &= 0,0733 \text{ USI}.\end{aligned}$$

Ces trois critères sont ceux que notre algorithme doit permettre d'améliorer. On définit ainsi une fonction **ponderation_cout(T5,D1,IAE)** (**listing page DR 5**) qui renvoie le « coût » global associé à ces critères. Le réglage optimal du correcteur doit alors permettre de minimiser cette fonction.

Q12. Dans quel intervalle doit être la valeur renournée par **ponderation_cout(T5,D1,IAE)** afin que la configuration soit meilleure que la réponse de référence ?

Nous allons chercher le meilleur triplet (K_p, T_i, T_d) qui minimise la fonction coût.

Les domaines de définition de K_p , T_i et T_d sont :

- $K_p \in [0,01, 100]$,
- $T_i \in [0,01 \text{ s}, 100 \text{ s}]$,
- $T_d \in [0 \text{ s}, 50 \text{ s}]$.

Afin de limiter l'étude, nous considérerons que chacune des trois variables ne peut prendre que 2^{16} valeurs réparties uniformément dans l'intervalle de définition :

$$K_p = (K_{pmax} - K_{pmin}) \cdot \frac{I_p}{2^{16} - 1} + K_{pmin}, \quad (1)$$

$$T_i = (T_{imax} - T_{imin}) \cdot \frac{I_i}{2^{16} - 1} + T_{imin}, \quad (2)$$

$$T_d = (T_{dmax} - T_{dmin}) \cdot \frac{I_d}{2^{16} - 1} + T_{dmin}. \quad (3)$$

La fonction **calcul_coef_correcteur(Ip,Ii,Id)** renvoie les coefficients du correcteur K_p , T_i et T_d .

Q13. Compléter la fonction **calcul_coef_correcteur(Ip,Ii,Id)** qui détermine les coefficients du correcteur à partir des trois valeurs entières I_p , I_i et I_d . Estimer le nombre de combinaisons possibles pour le correcteur.

La fonction **calcul_cout(Ip,Ii,Id)** doit permettre, pour une combinaison des trois valeurs (**Ip,Ii,Id**) dans leur domaine de définition, de déterminer le coût de la combinaison. Pour cela, il faut :

- déterminer les coefficients du correcteur P.I.D. ;
- déterminer le numérateur et le dénominateur de la fonction de transfert en boucle ouverte ;
- déterminer le numérateur et le dénominateur de la fonction de transfert en boucle fermée ;
- si la fonction de transfert en boucle fermée :
 - est stable, alors déterminer le temps de réponse, le dépassement et le critère IAE ;
 - n'est pas stable, alors prendre pour le coût une valeur supérieure aux valeurs possibles (ici coût = 100).

Finalement, la fonction doit renvoyer le coût.

Rappel des fonctions précédemment définies à utiliser :

stabilite(P),

Temps_reponse(s,t),

critere_IAE(s,t),

rep_Temp(Kp,Ti,Td),

dépassemement(s,t),

pondération_cout(T5,D1,IAE).

Q14. Compléter la fonction **calcul_cout(I_p,I_i,I_d)** en utilisant les fonctions précédemment définies.

Q15. Sachant que le temps de calcul de la fonction **calcul_cout(I_p,I_i,I_d)** pour une combinaison est de 10,3 ms, est-il envisageable de les tester toutes ?

Pour la suite, nous utiliserons la fonction **calcul_cout(I_p,I_i,I_d)** qui, à partir des trois valeurs entières I_p , I_i et I_d renvoie le coût de la combinaison. C'est cette quantité que nous chercherons à minimiser dans la partie suivante.

Partie IV - Résolution par un algorithme génétique

IV.1 - Principe

Les algorithmes génétiques sont des processus d'optimisation itératifs évolutionnistes. Ils permettent d'approcher la solution d'un problème d'optimisation (ici de recherche d'un minimum) en reproduisant des mécanismes de « sélection naturelle ». Le principe, décrit **figure 5**, consiste à initialiser un groupe de candidats possibles dont on va sélectionner les meilleurs éléments. Ces derniers sont conservés et croisés pour obtenir de nouveaux candidats. Ainsi, à chaque génération, la population se conserve et les caractéristiques du groupe s'améliorent jusqu'à converger vers un optimum.

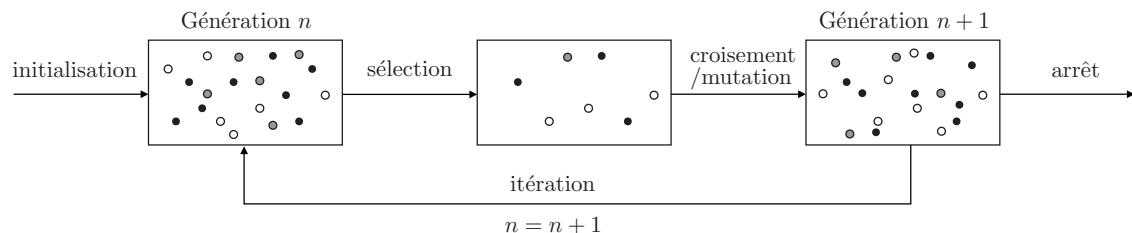


Figure 5 – Principe d'un algorithme génétique

Nous nous intéressons ici au programme, présenté dans son ensemble dans l'**annexe A.3**, page 16, qui doit permettre de trouver une combinaison des paramètres I_p , I_i et I_d qui présente un optimum vis-à-vis de la minimisation de la fonction **calcul_cout(I_p,I_i,I_d)** définie dans la partie précédente.

Nous utiliserons une population de 100 candidats. À chaque génération, seuls les 20 meilleurs sont sélectionnés et croisés de façon à générer 80 nouveaux candidats et conserver une population de 100 candidats. Afin d'éviter de converger trop rapidement vers un minimum local, les nouveaux candidats subissent une mutation aléatoire pour apporter de la diversité génétique.

Ce cycle est répété jusqu'à ce qu'un critère d'arrêt vienne l'interrompre. Pour le moment nous supposons que 50 cycles permettent d'obtenir une bonne convergence. Le meilleur individu de la dernière génération servira alors pour le réglage des paramètres I_p , I_i et I_d .

IV.2 - Initialisation des candidats

Chaque candidat possède trois gènes correspondant à chacun des paramètres I_p , I_i et I_d qui peuvent prendre 2^{16} valeurs distinctes (voir les équations (1), (2) et (3) page 8). Pour faciliter la manipulation des gènes et leurs croisements, ceux-ci sont codés sous la forme d'une chaîne de 16 caractères '0' ou '1' correspondant à un nombre binaire.

Q16. Définir le rôle de la fonction **genererGene(n)** et le type du paramètre **gen2** renvoyé. Quelle valeur de **n** faut-il alors employer pour créer un gène ?

Chaque candidat est pourvu de 3 gènes stockés sous la forme d'une liste **Candidat** de 3 chaînes de 16 caractères, par exemple :

Candidat=[’1000100010001000’,’1000101010000100’,’1000111001101000’].

Q17. Compléter la fonction **generer_liste_initiale(n)** de façon à générer une liste **CandidatS** de 100 candidats aléatoires sur le **listing page DR 7**.

Par ailleurs, pour employer la fonction **calcul_cout(Ip,Ii,Id)** définie page 8, il est nécessaire de traduire la valeur décimale associée au gène codé en binaire à l'aide d'une fonction **décodage()**. Soit un gène **b2** codé sur n caractères binaires, la commande **float(b2[0])** renvoie la valeur du bit de poids le plus faible (le premier) et **float(b2[n-1])** renvoie la valeur du bit de poids le plus fort (le n-ième).

Q18. Calculer le nombre décimal associé à **b2=’1001001000000000’**.

Q19. Compléter la fonction **decodage(b2)** qui prend en argument une chaîne de caractère **b2** et qui renvoie le nombre décimal **b10** associé.

IV.3 - Tri et sélection des candidats

Parmi les candidats triés dans l'ordre croissant de leur performance, ne sont sélectionnés et stockés que les 20 meilleurs candidats dans la liste **CandidatS_top** (**listing page DR 8**). Cette liste doit permettre d'une part de stocker ces candidats dans une base de données et d'autre part de constituer et générer la génération suivante.

Q20. Quel est l'algorithme de tri utilisé dans la fonction **tri(L)** du **listing page DR 8** pour trier **L** contenant les candidats ? Donner sa complexité dans le meilleur et le pire des cas.

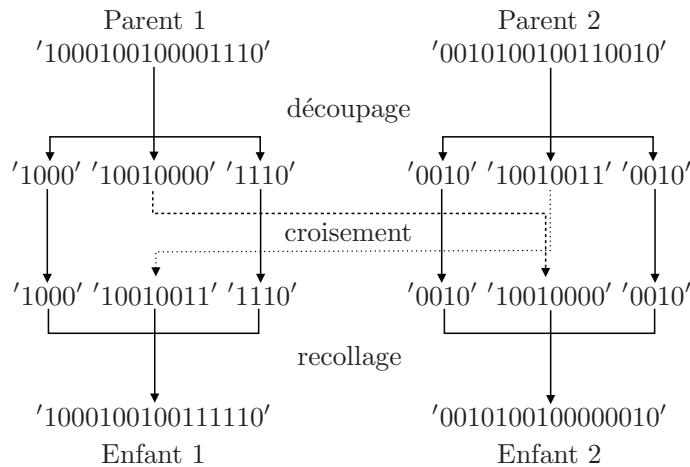
IV.4 - Croisement des candidats

La fonction **croisement(P1,P2)** doit permettre de générer deux nouveaux individus à partir de deux parents **P1** et **P2**. Les gènes des deux parents sont découpés, croisés et recombinés 2 à 2 de façon à générer deux nouveaux gènes.

Chacun des trois gènes des deux parents est donc découpé en trois morceaux (**figure 6**). La partie centrale allant du 5^e au 12^e bit est échangée de façon à créer les gènes de deux enfants. Ainsi, le couple de parents (**P1**, **P2**) ci-dessous génère, avant mutation, les deux enfants (**E1**, **E2**) :

P1=[’1111111111111111’,’0000000000000000’,’1111111100000000’],
P2=[’0000000000000000’,’1111111111111111’,’0000000011111111’],

E1=[’1111000000001111’,’000011111110000’,’1111000011110000’],
E2=[’00001111110000’,’111100000001111’,’000011100001111’].

**Figure 6 – Principe du croisement**

Q21. Compléter le **listing** de sorte que la fonction **croisement(P1,P2)** renvoie les deux enfants **E1, E2** issus de deux parents **P1, P2**.

Avant d'être intégré à la nouvelle génération et afin de limiter le risque d'apparition de clone, chaque enfant subit une mutation sur l'un de ces gènes.

Q22. Pour **Candidat=[‘1111111111111111’, ‘0000000000000000’, ‘1111111100000000’]**, donner ce que renvoie la commande **mutation(Candidat,1,12)** du **listing page DR 9**.

L'étape de croisement des 20 meilleurs candidats, qui sont conservés, doit permettre de générer 80 nouveaux candidats de façon à revenir à une population de 100 individus :

- 1) 20 sont issus de 10 croisements du meilleur des 20 meilleurs candidats avec un des 19 autres candidats (première boucle **for** du **listing page DR 9**);
- 2) 60 sont issus du croisement de 30 couples aléatoirement formés des 20 meilleurs candidats excepté le meilleur (deuxième boucle **for** du **listing page DR 9**).

Q23. Compléter la fonction **nouvGeneration(L)** du **listing page DR 9** de façon à traduire le point 2) ci-dessus.

Il est malgré tout possible que des clones apparaissent à chaque génération, d'autant plus que les candidats risquent de se ressembler au fur et à mesure des itérations. Pour éviter ce problème, on utilise une fonction **doublons()** qui repère les clones et qui les remplace par un candidat tiré aléatoirement.

Q24. Créer la fonction **doublons(L)** du **listing page DR 10** qui prend en argument une liste **L** de candidats et qui renvoie cette même liste débarrassée de ses clones et les remplace par un candidat tiré aléatoirement.

Partie V - Historique des candidats

Dans l'objectif d'étudier les paramètres qui permettent d'accélérer l'algorithme, nous décidons de stocker tous les meilleurs candidats de chaque itération dans une base de données. L'objectif est de déterminer un critère d'arrêt plus efficace que celui défini au-dessus.

