TP

Titre EXO

Source EXO

Savoirs et compétences :

- ....

ALGO

ALG-000

Votre robot dispose de nombreux récepteurs et enregistre tous les signaux qui l'entourent. Cependant vous avez remarqué que certains de ces signaux sont très bruités. Vous décidez donc d'écrire un programme qui atténue le bruit de ces signaux, en effectuant ce que l'on appelle un lissage.

Une opération de lissage d'une séquence de mesures (des nombres décimaux) consiste à remplacer chaque mesure sauf la première et la dernière, par la moyenne des deux valeurs qui l'entourent.

Par exemple, si l'on part de la séquence de mesures suivantes : 1 3 4 5

On obtient après un lissage : 1 2.5 4 5

Le premier et dernier nombre sont inchangés. Le deuxième nombre est remplacé par la moyenne du 1er et du 3e, soit $(1+4)/2 = 2.5$, et le troisième est remplacé par $(3+5)/2 = 4$.

On peut ensuite repartir de cette nouvelle séquence, et refaire un nouveau lissage, puis un autre sur le résultat, etc. Votre programme doit calculer le nombre minimum de lissages successifs nécessaires pour s'assurer que la valeur absolue de la différence entre deux valeurs successives de la séquence finale obtenue ne dépasse jamais une valeur donnée, `diffMax`.

On vous garantit qu'il est toujours possible d'obtenir la propriété voulue en moins de 5000 lissages successifs.

On cherche à définir la fonction suivante `def lissage(t:list[float], diffmax:float) -> int :` où

- l'entrée est une liste `t` contenant les mesures, qui sont des flottants, et un flottant `diffmax`;
- la sortie est un entier qui correspond au nombre minimal de lissages nécessaire.

▪ **Exemple** `lissage([1.292, 1.343, 3.322, 4.789, -0.782, 7.313, 4.212], 1.120) -> 13.` ▪

ALG-001

Il existe de nombreuses traditions étranges et amusantes sur Algoréa, la grande course de grenouilles annuelle en fait partie. Il faut savoir que les grenouilles algoréennes sont beaucoup plus intelligentes que les grenouilles terrestres et peuvent très bien être dressées pour participer à des courses. Chaque candidat a ainsi entraîné sa grenouille durement toute l'année pour ce grand événement.
La course se déroule en tours et, à chaque tour, une question est posée aux dresseurs. Le premier qui trouve la réponse gagne le droit d'ordonner à sa grenouille de faire un bond. Dans les règles de la course de grenouilles algoréennes, il est stipulé que c'est la grenouille qui restera le plus longtemps en tête qui remportera la victoire. Comme cette propriété est un peu difficile à vérifier, le jury demande votre aide.
Ce que doit faire votre programme :

`nbg` numérotées de 1 à `nbg` sont placées sur une ligne de départ. À chaque tour, on vous indique le numéro de la seule grenouille qui va sauter lors de ce tour, et la distance qu'elle va parcourir en direction de la ligne d'arrivée.
Écrivez un programme qui détermine laquelle des grenouilles a été strictement en tête de la course à la fin du plus grand nombre de tours.

ENTRÉE : deux entiers `nbg` et `nbt` et un tableau `t`.
`nbg` est le nombre de grenouilles participantes.
`nbt` est le nombre de tours de la course.
`t` est un tableau ayant `nbt` éléments, et tel que chaque élément est un couple : (numéro de la grenouille qui saute lors de ce tour, longueur de son saut).
SORTIE : vous devez renvoyer un entier : le numéro de la grenouille qui a été strictement en tête à la fin du plus grand nombre de tours. En cas d'égalité entre plusieurs grenouilles, choisissez celle dont le numéro est le plus petit.

EXEMPLE :
entrée : (4, 6, [[2,2], [1,2], [3,3], [4,1], [2,2], [3,1]])
sortie : 2.

ALG-002

Un marchand de légumes très maniaque souhaite ranger ses petits pois en les regroupant en boîtes de telle sorte que chaque boîte contienne un nombre factoriel de petits pois. On rappelle qu'un nombre est factoriel s'il est de la forme 1, 1×2, $1\times 2\times 3$, $1\times 2\times 3\times 4\cdots$ et qu'on les note sous la forme suivante :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Il souhaite également utiliser le plus petit nombre de boîtes possible.
Ainsi, s'il a 17 petits pois, il utilisera :
2 boîtes de $3! = 6$ petits pois
2 boîtes de $2! = 2$ petits pois
1 boîte de $1! = 1$ petits pois.
ce qui donne bien $2 \times 3! + 2 \times 2! + 1 \times 1! = 12 + 4 + 1 = 17$.