La base de données est ainsi constituée d'une table Historique (**tableau 1**) qui permet de stocker les meilleurs candidats au fil des itérations. En plus de l'Id et des trois paramètres flottants K_p , T_i et T_d , elle permet de stocker trois autres attributs : score, apparition et disparition (**tableau 1**).

Historique
Id
gene_Kp
gene_Ti
gene_Td
score
apparition
disparition

Tableau 1 – Table Historique

L'attribut score est un flottant correspondant au résultat de la fonction coût du candidat. Les attributs apparition et disparition sont des entiers et correspondent respectivement au numéro de l'itération où le candidat est apparu et au numéro de celle où il a disparu. On se basera ici sur l'extrait de cette base de données fourni **tableau 2**.

Q25. Définir le but et le résultat de la requête SQL suivante :

SELECT min(score) FROM Historique;

Q26. Donner la requête SQL qui permet d'obtenir la longévité du meilleur candidat. Donner le résultat de cette requête et conclure sur la possibilité de stopper l'algorithme génétique avant la 50^e itération.

Q27. Définir le but et le résultat de la requête SQL suivante :

SELECT Id FROM Historique WHERE 15 > apparition AND 15 < disparition.

Q28. Donner la requête SQL qui renvoie la valeur moyenne des gènes Kp, Ti et Td à la 20^e itération.

Partie VI - En conclusion

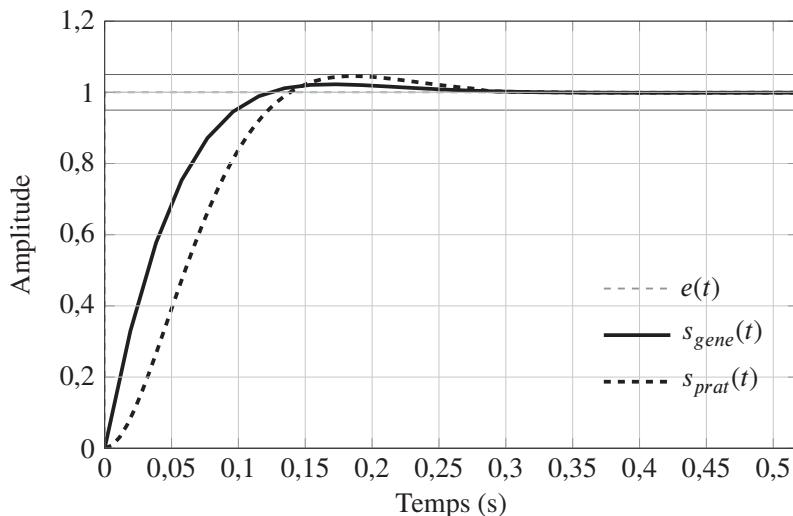


Figure 7 – Tracés comparés de la réponse temporelle déterminée par l'algorithme génétique et la réponse déterminée expérimentalement par la méthode de compensation des pôles

On retrouve sur la **figure 7** le tracé de la réponse « pratique » pour laquelle les coefficients du correcteur P.I.D. ont été déterminés expérimentalement ($s_{prat}(t)$) et le tracé de la réponse temporelle correspondant à la meilleure réponse de l'algorithme génétique ($s_{gene}(t)$).

Q29. Conclure sur l'opportunité d'utiliser un algorithme génétique.

Id	gene_Kp	gene_Ti	gene_Td	score	apparition	disparition
1	65.0542311742	25.9980929274	8.0804150454	1.1465868477	0	2
2	94.0221130694	21.8343222705	12.8343633173	1.47592141334	0	0
3	62.1003799496	12.9620883497	13.1906614786	1.53111188552	0	0
4	63.3377628748	5.39437033646	0.718699931334	1.66453693499	0	0
5	86.7534419776	39.6642320897	7.62493324178	1.76681066691	0	0
6	68.0126596475	2.17198718242	1.37788967727	0.720574832859	1	5
7	49.5983877317	6.71109224079	9.67421988251	1.23600614063	1	2
8	44.8868729686	7.2237441062	10.4859998474	1.24050982879	1	1
9	81.7001593042	29.2158988327	7.68596932937	1.24876332573	1	1
10	65.0542311742	25.9980929274	8.07431143664	1.16317640581	2	2
11	46.4492405585	7.2237441062	10.4921034562	1.24017983164	2	2
12	62.9593769741	4.02729869535	1.43892576486	0.750913688522	3	5
13	68.2079555962	2.17198718242	1.37788967727	0.774983006043	3	4
...
43	65.1686623941	0.417375143053	1.37865262837	0.621608931066	10	10
44	65.1808683909	0.411272144656	1.37865262837	0.620980646341	11	11
45	65.1808683909	0.412797894255	1.37865262837	0.621000002184	11	11
46	65.2296923781	0.402117647059	1.37865262837	0.618622779634	12	16
47	65.2296923781	0.42652964065	1.37865262837	0.618936823201	12	13
48	65.1869713893	0.414323643854	1.37865262837	0.620742325779	12	12
49	65.1823941405	0.402117647059	1.37865262837	0.620793058622	12	12
50	65.2357953765	0.414323643854	1.37865262837	0.618499911309	13	18
51	65.2296923781	0.414323643854	1.37865262837	0.618782759248	13	13
52	65.1884971389	0.408220645457	1.37865262837	0.620595007049	13	13
53	65.2327438773	0.414323643854	1.37865262837	0.618641425052	14	15
54	65.2296923781	0.408220645457	1.37865262837	0.618703560070	14	14
55	65.2312181277	0.414323643854	1.37865262837	0.618712114622	14	14
56	65.2342696269	0.414323643854	1.37865262837	0.618570690595	15	16
57	65.2312181277	0.408220645457	1.37865262837	0.61863287033	15	15
58	65.2357953765	0.406694895857	1.37865262837	0.618400625578	16	18
59	65.2373211261	0.414323643854	1.37865262837	0.618429087253	16	18
60	65.2373211261	0.402117647059	1.37865262837	0.618268651270	17	49
61	65.2373211261	0.406694895857	1.37865262837	0.618329745253	17	20
62	65.2373211261	0.405169146258	1.37865262837	0.618309616353	19	49
63	65.2357953765	0.402117647059	1.37865262837	0.61833956633	19	20
64	65.2373211261	0.408220645457	1.37865262837	0.618349663168	19	20
65	65.2373211261	0.400591897459	1.37865262837	0.618248479198	21	49
66	65.2373211261	0.403643396658	1.37865262837	0.618289307302	21	49
67	65.2357953765	0.400591897459	1.37865262837	0.618319406010	21	49

Tableau 2 – Historique partiel des meilleurs candidats

FIN

ANNEXE

A.1 Librairie `scipy.signal`

Le code ci-dessous illustre l'utilisation de cette librairie pour tracer la réponse temporelle d'un système du second ordre :

$$G(p) = \frac{K}{1 + 2 \cdot z \cdot \frac{p}{\omega_n} + \frac{p^2}{\omega_n^2}}.$$

Exemple d'utilisation de la librairie `scipy.signal`

```

from numpy import min, max, zeros
from scipy import linspace
from scipy.signal import lti, step, bode
from matplotlib import pyplot as plt
z=0.3
wn=10
K=1.3
d0=1
d1=2*z/wn
d2=1/wn**2
#Numérateur
num = [K]
#Dénominateur
den = [d2, d1, d0]
# Fonction de transfert
G = lti(num, den) #déclaration de la fonction de transfert
t, s = step(G)
# Ici, on appelle de nouveau la fonction afin de préciser
# le nombre de points à afficher
t, s = step(G, T = linspace(min(t), t[-1], 1000))
#Affichage
plt.plot(t, s)
plt.title('Réponse temporelle')
plt.xlabel('Temps(s)')
plt.ylabel('Amplitude')
plt.hlines(1, min(t), max(t), colors='r') #tracé de l'échelon unitaire
plt.hlines(0, min(t), max(t))
plt.xlim(xmax = max(t))
plt.legend(('Réponse temporelle à un échelon',), loc=0)
plt.grid()
plt.show()

```

Nous utiliserons dans ce sujet les deux fonctions définies sur la page suivante.

lti(num,den) : cette fonction (ligne 16 au-dessus) permet de définir la fonction de transfert, elle prend comme argument deux listes, l'une comportant les coefficients du numérateur, l'autre comportant les coefficients du dénominateur (lignes 12 et 14).

Les coefficients des deux listes sont rangés dans l'ordre décroissant.

Si les polynômes du numérateur et du dénominateur sont notés :

$$N(p) = n_0 + n_1 \cdot p + n_2 \cdot p^2 + \dots + n_k \cdot p^k$$

$$D(p) = d_0 + d_1 \cdot p + d_2 \cdot p^2 + \dots + d_\ell \cdot p^\ell$$

alors les deux listes représentant les polynômes utilisables par la fonction **lti(num,den)** sont notées :

$$\text{num} = [n_k, n_{k-1}, \dots, n_2, n_1, n_0],$$

$$\text{den} = [d_\ell, d_{\ell-1}, \dots, d_2, d_1, d_0].$$

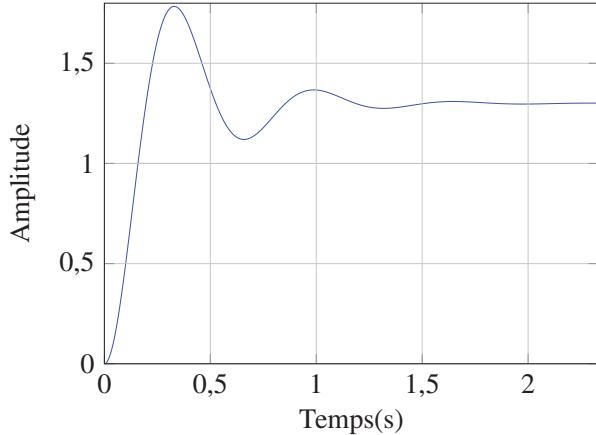


Figure 8 – Exemple d'utilisation de la librairie `scipy.signal`

step(G) : cette fonction de la librairie `scipy.signal` renvoie deux listes, le temps et la réponse temporelle pour une entrée en échelon unitaire de la fonction de transfert G . Si la fonction est appelée sans préciser la durée de simulation et le nombre de points, ceux-ci sont déterminés automatiquement par la librairie. Il est d'usage de l'appeler deux fois, la première sans paramètres pour laisser la librairie calculer le temps de simulation, puis une deuxième fois en reprenant le temps calculé et en imposant le nombre de points. La représentation temporelle est tracée sur la **figure 8**.

A.2 Librairie numpy

min(liste), max(liste) : ces deux fonctions qui prennent une liste comme argument, retournent respectivement le minimum et le maximum d'une liste.

roots(P) : cette fonction qui prend comme argument une liste dont les termes sont les coefficients d'un polynôme, renvoie une liste dont les termes sont les racines du polynôme.

```
p = [1, -3, -2, -1] # le polynôme P(x)=x^3-3x^2-2x-1
racines=roots(p)
print(racines)
```

Le script affiche : [3.62736508 + 0.j, -0.31368254 + 0.42105281j, -0.31368254 - 0.42105281j]

real(Q) : fonction dont l'argument est une liste et qui renvoie une liste constituée de la partie réelle de chaque terme de la liste passée en argument.

```
print(real(racines))
```

Le script affiche : [3.62736508, -0.31368254, -0.31368254]

zeros(n) : retourne une liste de n termes nuls.

A.3 Programme d'optimisation génétique

Cette annexe présente la structure générale du programme d'optimisation génétique et les différentes fonctions que vous aurez à commenter ou à définir. Ce programme doit permettre de trouver un optimum à la fonction **calcul_cout(Ip,li,Id)** à l'aide d'algorithme génétique.

Programme d'optimisation par algorithme génétique

```
#-----Paramètres globaux-----
n=_
cycle=50
#-----Fonctions-----
def genererGene(n):
    # Détermination d'un gène
    return gen2

def generer_liste_initiale(n):
    # Génération de la liste initiale
    return liste_initiale

def decodage(b2):
    return b10

def perf(Li): # Li est la liste des attributs d'un candidat
    # Détermination de la performance d'une combinaison
    return calcul_cout(Ip ,li ,Id)

def tri(L): # L est la liste de tous les candidats
    # Tri croissant des différents candidats selon la performance
    return L

def croisement(P1,P2):
    # Fonction réalisant le croisement génétique
    return

def mutation(E,i,j):
    # Fonction réalisant une mutation
    return E

def nouvGeneration(L): # L la liste actuelle des candidats
    # Fonction déterminant la nouvelle génération
    return L_new

def doublons(L):
    # Fonction chargée d'éliminer les doublons

# Finalement le programme principal
#-----Main-----
CandidatS=Generer_liste_initiale(n)
for i in range(cycle):
    CandidatS_top=tri(CandidatS)[0 :20]
    CandidatS=nouvGeneration(CandidatS_top)
    CandidatS=doublons(CandidatS)
solution=CandidatS [0]
```

<p>Académie :</p> <p>Examen ou Concours : <u>Concours Communs Polytechniques</u></p> <p>Spécialité/option : <u>FILIÈRE TSI</u></p> <p>Épreuve/sous-épreuve : <u>INFORMATIQUE</u></p> <p>NOM : (en majuscules, suivi, s'il y a lieu, du nom d'épouse)</p> <p>Prénoms : _____</p> <p>Né(e) le _____</p>	<p>Session :</p> <p>Série* : _____</p> <p>Repère de l'épreuve : _____</p> <p>N° du candidat _____</p> <p><i>(le numéro est celui qui figure sur la convocation ou la liste d'appel)</i></p>
<p>Examen ou Concours : <u>Concours Communs Polytechniques</u> Série* : _____</p> <p>Spécialité/option : <u>FILIÈRE TSI</u></p> <p>Repère de l'épreuve : <u>INFORMATIQUE</u></p> <p>Épreuve/sous-épreuve : _____</p> <p><i>(Préciser, s'il y a lieu, le sujet choisi)</i></p>	
<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <p>Note :</p> <div style="text-align: right; margin-right: -10px;">  </div> <div style="text-align: right; margin-right: -10px;">20</div> </div>	<p>Appréciation du correcteur* :</p>

DOCUMENT RÉPONSE

TSIIN07

Pour l'ensemble du sujet, les bibliothèques usuelles sont chargées avec la syntaxe suivante :

Librairies

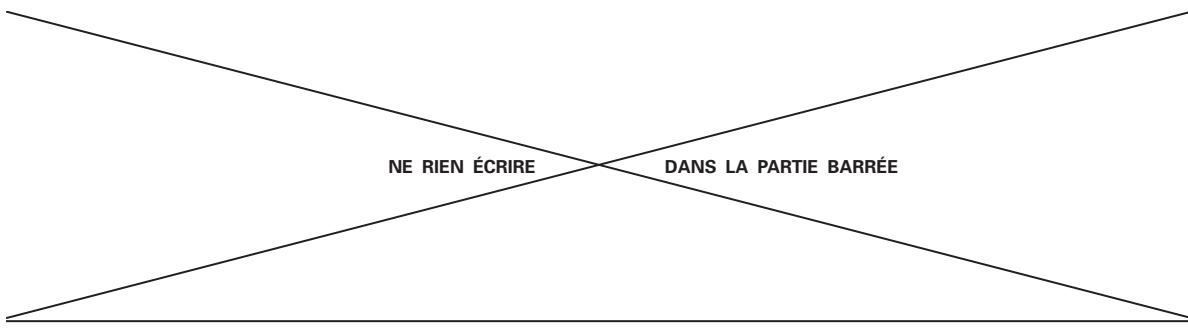
```
1 from numpy import min, max, zeros, roots, real
2 from scipy import linspace
3 from scipy.signal import lti, step
4 from matplotlib import pyplot as plt
5 from random import random
```

Question 1

```
1 numG = [ ...
2
3 denG = [ ...
4
5 G = lti(numG, denG)
```

Question 2

```
1 def correcteur(Kp,Ti,Td) :
2     ...
3     ...
4     ...
5     ...
6
7
8
9
10
11
12
13     return num,den
14 numC,denC=correcteur(Kp,Ti,Td) # appel de la fonction, numC et denC les deux listes de coefficients.
```

**Question 3**

```

1 def multi_listes(P,Q) :
2     deg_max=(len(P)-1)+(len(Q)-1)
3     P1=zeros(deg_max+1)
4     Q1=zeros(deg_max+1)
5     #P1 et Q1 deux listes de zéros de dimension deg_max+1
6     for j in range(len(P)) :
7         P1[j]=P[j]
8         for j in range(len(Q)) :
9             Q1[j]=Q[j]
10        R=zeros(deg_max+1)
11        for k in range(deg_max+1) :
12            for i in range(k+1) :
13                R[k]=R[k]+P1[i]*Q1[k-i]
14    return R

```

Q1=

.....
(k = 2 et i = 1) : R=

.....
R=

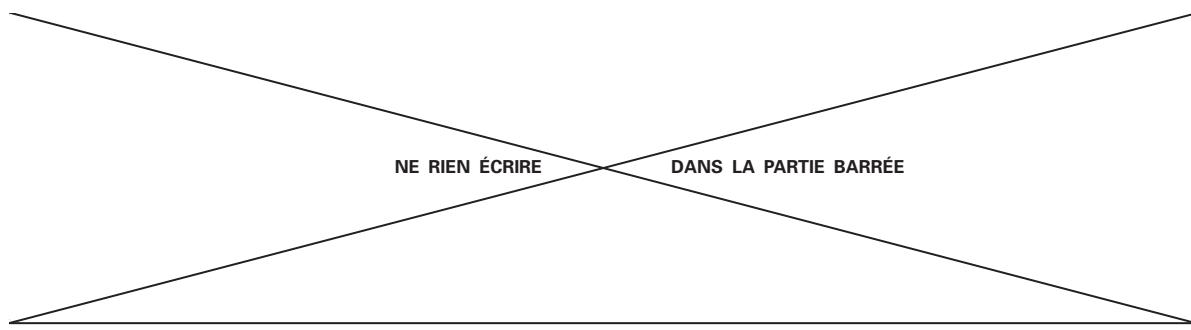
Que fait multi_listes?.....

Question 4

```

1 def inverse(liste) :
2     ...
3     ...
4     ...
5     ...
6     ...
7     ...
8     ...
9     ...
10    ...
11    ...
12    ...
13    return liste_r

```

**Question 5**

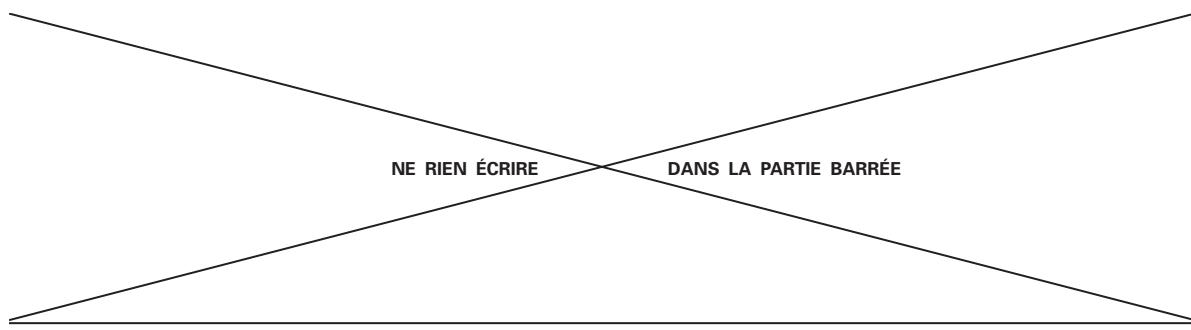
```
1 def multi_FT(num1,den1,num2,den2) :  
2     ...  
3     ...  
4     ...  
5     ...  
6     ...  
7     ...  
8     ...  
9     ...  
10    ...  
11    ...  
12    ...  
13    ...  
14    ...  
15        return ... ... ...
```

Question 6

```
1 def somme_poly(P,Q) :  
2     ...  
3     ...  
4     ...  
5     ...  
6     ...  
7     ...  
8     ...  
9     ...  
10    ...  
11    ...  
12    ...  
13    ...  
14    ...  
15        ...  
16        return somme
```

Question 7

```
1 def stabilité(P) :  
2     ...  
3     ...  
4     ...  
5     ...  
6     ...  
7     ...  
8     ...  
9     ...  
10    ...  
11    ...  
12    ...  
13    ...  
14        return ...
```

**Question 8**

```
1 def Temps_reponse(s,t) :
2     T5=0
3     s_fin=s[-1]
4     for tt in range(len(s)) :
5         j=len(t)-tt-1
6         if ((s[j]>s_fin*0.95 and s[j-1]<s_fin*0.95)or(s[j]<s_fin*1.05 and s[j-1]>s_fin*1.05)) :
7             T5=t[j]
8             break
9     return T5
```

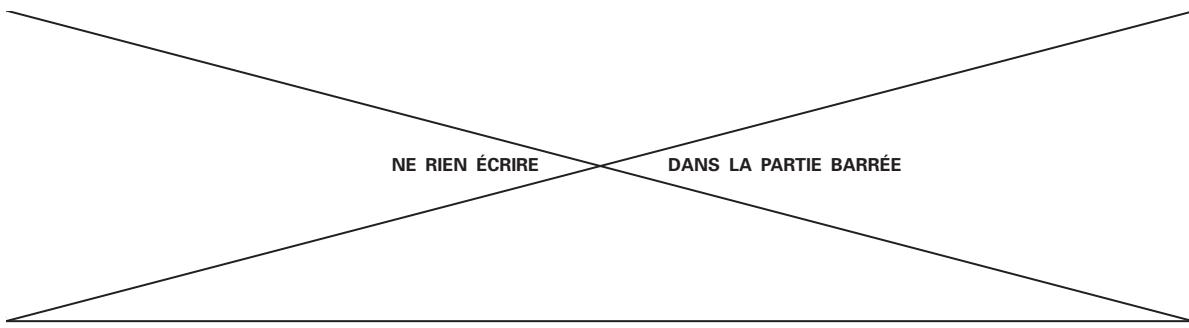
.....
.....
.....
.....
.....
.....

Question 9

```
1 def Temps_reponse(s,t) :
2     ...
3     ...
4     ...
5     ...
6     ...
7     ...
8     ...
9     ...
10    ...
11    ...
12    ...
13    return T5
```

Question 10

```
1 def depassement(s,t) :
2     ...
3     ...
4     ...
5     ...
6     ...
7     ...
8     ...
9     ...
10    ...
11    ...
12    ...
13    return D1
```

**Question 11**

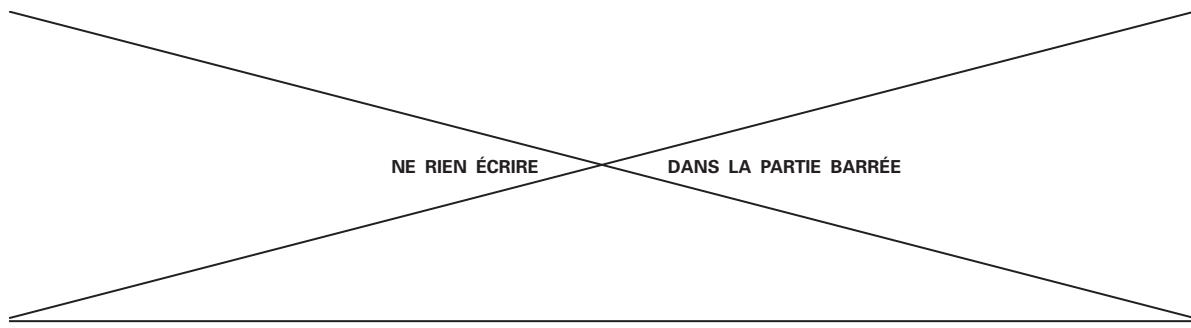
```
1 def critere_IAE(s,t) :  
2     ...  
3     ...  
4     ...  
5     ...  
6     ...  
7     ...  
8     ...  
9     ...  
10    ...  
11    return IAE
```

.....
.....
.....
.....
.....
.....
.....
.....
.....