D'une manière générale, s'il a `nbp` petits pois, il doit trouver une suite `a_1, a_2⋯, a_p` d'entiers positifs ou nuls avec `a_p`> 0 et telle que : `nbp=a_1 x 1!+a_2 x 2!+⋯+a_p x p!` avec `a_1+⋯+a_p` minimal.
Remarque mathématique : si à chaque étape on cherche le plus grand entier `k` possible tel que `k!` soit inférieur au nombre de petits pois restant, on est sûrs d'obtenir la décomposition optimale : en termes informatiques, on dit que l'algorithme *glouton* est optimal.

ENTRÉE : un entier, `nbp`, le nombre total de petits poids.
SORTIE : un couple constitué de l'entier `p` et du tableau `[ a_1, a_2, ⋯, a_p ]`.

EXEMPLE :
entrée : `17`
sortie : `(3, [ 1, 2, 2 ] )`.

ALG-003

Rien de tel que de faire du camping pour profiter de la nature. Cependant sur Algoréa, les moustiques sont particulièrement pénibles et il faut faire attention à l'endroit où l'on s'installe, si l'on ne veut pas être sans cesse piqué.

Vous disposez d'une carte sur laquelle est indiquée, pour chaque parcelle de terrain, si le nombre de moustiques est supportable ou non. Votre objectif est de trouver le plus grand camping carré évitant les zones à moustiques qu'il est possible de construire.

ENTRÉE : un tableau `t` de n éléments correspondant à des lignes, chacune de ces lignes contenant p élements, qui ne sont que des 0 et des 1. 0 signifie qu'il n'y a pas de moustiques et 1 qu'il y a des moustiques.
SORTIE : un entier : la taille maximale du côté d'un carré ne comportant que des 0 et dont les bords sont parallèles aux axes.

EXEMPLE 1 :

entrée : `t`=
```
[ [1, 0, 0, 1, 0, 0, 1],
  [0, 0, 0, 0, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 1, 0, 0, 0, 0, 1],
  [1, 0, 0, 0, 1, 0, 1] ]
```
sortie : 4.

EXEMPLE 2 :

entrée : `t`=
```
[ [0, 0, 0, 1, 1, 1, 1],
  [0, 0, 0, 1, 1, 1, 1],
  [0, 0, 0, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 0, 0],
  [1, 1, 1, 1, 1, 0, 0] ]
```
sortie : 3.

ALG-004

On ne s'intéresse pas ici à la validité d'un nombre écrit en chiffre romains, mais à sa valeur. On rappelle quelques principes de base. Les sept caractères de la numération romaine sont :

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

Certaines lettres sont dites d'*unité*. Ainsi on dit que I est une unité pour V et X, X est une unité pour L et C, C est une unité pour D et M.

Pour trouver la valeur d'un nombre écrit en chiffres romains, on s'appuie sur les règles suivantes :
- toute lettre placée à la droite d'une lettre dont valeur est supérieure ou égale à la sienne s'ajoute à celle-ci ;
- toute lettre d'unité placée immédiatement à la gauche d'une lettre plus forte qu'elle indique que le nombre qui lui correspond doit être retranché au nombre qui suit ;
- les valeurs sont groupées en ordre décroissant, sauf pour les valeurs à retrancher selon la règle précédente ; 1
- la même lettre ne peut pas être employée quatre fois consécutivement sauf M.

Par exemple, DXXXVI = 536, CIX = 109 et MCMXL = 1940.

1. Écrire une fonction `valeur (caractere)` qui retourne la valeur décimale d'un caractère romain. Cette fonction doit renvoyer 0 si le caractère n'est pas l'un des 7 chiffres romains.
2. Écrire la fonction principale `conversion (romain)` qui permet de convertir un nombre romain en nombre décimal. Cette fonction doit prendre en argument une chaîne de caractères `romain`. Si cette chaîne est écrite en majuscule et correspond à un nombre romain correctement écrit, la fonction doit renvoyer le nombre décimal égal au nombre romain passé en argument. Sinon, la fonction doit renvoyer -1.