Question 12

```
1 T50=0.123  
2 D10=0.05  
3 IAE0=0.0733  
4 def ponderation_cout(T5,D1,IAE) :  
5     k1=1  
6     k2=1  
7     k3=1  
8     cout=1/(k1+k2+k3)*(k1*T5/T50+k2*D1/D10+k3*IAE/IAE0)  
9     return cout
```

.....
.....
.....
.....
.....
.....

**Question 13**

```
1 def calcul_coef_correcteur(Ip,li,ld) :  
2     ...  
3     ...  
4     ...  
5     ...  
6     ...  
7     ...  
8     ...  
9     ...  
10    ...  
11    ...  
12    ...  
13    ...  
14    return Kp,Ti,Td
```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Question 14

```
1 def calcul_cout(Ip,li,ld) :  
2     Kp,Ti,Td=coef_correcteur(Ip,li,ld)  
3     numC,denC=correcteur(Kp,Ti,Td)  
4     num_BO, den_BO=multi_FT(numG,denG,numC,denC)  
5     num_BF,den_BF=FTBF(num_BO,den_BO)  
6     stable=...  
7     ...  
8     ...  
9     ...  
10    ...  
11    ...  
12    ...  
13    ...  
14    ...  
15    ...  
16    ...  
17    ...  
18    ...  
19    ...  
20    ...  
21    ...  
22    ...  
23    return cout
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 15

Question 16

```
1 n=...
2 def genererGene(n) :
3     b2=""
4     for i in range(n) :
5         b2=b2+str(int(random())*2))
6     return b2
```

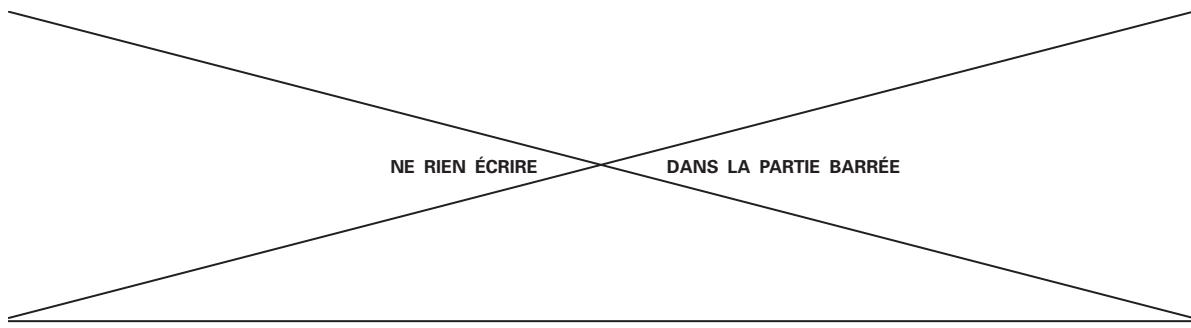
.....
.....
.....
.....
.....

Question 17

```
1 def generer_liste_initiale(n) :
2     CandidatS = []
3     for i in ... :
4         Candidat = ...
5         CandidatS.append(Candidat)
6     return CandidatS
```

Question 18 _____

.....
.....
.....
.....
.....
.....
.....

**Question 19**

```

1 def decodage(b2) :
2     b10=0
3     for...
4
5
6
7
8
9
10    return b10

```

Programme principal

```

1 cycle=50
2 #----- Main -----
3 CandidatS=Generer_liste_initiale(n)
4 for i in range(cycle):
5     CandidatS_top=tri(CandidatS)[0 :20]
6     CandidatS=nouvGeneration(CandidatS_top)
7     CandidatS=doublons(CandidatS)
8 solution=CandidatS[0]

```

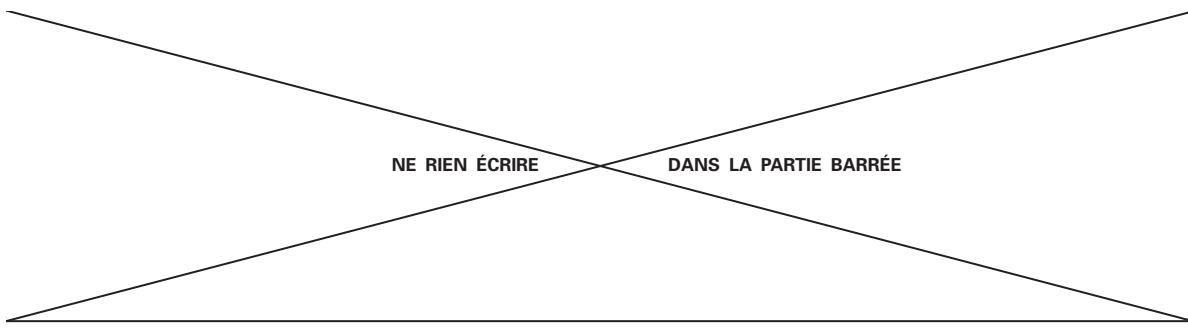
Question 20

```

1 def perf(Li) : # Li est la liste des attributs d'un candidat
2     lp,li,id=decodage(Li[0]),decodage(Li[1]),decodage(Li[2])
3     return calcul_cout(lp,li,id)
4
5 def tri(L) : # L est la liste de tous les candidats
6     for i in range(1,len(L)) :
7         if perf(L[i])<perf(L[i-1]) :
8             p=i
9             while p>0 and perf(L[i])<perf(L[p-1]) :
10                 p=p-1
11                 X=L.pop(i)
12                 L.insert(p,X)
13
return L

```

.....
.....
.....
.....
.....
.....
.....
.....

**Question 21**

```

1 def croisement(P1,P2) :
2     E1=[]
3     E2=[]
4     ...
5     ...
6
7
8
9
10
11
12
13
14
15     return ...

```

Question 22

```

1 def mutation(E,i,j) :
2     if E[i][j]==-'1' :
3         E[i]=E[i][:j]+'0'+E[i][j+1 :]
4     else :
5         E[i]=E[i][:j]+'1'+E[i][j+1 :]
6     return E
.....
```

Question 23

```

1 def nouvGeneration(L) : # L la liste actuelle des candidats
2     L_new=L
3     for i in range(10) :
4         E1,E2=croisement(L[0],L[int(random()*19)+1])
5         L_new.append(mutation(E1,int(random()*3),int(random()*16)))
6         L_new.append(mutation(E2,int(random()*3),int(random()*16)))
7     for i in ... :
8         ...
9         ...
10        ...
11        ...
12
13
14
15
16
17     return L_new

```

NE RIEN ÉCRIRE

DANS LA PARTIE BARRÉE

Question 24

```
1 def doublons ...
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
```

Cadre réponse pour les questions 25 à 29

Préciser le numéro de la question, séparer les réponses.

DR 10/ 12

NE RIEN ÉCRIRE

DANS LA PARTIE BARRÉE

Cadre réponse pour les questions 25 à 29 (suite)

Préciser le numéro de la question, séparer les réponses.

~~NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE~~

Cadre réponse libre

Préciser le numéro de la question, séparer les réponses.



ÉPREUVE SPÉCIFIQUE - FILIÈRE TPC

MODÉLISATION

Jeudi 3 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 deux parties (pages 1 à 14)
et d'une annexe (pages 15 à 18).**

PROBLÈME

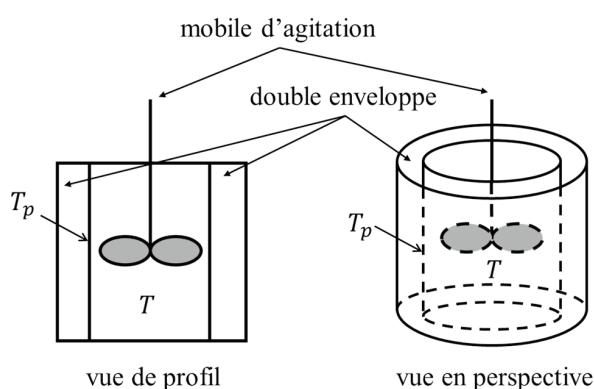
Étude d'une réaction exothermique : stabilité thermique en réacteur fermé

Présentation du problème

De nombreux procédés industriels font intervenir des réacteurs fermés pour la synthèse de molécules à haute valeur ajoutée. Pour optimiser le rendement de la synthèse, il est nécessaire de bien comprendre l'influence des paramètres physiques (comme la température de la réaction,...) sur la marche du réacteur. La maîtrise des échanges thermiques est cruciale dans le cas des réactions exothermiques car la chaleur dégagée par la réaction provoque une augmentation de la température du mélange réactionnel. Selon les conditions opératoires, cette augmentation de température peut entraîner un emballement thermique du réacteur et conduire à des dégâts irréversibles.

L'accident de Seveso le 10 juillet 1976 illustre les problèmes liés à l'emballement thermique des réacteurs. Il s'agissait d'un procédé de production de 2,4,5-trichloro-phénol à partir de 1,2,4,5-tétrachloro-benzène et de soude dans un solvant (l'éthylène glycol) à une température voisine de 150 °C et à pression atmosphérique en réacteur fermé. La mauvaise maîtrise de la température de la réaction a entraîné le déroulement de réactions secondaires conduisant d'une part à une augmentation de la température et de la pression et d'autre part à la formation de produits secondaires toxiques : les dioxines. La rupture de la soupape de sécurité due à l'augmentation de la pression a conduit au rejet de dioxines dans l'atmosphère.

L'étude de l'influence des paramètres physiques sur la marche d'un réacteur se fait la plupart du temps à l'échelle du laboratoire dans des dispositifs de dimensions beaucoup plus petites que celles des réacteurs utilisés dans les procédés industriels. La particularité du réacteur utilisé pour la présente étude est qu'il possède une géométrie cylindrique et qu'il est équipé d'une double enveloppe externe dans laquelle circule un fluide permettant de refroidir la paroi du réacteur et d'empêcher un emballement thermique (**figure 1**). L'emballement thermique survient lorsque la chaleur dégagée par la réaction excède la capacité du réacteur à évacuer l'énergie.



T et T_p représentent respectivement la température dans le réacteur et la température à la paroi (côté refroidissement).

Figure 1 – Schéma simplifié d'un réacteur fermé parfaitement agité avec une double enveloppe pour son refroidissement

Pour caractériser le comportement thermique du réacteur, on commence la plupart du temps par une étude en l'absence de réaction. Cette étude permet dans un premier temps de caractériser la capacité du réacteur à évacuer l'énergie avec la détermination du coefficient de transfert thermique à la paroi. Dans un deuxième temps, on met en œuvre dans ce réacteur une réaction exothermique $R \rightarrow \text{produits}$. Ces études permettent de déterminer les valeurs de paramètres clefs intervenant dans les équations décrivant le comportement du réacteur (modèle théorique). L'utilisation de ce modèle théorique permet de prédire la stabilité thermique du réacteur. L'établissement du modèle théorique repose sur l'écriture de bilans de matière et de chaleur. Une fois que l'influence des conditions physiques sur la marche du réacteur est déterminée, on peut en déduire les conditions de stabilité du réacteur industriel.

Ce sujet est constitué de deux parties. La première partie porte sur la modélisation du réacteur fermé parfaitement agité avec double enveloppe. Elle permet l'établissement du système d'équations différentielles régissant les variations de la conversion du réactif et de la température en fonction du temps, ainsi que la détermination de la valeur de paramètres physico-chimiques intervenant dans ces équations. La deuxième partie porte sur le traitement numérique des données expérimentales avec la détermination des paramètres d'un modèle par régression linéaire, puis la prédiction du comportement thermique du réacteur par résolution d'un système d'équations différentielles par la méthode d'Euler implicite.

Partie I – Modélisation du réacteur fermé parfaitement agité avec double enveloppe

I.1 – Etalonnage thermique du réacteur

Dans cette partie, on souhaite caractériser les transferts de chaleur entre un liquide contenu à l'intérieur du réacteur et la paroi en l'absence de toute réaction chimique. On supposera que la capacité thermique massique et la masse volumique de ce liquide sont constantes quelle que soit la température. Pour caractériser ces transferts de chaleur, on réalise deux expériences successives avec la température de la paroi, T_p , maintenue constante dans les deux cas.

- La première expérience consiste à chauffer le liquide (initialement à une température identique à celle de la paroi) avec un dispositif annexe (une résistance chauffante) dissipant une puissance thermique P_{th} de 96,0 W. On constate que la température de la phase liquide augmente, puis atteint une valeur asymptotique en régime permanent (**figure 2a**).
- Après avoir atteint le régime permanent lors de la phase de chauffe, on réalise une seconde expérience en coupant le chauffage. La température du liquide décroît jusqu'à ce qu'elle tende vers la température de la paroi (**figure 2b**).

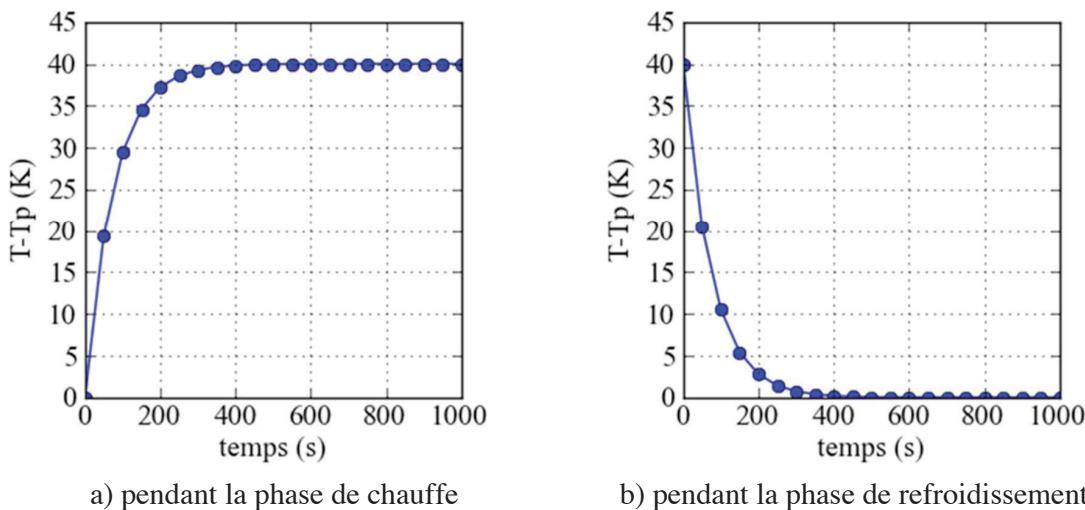


Figure 2 – Évolution de la température du liquide dans le réacteur

On exploite d'abord la courbe obtenue lors de la phase de chauffe (**figure 2a**) pour déterminer le coefficient de transfert de chaleur à la paroi, noté U (unité : $\text{W} \cdot \text{m}^{-2} \cdot \text{K}^{-1}$). Ce coefficient rend compte des échanges de chaleur entre la phase réactionnelle et le fluide caloporteur dans la double enveloppe à travers la paroi du réacteur. Dans le cas présent, la température T_p étant la température de paroi du côté du fluide caloporteur, il s'agit d'un coefficient de transfert thermique global qui tient compte du transfert convectif côté réaction et du transfert par conduction dans la paroi qui sépare les deux fluides.

Pour obtenir la valeur de U , on commence par établir l'équation différentielle qui régit l'évolution de la température T en fonction du temps en réalisant un bilan d'énergie.

Le bilan d'énergie, conséquence directe du premier principe de la thermodynamique, appliqué au système constitué par la phase réactionnelle lors de la phase de chauffe, conduit à l'équation différentielle suivante (équation (1))

$$(\rho \times V \times Cp) \frac{dT}{dt} = P_{th} - U \times A \times (T - T_p), \quad (1)$$

où T est la température du fluide à l'intérieur du réacteur, ρ , V et C_p sont respectivement la masse volumique, le volume et la capacité thermique massique du fluide, P_{th} est la puissance thermique cédée par la résistance chauffante au milieu réactionnel, T_p est la température à la paroi, maintenue constante ($T_p = 320,0$ K) et A la surface latérale du réacteur sur laquelle le fluide à l'intérieur du réacteur est en contact avec la double enveloppe.

Q1. Interpréter concrètement chacun des trois termes du bilan d'énergie en précisant leur signification physique et vérifier qu'ils sont homogènes à des puissances.

Q2. Donner l'expression de $T - T_p$ en régime permanent. Il est rappelé que la température de la paroi, T_p , est maintenue constante tout au long des essais. Il est précisé que la température de la phase liquide à l'instant initial est égale à T_p .

Q3. D'après les résultats obtenus lors de la première expérience (**figure 2a**), donner la valeur de la différence de température $T - T_p$ lorsqu'on atteint le régime permanent. Calculer la valeur du coefficient de transfert de chaleur à la paroi dans les unités SI. On donne $T_p = 320,0$ K, $\rho = 1\ 000,0\ \text{kg}\cdot\text{m}^{-3}$, $V = 0,1 \times 10^{-3}\ \text{m}^3$ et $C_p = 1\ 800,0\ \text{J}\cdot\text{kg}^{-1}\cdot\text{K}^{-1}$, $P_{th} = 96,0\ \text{W}$ et $A = 8,0 \times 10^{-3}\ \text{m}^2$.

Q4. On souhaite faire apparaître un temps caractéristique d'échange thermique τ_c du système. Montrer que le bilan d'énergie peut se mettre sous la forme suivante (équation (2)) :

$$\frac{dT}{dt} = s + \frac{T_p - T}{\tau_c}. \quad (2)$$

Donner les expressions de s et τ_c . Vérifier que τ_c est homogène à un temps.

On souhaite maintenant exploiter les résultats obtenus lors de la phase de refroidissement (**figure 2b**) pour confirmer la valeur du temps caractéristique d'échange thermique déterminé précédemment.

Q5. Le bilan d'énergie établi à la question **Q4** est-il modifié ? Si oui, donner la nouvelle expression de $\frac{dT}{dt}$.

Q6. Donner l'expression de $T - T_p$ en fonction du temps t par résolution de l'équation différentielle. On notera T_{max} la température initiale lors de la phase de refroidissement.

Q7. Le tracé de $\ln(T - T_p)$ (avec $T - T_p$ en K) en fonction du temps t (**figure 3**) donne une droite d'équation $y = -1,33 \times 10^{-2} \times x + 3,68$ (avec x en secondes). En déduire la valeur du temps caractéristique d'échange thermique τ_c . Calculer la valeur du coefficient de transfert de chaleur à la paroi et vérifier que cette valeur correspond à celle obtenue avec la première expérience.

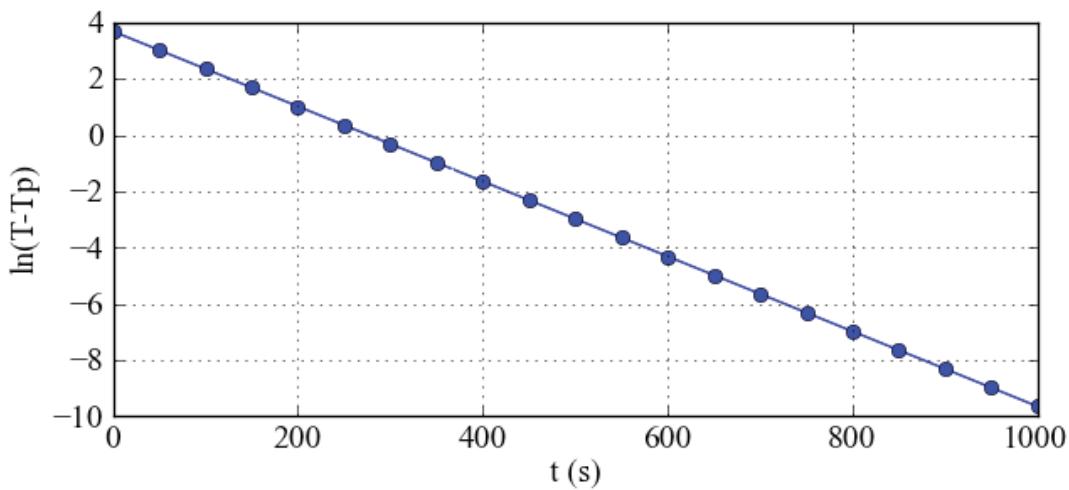


Figure 3 – Tracé de $\ln(T - T_p)$ en fonction de t à l'aide des points enregistrés lors de la phase de refroidissement (**figure 2b**)

I.2 – Etude d'une réaction exothermique en réacteur fermé à double enveloppe

Dans cette sous-partie, on considère que l'on met en œuvre une réaction chimique exothermique $R \rightarrow \text{produits}$ (R est dissous dans un solvant) dans le même réacteur que celui dont on a étudié les échanges thermiques dans la sous-partie précédente. Il s'agit ici de caractériser le comportement thermique du réacteur en présence d'une réaction exothermique.

Le comportement du réacteur fermé parfaitement agité avec double enveloppe peut être représenté par un système constitué de deux équations différentielles ordinaires. La première équation différentielle ordinaire représente l'évolution temporelle de la conversion du réactif R ; la deuxième permet de caractériser l'évolution de la température de la réaction en fonction du temps.

Le réactif R étant dissout dans un solvant, on considère que le volume du mélange réactionnel V reste constant au cours du temps. On considère également que la réaction est homogène et qu'elle a lieu dans tout le volume du mélange réactionnel.

On commence par déterminer l'équation différentielle qui régit l'évolution de la conversion du réactif R en fonction du temps.

Q8. On considère que la réaction est d'ordre 1 par rapport au réactif R . Donner l'expression de la vitesse de la réaction (exprimée par unité de volume de mélange réactionnel) que l'on notera r (on notera C_R la concentration molaire du réactif R et k la constante cinétique). Préciser la dimension et l'unité de r dans le Système International.

Q9. Rappeler la loi d'Arrhenius donnant les variations de la constante de réaction en fonction de la température. On notera k_0 le facteur pré-exponentiel et E_a l'énergie d'activation. Préciser les dimensions et les unités SI de k , k_0 et E_a .

Q10. Écrire le bilan de matière sous la forme $\frac{dC_R}{dt} = f(C_R, T)$. Préciser l'expression de $f(C_R, T)$. Il est rappelé que le volume de la phase réactionnelle reste constant au cours du temps.

Q11. Dans le cas où le réacteur fonctionnerait en marche isotherme, résoudre l'équation différentielle et donner l'expression de la concentration de R en fonction du temps sous la forme $C_R = g(t)$. On notera C_R^0 la concentration initiale en R .

Q12. Pour simplifier les bilans, on introduit le taux de conversion de R , noté X_R , défini par la relation suivante : $X_R = (C_R^0 - C_R)/C_R^0$. Exprimer l'évolution de taux de conversion X_R en fonction du temps pour le cas de la marche isotherme.

Q13. Donner l'expression de l'équation différentielle qui régit l'évolution du taux de conversion X_R en fonction du temps dans le cas général (marche quelconque, c'est-à-dire non isotherme), sans chercher à la résoudre.

L'évolution de la température en fonction du temps est régie par une seconde équation différentielle obtenue en réalisant un bilan d'énergie sur le réacteur, conséquence directe du premier principe de la thermodynamique.

La réaction chimique qui se déroule dans le réacteur produit par unité de temps une variation de l'enthalpie du système réactionnel donnée par $S(t, X, T)$ (équation (3)) qui correspond à la chaleur dégagée par la réaction. Ce paramètre est appelé « terme source » dans la suite.

$$S(t, X_R, T) = -\Delta_r H^0(T) \times r(t, X_R, T) \times V, \quad (3)$$

où V est le volume du mélange réactionnel, r est la vitesse de la réaction et $\Delta_r H^0(T)$ est l'enthalpie molaire standard de la réaction. Dans la suite, on suppose que $\Delta_r H^0(T)$ ne dépend pas de la température. On prendra $\Delta_r H^0(T) = \Delta_r H^0(T_p)$ que l'on notera $\Delta_r H^0$ pour simplifier.

Q14. Donner la dimension du terme source $S(t, X_R, T)$.

Q15. Montrer qu'un bilan enthalpique appliqué à un système que l'on précisera avec soin permet d'aboutir à la relation suivante (équation (4))

$$\frac{dT}{dt} = J \frac{dX_R}{dt} - \frac{T - T_p}{\tau_c}, \quad (4)$$

où l'on exprimera J et τ_c en fonction de $\Delta_r H$, C_R^0 , ρ , C_p , V , U et A . On admettra qu'il est légitime de négliger la contribution des réactifs, des produits et des accessoires situés à l'intérieur du réacteur au travers de leur capacités thermiques devant celle du solvant.

Q16. Donner l'expression du paramètre J ainsi que sa dimension.

Q17. Calculer la valeur du paramètre J en unité SI. On donne $\Delta_r H^0 = -360,0 \text{ kJ}\cdot\text{mol}^{-1}$, $\rho = 1\,000,0 \text{ kg}\cdot\text{m}^{-3}$, $C_p = 1\,800,0 \text{ J}\cdot\text{kg}^{-1}\cdot\text{K}^{-1}$ et $C_R^0 = 500,0 \text{ mol}\cdot\text{m}^{-3}$.

I.3 – Stabilité thermique du réacteur

Une première condition de stabilité, valable pour le cas d'une marche adiabatique, est que la température finale T_f de la réaction soit inférieure à une température maximale T_{max} , telle que $T_{max} = 1,25 \times T_p$.

Q18. Exprimer l'équation différentielle (équation (4)) dans le cas d'un fonctionnement adiabatique.

Q19. Déterminer alors l'expression de la température T en fonction du taux de conversion X_R , du paramètre J et de T_0 la température initiale du mélange réactionnel.

Q20. Donner l'expression de la température T_f atteinte en fin de réaction dans le cas d'une marche adiabatique sachant que $T_0 = T_p = 320,0 \text{ K}$. Conclure quant à la stabilité du réacteur dans le cas de cette étude. Donner la signification physique du paramètre J .

Partie II – Traitement numérique des données expérimentales

Les algorithmes demandés au candidat **devront être réalisés dans le langage Python**. On supposera les bibliothèques « numpy » et « matplotlib.pyplot » chargées. Une **annexe** présentant les fonctions usuelles de Python est disponible pages 15 à 18. Les commentaires suffisants à la compréhension du programme devront être apportés et des noms de variables explicites devront être utilisés lorsque ceux-ci ne sont pas imposés.

II.1 – Détermination des paramètres d'un modèle par régression linéaire

Pour calculer la valeur du temps caractéristique d'échange thermique du réacteur à la question **Q7**, un modèle de régression linéaire simple a été estimé à partir des points expérimentaux enregistrés lors de la phase de refroidissement.

Le modèle de régression linéaire simple (fonction affine) est un modèle de régression d'une variable expliquée ($\ln(T - T_p)$ dans notre cas) sur une variable explicative (le temps t dans notre cas) dans lequel on fait l'hypothèse que la fonction qui relie la variable explicative à la variable expliquée est linéaire dans ses paramètres.

Soit n le nombre de points expérimentaux. Le modèle linéaire simple s'écrit de la manière suivante pour un point i ($1 \leq i \leq n$)

$$y_i = \widehat{\beta}_1 \times x_i + \widehat{\beta}_0, \quad (5)$$

où $\widehat{\beta}_0$ et $\widehat{\beta}_1$ sont les paramètres du modèle, y_i est la variable expliquée et x_i est la variable explicative.

On propose de déterminer les paramètres du modèle par deux méthodes directes.

La méthode consiste à écrire le modèle (équation (5)) sous la forme matricielle $Y = L \times \widehat{B}$. \widehat{B} est un vecteur colonne contenant les paramètres du modèle $\widehat{\beta}_0$ et $\widehat{\beta}_1$, Y est un vecteur colonne contenant les n valeurs y_i et L une matrice à n lignes et 2 colonnes, telle que $L(i, 1) = \frac{\partial y_i}{\partial \widehat{\beta}_0}$ et $L(i, 2) = \frac{\partial y_i}{\partial \widehat{\beta}_1}$. Rappelons que $\widehat{B} = (L^t \times L)^{-1} \times L^t Y$ où L^t est la matrice transposée de L .

Q21. Donner les expressions de $\frac{\partial y_i}{\partial \widehat{\beta}_0}$ et $\frac{\partial y_i}{\partial \widehat{\beta}_1}$. En déduire la valeur des coefficients de la matrice L .

Q22. On suppose que les vecteurs colonnes Y et X , qui contiennent les valeurs y_i et x_i ($1 \leq i \leq n$), sont déjà créés. Donner le code permettant de créer la matrice L .

Q23. Donner le code permettant de déterminer les coefficients de la matrice $P = L^t \times L$. Préciser les dimensions de la matrice P .

Q24. Donner le code permettant de déterminer les coefficients de la matrice $Q = L^t \times Y$. Préciser les dimensions de la matrice Q .

Q25. Donner le code permettant de créer une fonction **inv_mat(M)** qui renvoie la matrice inverse de la matrice M de dimension (2×2) donnée comme argument d'entrée.

Q26. On note N la matrice inverse de M . Donner le code permettant de déterminer les coefficients $\widehat{\beta}_0$ et $\widehat{\beta}_1$ de la matrice \widehat{B} .

II.2 – Prédiction du comportement thermique du réacteur

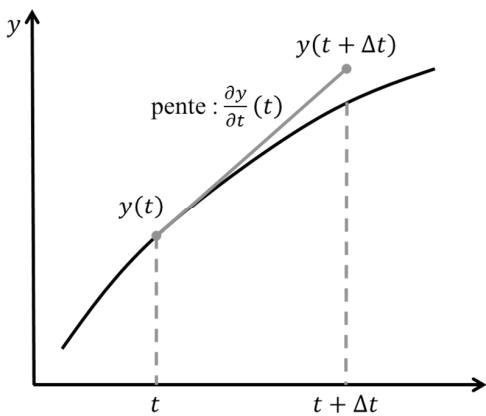
Dans cette sous-partie, on souhaite utiliser le modèle constitué du système d'équations différentielles établies dans la **Partie I** qui décrit l'évolution du taux de conversion du réactif X_R et de la température de réaction T en fonction du temps pour prédire le comportement thermique du réacteur en présence d'une réaction exothermique.

Pour simplifier les notations, on met le système d'équations différentielles sous la forme suivante (équation (6)) :

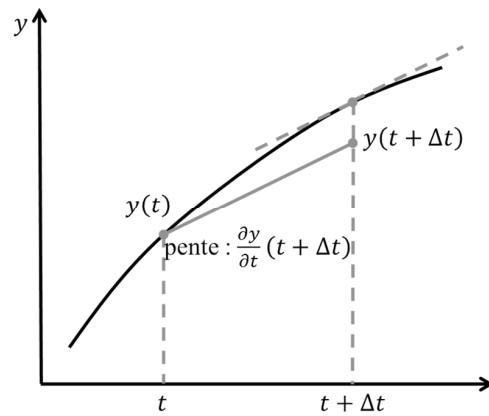
$$\begin{cases} \frac{dX_R}{dt} = f_1(t, X_R, T), \\ \frac{dT}{dt} = f_2(t, X_R, T). \end{cases} \quad (6)$$

La méthode de résolution proposée pour résoudre le système d'équations différentielles est la méthode d'Euler implicite à pas fixe. Cette méthode est préférée car elle donne de meilleurs résultats que la méthode explicite pour les systèmes dits raides (un système raide est un système qui est caractérisé par une évolution rapide des phénomènes en fonction du temps, ce qui est le cas ici pour la température de réaction).

Soit une variable y qui dépend du temps t . Comme la méthode d'Euler explicite, la méthode d'Euler implicite consiste à évaluer la valeur de $y(t + \Delta t)$ à partir de celle de $y(t)$ et de la dérivée $\frac{\partial y}{\partial t}$ (**figure 4**, page suivante). La différence entre les deux méthodes réside dans le choix de l'abscisse à laquelle est évaluée la dérivée $\frac{\partial y}{\partial t}$. Dans le cas de la méthode explicite, elle est évaluée en $t : \frac{\partial y}{\partial t}(t)$ comme le montre le schéma de la **figure 4a**, page suivante. Pour la méthode implicite, elle est évaluée en $t + \Delta t : \frac{\partial y}{\partial t}(t + \Delta t)$ (**figure 4b**, page suivante).



a) cas de la méthode explicite



b) cas de la méthode implicite

Figure 4 – Approximation de $y(t + \Delta t)$ par la méthode d'Euler

Dans le cas d'un schéma implicite (**figure 4b**), l'expression de $y(t + \Delta t)$ en fonction de $y(t)$ et de la dérivée $\frac{\partial y}{\partial t}$ ($t + \Delta t$) évaluée en $t + \Delta t$ est obtenue en réalisant un développement limité dit rétrograde : $y(t) = y(t + \Delta t) - \Delta t \times \frac{\partial y}{\partial t}(t + \Delta t) + o(\Delta t)$.

Q27. À l'aide d'un développement limité rétrograde de la fonction $X_R(t)$, donner l'expression de $X_R(t + \Delta t)$ à l'ordre 1 en fonction de $X_R(t)$ et de sa dérivée partielle par rapport à t , $\frac{dX_R}{dt}(t + \Delta t)$ évaluée en $t + \Delta t$.

Q28. En déduire une valeur approchée de $\frac{dX_R}{dt}(t + \Delta t)$ à l'ordre 0 en fonction de $X_R(t)$, $X_R(t + \Delta t)$ et Δt .

Q29. À l'aide d'un développement limité rétrograde de la fonction $T(t)$, donner l'expression de $T(t + \Delta t)$ à l'ordre 1 en fonction de $T(t)$ et de sa dérivée partielle par rapport à t , $\frac{dT}{dt}(t + \Delta t)$.

Q30. En déduire une valeur approchée de $\frac{dT}{dt}(t + \Delta t)$ à l'ordre 0 en fonction de $T(t)$, $T(t + \Delta t)$ et Δt .

On procède à la discrétisation des équations. On note X_{Ri} la conversion évaluée au temps t_i , $X_{R,i+1}$ la conversion évaluée au temps t_{i+1} et $\left. \frac{dX_R}{dt} \right|_{t_{i+1}}$ la dérivée de X_R évaluée à l'instant t_{i+1} . De même, on note T_i la température évaluée au temps t_i , T_{i+1} la température évaluée au temps t_{i+1} et $\left. \frac{dT}{dt} \right|_{t_{i+1}}$ la dérivée de T évaluée à l'instant t_{i+1} .

Q31. Donner l'expression de $\frac{dX_R}{dt}\Big|_{t_{i+1}}$ en fonction de X_{Ri} , X_{Ri+1} et Δt .

Q32. Donner l'expression approchée de X_{Ri+1} en fonction de X_{Ri} , Δt et de la fonction $f_1(t_{i+1}, X_{Ri+1}, T_{i+1})$, évaluée en t_{i+1} .

Q33. Donner l'expression de $\frac{dT}{dt}\Big|_{t_{i+1}}$ en fonction de T_i , T_{i+1} et Δt .

Q34. Donner l'expression approchée de T_{i+1} en fonction de T_i , Δt et de la fonction $f_2(t_{i+1}, X_{Ri+1}, T_{i+1})$, évaluée en t_{i+1} .

On constate que les expressions obtenues aux questions **Q32** et **Q34** constituent un système non linéaire dont les inconnues sont X_{Ri+1} et T_{i+1} . On propose d'utiliser la méthode de Newton-Raphson pour trouver les valeurs de X_{Ri+1} et T_{i+1} à chaque itération de la méthode d'Euler.

La méthode de Newton-Raphson pour la résolution d'un système de n équations non linéaires à n inconnues $x = (x_1, \dots, x_n)$, mis sous la forme de l'équation (7) suivante,

$$g(x) = \begin{bmatrix} g_1(x_1, \dots, x_n) \\ \vdots \\ g_n(x_1, \dots, x_n) \end{bmatrix} = 0, \quad (7)$$

est une extension de la méthode de Newton permettant de trouver la racine d'une fonction d'une variable.

On peut démontrer la formule de récurrence suivante (équation (8)) :

$$x^{j+1} = x^j - (Dg(x^j))^{-1} g(x^j), \quad (8)$$

où x^{j+1} est la valeur du vecteur x à l'itération $j + 1$, x^j est la valeur du vecteur x à l'itération j , $g(x^j)$ est la valeur de $g(x)$ à l'itération j et $Dg(x^j)$ est la matrice Jacobienne évaluée en x^j (équation (9)) :

$$Dg(x^j) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix}_{x=x^j}. \quad (9)$$

La formule de récurrence s'accompagne du choix d'une valeur initiale, notée x^0 , et d'un critère d'arrêt, par exemple $\|x^{j+1} - x^j\| \leq \varepsilon$.

Q35. Transformer les expressions obtenues aux questions **Q32** et **Q34** pour les mettre sous la forme $g_1(X_{Ri+1}, T_{i+1}) = 0$ et $g_2(X_{Ri+1}, T_{i+1}) = 0$.

Q36. Donner les expressions de $\frac{\partial g_1}{\partial X_{Ri+1}}$, $\frac{\partial g_1}{\partial T_{i+1}}$, $\frac{\partial g_2}{\partial X_{Ri+1}}$ et $\frac{\partial g_2}{\partial T_{i+1}}$ permettant de construire la matrice Jacobienne $Dg(X_{Ri+1}, T_{i+1})$.

Q37. Écrire une fonction **mat_Dg(x)** qui a pour argument d'entrée un vecteur x contenant les valeurs de X_{Ri+1} et T_{i+1} à l'itération j et qui retourne la matrice Jacobienne $Dg(x^j)$. On supposera que les paramètres suivants ont été au préalable déclarés comme variables globales : $\Delta t = 0,01$ s, $k_0 = 5,0$ s $^{-1}$, $E_a = 20\,000,0$ J.mol $^{-1}$, $J = 100,0$ K, $\tau_c = 75$ s et $R = 8,314$ J.K $^{-1}.\text{mol}^{-1}$.

Q38. Écrire le code permettant de calculer les valeurs du vecteur x à l'itération $j + 1$ à l'aide de l'équation (8) lors d'une boucle de l'algorithme de Newton-Raphson. On notera **x_old** la valeur de x à l'itération j et **x_new** la valeur de x à l'itération $j + 1$. De même, on notera **x_euleri** et **T_euleri** les vecteurs contenant les valeurs de X_{Ri} et T_i à l'itération i de la méthode d'Euler implicite. Pour l'inversion de matrice, on utilisera la fonction **inv_mat(M)** écrite à la question **Q25**.

Q39. Pour obtenir la valeur de **x_new** par la méthode de Newton-Raphson, on souhaite créer une boucle itérative avec condition. La condition d'arrêt porte sur la valeur absolue de la différence des températures T_{i+1}^j et T_{i+1}^{j+1} que l'on souhaite inférieure à 10^{-5} K. Pour les valeurs initiales de X_{ri+1}^0 et T_{i+1}^0 on prendra respectivement 0,5 et $T_p + J/2$. Écrire le code correspondant. Il est inutile de recopier l'intégralité du code écrit à la question précédente ; on indiquera néanmoins sa place dans le code de cette question.

Maintenant que le code permettant de trouver les valeurs de X_{Ri+1} et T_{i+1} par la méthode de Newton-Raphson lors d'une itération de la méthode d'Euler implicite a été établi, on souhaite calculer les valeurs pour l'ensemble des itérations de la méthode d'Euler implicite. On rappelle que $X_R(t = 0) = 0$ et $T(t = 0) = T_p = 320,0$ K. En plus de noter **x_euleri** et **T_euleri** les vecteurs contenant les valeurs de X_{Ri} et T_i pour chaque itération i de la méthode d'Euler implicite, on notera **t_euleri** le vecteur contenant les valeurs de t_i à chaque itération. L'intégration sera réalisée sur l'intervalle $[t_0, t_f]$ avec $t_0 = 0$ s et $t_f = 1\,000$ s. Le pas de temps Δt est de 0,01 s.

Q40. Donner le code permettant de calculer le nombre m d'intervalles Δt compris dans l'intervalle $[t_0, t_f]$ (m est un entier).

Q41. Écrire le code permettant de calculer les valeurs des éléments des vecteurs **x_euleri**, **T_euleri** et **t_euleri** à chaque itération de la méthode d'Euler implicite.

Q42. Donner le code permettant de tracer les graphes de la **figure 5** montrant l'évolution de la conversion et de la température en fonction du temps que l'on obtiendrait en réalisant la résolution numérique du système d'équations différentielles (simulation réalisée avec $k_0 = 5,0 \text{ s}^{-1}$, $E_a = 20\,000,0 \text{ J}\cdot\text{mol}^{-1}$, $J = 100,0 \text{ K}$, $\tau_c = 75 \text{ s}$).

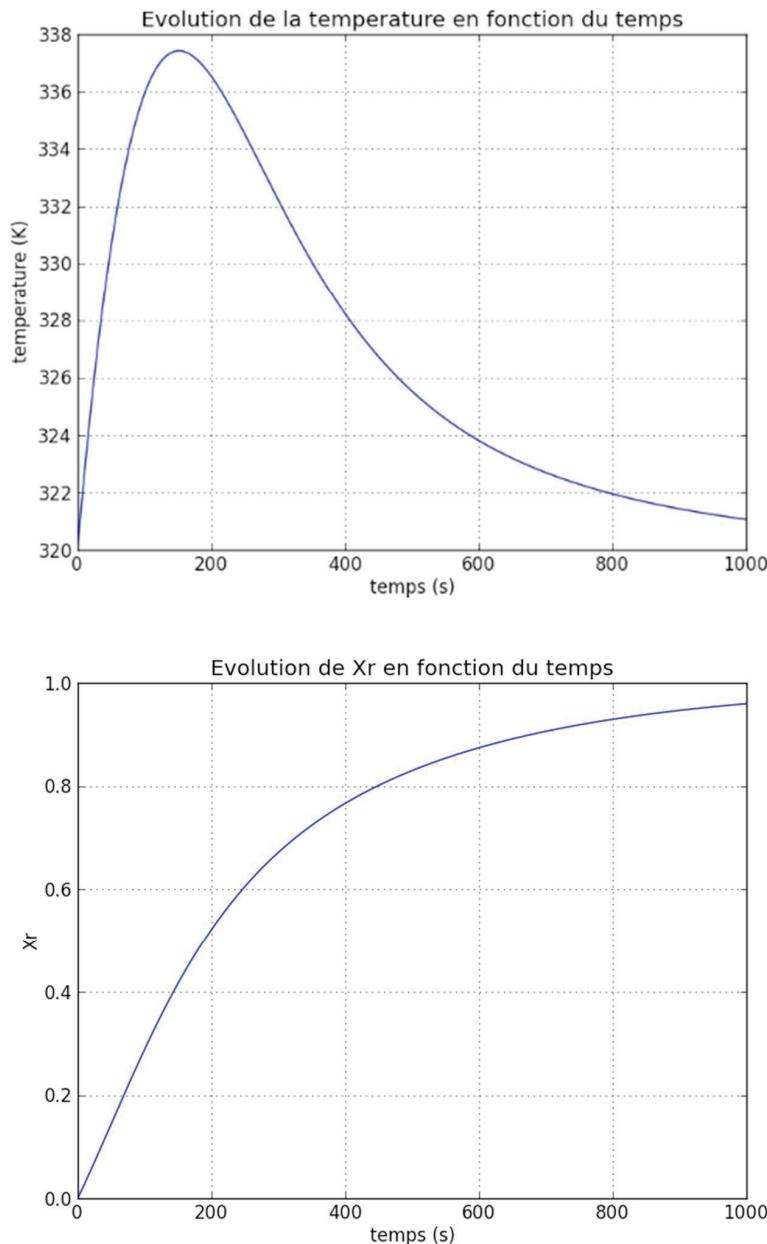


Figure 5 – Température et conversion calculées à partir du modèle constitué du système d'équations différentielles déterminées dans la **Partie I**

FIN

Annexe : Fonctions de Python

1. Bibliothèque numpy de Python

Dans les exemples ci-dessous, la bibliothèque numpy a préalablement été importée à l'aide de la commande : **import numpy as np**

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

np.array(liste)

Description : fonction permettant de créer une matrice (de type tableau) à partir d'une liste.

Argument d'entrée : une liste définissant un tableau à 1 dimension (vecteur) ou 2 dimensions (matrice).

Argument de sortie : un tableau (matrice).

Exemples : `np.array([4,3,2])`
 ⇒ [4 3 2]

`np.array([[5],[7],[1]])`
 ⇒ [[5]
 [7]
 [1]]]

`np.array([[3,4,10],[1,8,7]])`
 ⇒ [[3 4 10]
 [1 8 7]]]

$A[i,j]$.

Description : fonction qui retourne l'élément $(i + 1, j + 1)$ de la matrice A . Pour accéder à l'intégralité de la ligne $i+1$ de la matrice A , on écrit $A[i,:]$. De même, pour obtenir toute la colonne $j+1$ de la matrice A , on utilise la syntaxe $A[:,j]$.

Arguments d'entrée : une liste contenant les coordonnées de l'élément dans le tableau A .

Argument de sortie : l'élément $(i + 1, j + 1)$ de la matrice A .

ATTENTION : en langage Python, les lignes d'un tableau A de dimension $n \times m$ sont numérotées de 0 à $n - 1$ et les colonnes sont numérotées de 0 à $m - 1$

Exemple : `A=np.array([[3,4,10],[1,8,7]])`

`A[0,2]`
 ⇒ 10

`A[1,:]`
 ⇒ [1 8 7]

`A[:,2]`
 ⇒ [10 7]

np.zeros((n,m))

Description : fonction créant une matrice (tableau) de dimensions $n \times m$ dont tous les éléments sont nuls.

Arguments d'entrée : un tuple de deux entiers correspondant aux dimensions de la matrice à créer.

Argument de sortie : un tableau (matrice) d'éléments nuls.

Exemple : np.zeros((3,4))

$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

np.linspace(Min,Max,nbElements)

Description : fonction créant un vecteur (tableau) de $nbElements$ nombres espacés régulièrement entre Min et Max . Le 1^{er} élément est égal à Min , le dernier est égal à Max et les éléments sont espacés de $(Max - Min)/(nbElements - 1)$:

Arguments d'entrée : un tuple de 3 entiers.

Argument de sortie : un tableau (vecteur).

Exemple : np.linspace(3,25,5)

$$\Rightarrow [3 \ 8.5 \ 14 \ 19.5 \ 25]$$

np.loadtxt('nom_fichier',delimiter='string',usecols=[n])

Description : fonction permettant de lire les données sous forme de matrice dans un fichier texte et de les stocker sous forme de vecteurs.

Arguments d'entrée : le nom du fichier qui contient les données à charger, le type de caractère utilisé dans ce fichier pour séparer les données (par exemple un espace ou une virgule) et le numéro de la colonne à charger (ATTENTION, la première colonne porte le numéro 0).

Argument de sortie : un tableau.

Exemple : data=np.loadtxt('fichier.txt',delimiter=' ',usecols=[0])
#dans cet exemple data est un vecteur qui correspond à la première #colonne de la matrice contenue dans le fichier 'fichier.txt'.

2. Bibliothèque matplotlib.pyplot de Python

Cette bibliothèque permet de tracer des graphiques. Dans les exemples ci-dessous, la bibliothèque matplotlib.pyplot a préalablement été importée à l'aide de la commande :

import matplotlib.pyplot as plt

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

plt.plot(x,y)

Description : fonction permettant de tracer un graphique de n points dont les abscisses sont contenues dans le vecteur x et les ordonnées dans le vecteur y . Cette fonction doit être suivie de la fonction **plt.show()** pour que le graphique soit affiché.

Arguments d'entrée : un vecteur d'abscisses x (tableau de dimension n) et un vecteur d'ordonnées y (tableau de dimension n).

Argument de sortie : un graphique.

Exemple :

```
x= np.linspace(3,25,5)
y=sin(x)
plt.plot(x,y)
plt.title('titre_graphique')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

plt.title('titre')

Description : fonction permettant d'afficher le titre d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.xlabel('nom')

Description : fonction permettant d'afficher le contenu de nom en abscisse d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.ylabel('nom')

Description : fonction permettant d'afficher le contenu de nom en ordonnée d'un graphique.

Argument d'entrée : une chaîne de caractères.

plt.show()

Description : fonction réalisant l'affichage d'un graphe préalablement créé par la commande **plt.plot(x,y)**. Elle doit être appelée après la fonction **plt.plot** et après les fonctions **plt.xlabel** et **plt.ylabel**.

3. Fonction intrinsèque de Python

sum(x)

Description : fonction permettant de faire la somme des éléments d'un vecteur ou tableau.

Arguments d'entrée : un vecteur ou un tableau de réel, entier ou complexe.

Argument de sortie : un scalaire y qui est la somme des éléments de x.

Exemple : $y = \text{sum}(x)$

//y retourne la somme des éléments de x.

ÉCOLE NATIONALE DE L'AVIATION CIVILE

ICNA 2018

AVERTISSEMENTS

Les scripts et fonctions Python présentés dans les énoncés sont rédigés en Python 3.

Les questions 10, 11, 12 et 13 sont liées.

Question 1 :

En informatique, le codage des caractères repose sur l'association d'un caractère à un nombre.
Plusieurs codes existent. Trouver le ou les intrus.

- A) ISO-8859-1
- B) UTF-8
- C) UCI
- D) ASCII

Question 2 :

Qu'est-ce qu'un logiciel libre ?

- A) Un logiciel dont son brevet est arrivé à expiration.
- B) Un logiciel qui est gratuit.
- C) Un logiciel qui n'a aucune restriction temporelle d'exécution.
- D) Un logiciel qui peut être utilisé, copié, modifié, et redistribué librement.

Question 3 :

Quelle est la complexité temporelle du tri par fusion pour une entrée de taille **n** ?

- A) $O(n^2)$
- B) $O(n)$
- C) $O(\log(n))$
- D) $O(n^{2^*}\log(n))$

Question 4 :

Quelle variable faut-il afficher avec le programme python suivant pour produire la chaîne de caractères : Air France/Toulouse-Blagnac (LFBO)

```
aero      = "/Toulouse"
compagnie = "-Air France"
oaci      = " (LFBO)-"
ville     = "-Blagnac"

res1 = (compagnie+aero+ville+oaci).replace("-", "", 1)
res2 = (compagnie+aero+ville+oaci).replace("-", "", 2)
res3 = res1[1:].replace("-", "")
res4 = (compagnie+aero+ville+oaci)[1:]
```

- A) res1
- B) res2
- C) res3
- D) res4

Question 5 :

Qu'affiche le programme python suivant quand on l'invoque via la ligne de commandes :
 python icna5.py

```
# Fichier : icna5.py
n = 0
if __name__ == '__main__':
    for i in range(5):
        n = n + 4
else:
    for n in range(3, 6):
        n = n + 4
print(n)
```

- A) 9
- B) 10
- C) 20
- D) 24

Question 6 :

On considère la méthode de Newton qui est un algorithme efficace pour trouver numériquement une approximation précise d'une racine d'une fonction réelle d'une variable réelle.

- A) L'approximation d'un zéro de la fonction s'opère grâce à un développement de Taylor au premier ordre.
- B) Seul le signe de la fonction est utilisé et cela mène à une convergence linéaire.
- C) La fonction dérivée n'a pas besoin d'être calculée.
- D) Toute mise en œuvre de la méthode doit inclure un contrôle du nombre d'itérations.

Question 7 :

Considérant le programme Python suivant qui vise à approximer une racine d'une fonction réelle d'une variable réelle :

```
def MethodeNewton(f, pas, x):
    for i in range(0, pas):
        x1 = x - f(x) / (3*x**2-(3/2))
        x = x1
    return x

def f(x):
    return x**3-(3/2)*x-2

x0 = 8
pas = 10
x = MethodeNewton(f, pas, x0)

print("x      = %0.6f" % x)
print("f(x) = %0.6f" % f(x))
```

- A) Le nombre d'itérations nécessaires à un résultat probant dépend de la valeur de la variable x_0 .
- B) Il manque l'incrémentation du pas à chaque itération.
- C) Le résultat $f(x)$ sera très proche de zéro.
- D) Le résultat $f(x)$ sera très proche de $(f(x_0)-f(x))/\text{pas}$.

Question 8 :

Considérant le programme Python suivant qui opère un traitement itératif pour calculer une aire

```
def Rectangle(f, xA, xB, intervalles):
    resultat, i = 0, 0
    base = (xB-xA)/intervalles
    while i < intervalles:
        resultat = resultat + f(xA+i*base)*base
        i = i + 1
    return resultat

def f(x):
    return x**3-(3/2)*x-2

xA = 0.0
xB = 1.0
intervalles = 100
resultat = Rectangle(f, xA, xB, intervalles)
print("resultat = %0.6f" % resultat)
```

- A) Le nombre d'itérations nécessaires à un résultat probant dépend de la valeur de la variable `intervalles`.
- B) Il manque la prise en compte de la dernière itération.
- C) Le résultat sera très proche de $-5/2$.
- D) Le résultat sera très proche de $5/2$.

Question 9 :

Considérant le programme Python suivant qui opère un traitement itératif pour calculer une aire

```
def Median(f, xA, xB, intervalles):
    i, resultat = 0, 0
    base = (xB-xA)/intervalles
    while i < intervalles:
        resultat = resultat + f((2*xA+(2*i+1)*base)/2)*base
        i = i + 1
    return resultat

def f(x):
    return x**3-(3/2)*x-2

xA = 0.0
xB = 1.0
intervalles = 100
resultat = Median(f, xA, xB, intervalles)

print("resultat = %0.6f" % resultat)
```

- A) Le nombre d'itérations nécessaires à un résultat probant ne dépend pas de la valeur de la variable `intervalles`.
- B) Le fait de considérer des rectangles construits entre `xA` et `xB` plutôt que de raisonner sur un point médian entre chaque intervalle était moins précis.
- C) Il ne manque pas la prise en compte de la dernière itération.
- D) Le résultat sera très proche de $5/2$.

Soit le programme `bissextile.py` suivant qui détermine si une année saisie au clavier est une année bissextile ou non. L'objectif est de tester 3 codes différents.

Rappel algorithmique : Depuis l'ajustement du calendrier grégorien, l'année sera bissextile si elle est divisible par 4 et non divisible par 100, ou si elle est divisible par 400.

```
# -*- coding: utf-8 -*-
import os

print("Cette année est-elle bissextile ?")
annee = -1
while annee <= 0:
    annee = input("Entrez une année : ")
    try:
        annee = int(annee) # ligne a
        assert annee > 0 # ligne b
    except ValueError:
        print("Erreur de saisie\n")
        annee = -1 # ligne c
    except AssertionError:
        print("Erreur de saisie\n")
        annee = -1

# _____ Code n°1 _____
if (annee % 4 == 0) ^ (annee % 100 == 0) ^ (annee % 400 == 0):
    print("C'est une année bissextile")
else:
    print("C'est une année non bissextile")
# _____ Code n°2 _____
bissextile = False
if annee % 400 == 0:
    bissextile = True
elif annee % 100 == 0:
    bissextile = False
elif annee % 4 == 0:
    bissextile = True

if bissextile:
    print("C'est une année bissextile")
else:
    print("C'est une année non bissextile")

# _____ Code n°3 _____ # ligne d
print("C'est une année bissextile")
else:
    print("C'est une année non bissextile")

print ("Fin du programme")
os.system("pause")
```

Question 10 :

Quelles sont les assertions vraies à propos des instructions du début du programme `bissextile.py` ?

- A) L'instruction de la ligne a est parfaitement inutile.
- B) L'instruction de la ligne b permet de lever une exception.
- C) `except ValueError:` permet de détecter la saisie d'un nombre négatif.
- D) L'instruction de la ligne c est parfaitement inutile.

Question 11 :

Le **code n°1** affiche pour l'année :

- A) 1900 C'est une année bissextile
- B) 2018 C'est une année non bissextile
- C) 2000 C'est une année bissextile
- D) 1802 C'est une année bissextile

Question 12 :

Le **code n°2** est-il correct ?

- A) Oui
- B) Non car la variable `bissextile` est mal initialisée.
- C) Non car il manque une partie `else:` pour finir la première instruction conditionnelle.
- D) Non car `if bissextile:` provoque une erreur d'exécution.

Question 13 :

Pour un bon fonctionnement du **code n°3**, il faut écrire en ligne d :

- A) `if annee % 4 == 0 and (annee % 400 == 0 or annee % 100 != 0):`
- B) `if annee % 400 == 0 or (annee % 4 == 0 and annee % 100 != 0):`
- C) `if annee % 4 != 0 and (annee % 400 != 0 or annee % 100 == 0):`
- D) `if annee % 400 != 0 or (annee % 4 == 0 and annee % 100 != 0):`

Question 14 :

Soit le programme suivant :

```
import numpy as np

def matrice(n):
    A = np.zeros((n,n))
    for i in range(1,n+1):
        A[i-1,i-1] = 1
        A[i-1,0] = i
        A[0,i-1] = i
        A[n-1,i-1] = n+1-i
        A[i-1,n-1] = n+1-i
    return A

print(matrice(4))
```

L'exécution de ce programme affiche :

A)
[[1. 1. 3. 3.]
 [1. 1. 0. 3.]
 [2. 0. 1. 2.]
 [2. 2. 2. 1.]]

B)
[[0. 2. 3. 4.]
 [2. 1. 1. 3.]
 [3. 1. 1. 2.]
 [4. 3. 2. 0.]]

C)
[[1. 2. 3. 4.]
 [2. 1. 0. 3.]
 [3. 0. 1. 2.]
 [4. 3. 2. 1.]]

D)
[[4. 2. 3. 1.]
 [3. 0. 1. 2.]
 [3. 1. 0. 3.]
 [1. 3. 2. 4.]]

Question 15 :

L'acronyme anglais d'un système de gestion de base de données est :

- A) DB2
- B) SGBD
- C) BIG DATA
- D) SQL Server

Question 16 :

Les index d'une base de données relationnelle ont pour fonction :

- A) de faciliter les modifications.
- B) de faciliter les suppressions.
- C) de faciliter les interrogations.
- D) Toutes les réponses précédentes conviennent.

Question 17 :

Quelle est l'instruction SQL qui permet de supprimer une ligne d'une table d'une base de données relationnelle ?

- A) DROP
- B) TRUNCATE
- C) DELETE
- D) RESET

Question 18 :

Dans une requête SQL, un tri s'opère dans la clause :

- A) MERGE
- B) ORDER BY
- C) SORT
- D) Toutes les réponses précédentes conviennent.

Question 19 :

Avec une table similaire appelée `sieges` :

vol_id	siege	client_id	prix_client
15	01A	5	130,7
15	01C	3	170
15	02A	1	120
15	02B	10	150
19	01A	6	130
24	01B	7	120
24	01C	10	155
24	02A	11	182,5
...			

La requête suivante est capable d'extraire :

```
SELECT vol_id, siege, client_id, prix_client
FROM   sieges
WHERE  prix_client > 130
AND    vol_id = 15
OR     vol_id = 24
ORDER BY vol_id, siege, prix_client
```

- A) les passagers qui ont voyagé sur un des deux vols en payant plus de 130 euros pour chaque vol.
- B) les passagers qui ont payé plus de 130 euros pour chacun des vols.
- C) les passagers qui ont voyagé ensemble sur deux vols en payant plus de 130 euros pour chaque vol.
- D) les passagers d'un vol ayant payé plus de 130 euros avec tous ceux d'un autre vol.

Question 20 :

Avec une table similaire et pour extraire les vols qui partent de Toulouse (TLS) avec ceux qui y arrivent, il faudra utiliser la condition :

VOL_ID	NUM_VOL	AERO_DEP	AERO_ARR	HEURE_DEP
15	AF6143	ORY	TLS	06/09/2017 15:45
19	AF6143	ORY	TLS	06/10/2017 17:45
24	AF6140	ORY	CDG	06/09/2017 16:00
65	AF6148	TLS	ORY	06/10/2017 19:45
...				

- A) WHERE = 'TLS' IN (aero_dep, aero_arr)
- B) WHERE aero_dep = 'TLS' AND aero_arr = 'TLS'
- C) WHERE aero_dep = 'TLS' OR aero_arr = 'TLS'
- D) WHERE aero_dep = 'TLS' XOR aero_arr = 'TLS'