ALG-004-cor

1. Nous pouvons proposer la fonction programme suivante :

```
def valeur (caractere) :
    """ renvoie la valeur décimale d'un caractè↙
        re
    s'il est utilisé dans la numération romaine
    renvoie 0 sinon.
    Précondition : caractere est une lettre de l'↙
        alphabet
    latin """
```

```
    if caractere == "I" :
        return 1
    elif caractere == "V" :
        return 5
    elif caractere == "X" :
        return 10
    elif caractere == "L" :
        return 50
    elif caractere == "C" :
        return 100
    elif caractere == "D" :
        return 500
    elif caractere == "M" :
        return 1000
    else :
        return 0
```

Notons que cette fonction serait plus courte en utilisant la notion de *dictionnaire* python, qui ne figure pas au programme.

2. Nous pouvons proposer la fonction programme suivante :

```
def conversion (romain) :
    L = ["I","V","X","L","C","D","M"]
    decimal = 0
    n = len(romain)
    if n==0 :
        return -1
    for k in range(n) :
        if romain[k] in L :
            v = valeur(romain[k])
            if k+1 < n and v < valeur(romain[k+1]) :
                decimal -= v
            else :
                decimal += v
        else :
            return -1
    return decimal
```

ALG-005

Pour tout $n \in \mathbb{N}\setminus\{0,1\}$, on pose $S_n = \sum_{k=2}^{n} \frac{(-1)^k}{k\ln k}$. On admet que la suite $(S_n)_{n\in\mathbb{N}\setminus\{0,1\}}$ a une limite finie ℓ en $+\infty$, et que pour tout $n \in \mathbb{N}\setminus\{0,1\}$, $u_{2n+1} \leq \ell \leq u_{2n}$.

1. Écrire un script Python donnant une valeur approchée de ℓ à 10^{-8} près.
2. Démontrer que le résultat donné par cet script est correct. On donnera clairement les éventuels invariants et variants des boucles intervenant dans le programme.

ALG-005-cor

1. Nous pouvons proposer le programme suivant :

```
from math import log

def limite () :
    """ renvoie une approxiamtion à10(—8) près de
    la limite de la suite (S_n) """
    k = 2
    u = (-1)**k/(k*log(k)) # u = S_k
    v = u + (-1)**(k+1)/((k+1)*log(k+1)) # v = S_{k+1}
    while abs(u-v) >= 2*10**(-8) :
        u = v
        k += 1
        v = u + (-1)**(k+1)/((k+1)*log(k+1))
        # invariant en sortie : u=S_k et v=S_{k+1}
        # variant : abs(u—v), réel positif
        # de limite nulle
    return (u+v)/2


limite()
```

2. Démontrons l'invariant : à chaque tour de boucle, $u = S_k$ et $v = S_{k+1}$.
Initialement, à l'entrée du premier tour de boucle, nous avons $k = 2$, $u = \frac{(-1)^2}{2\ln 2} = S_2$ et $v = u + \frac{(-1)^{k+1}}{(k+1)\ln(k+1)} = S_3$. L'invariant est donc vérifié.
Supposons l'invariant vérifié à l'entrée d'un tour de boucle donné. Notons alors k_1, u_1 et v_1 les valeurs à l'entrée de ce tour de boucle, et k_2, u_2 et v_2 leurs valeurs à la sortie. Alors $k_2 = k_1 + 1$, $u_2 = v_1 = S_{k_1+1} = S_{k_2}$ et $v_2 = u_2 + \frac{(-1)^{k_2+1}}{(k_2+1)\ln(k_2+1)} = S_{k_2+1}$: l'invariant est donc vérifié en sortie de boucle.
Par principe de récurrence, l'invariant est donc bien vérifié à chaque tour de boucle.

Démontrons maintenant la terminaison de l'algorithme : puisque nous savons que la suite (S_n) a une limite, alors à partir d'un certain rang la valeur absolue de la différence entre deux termes consécutifs de la suite est strictement inférieure à 2.10^{-8}. Puisqu'à l'entrée de chaque tour de boucle, $u = S_k$ et $v = S_{k+1}$, et puisque k augmente de 1 à chaque tour de boucle, alors à partir d'un certain nombre de tours de boucle, nous aurons $|u-v| < 2.10^{-8}$, et l'algorithme se termine donc.

Pour finir, démontrons la correction de l'algorithme : à la sortie du dernier tour de boucle, nous avons $|u-v| < 2.10^{-8}$. Or l'énoncé assure que la limite ℓ se trouve entre u et v. Ainsi, cette limite est à une distance inférieure à $\frac{|u-v|}{2}$ du milieu de l'intervalle $[u,v]$ (ou $[v,u]$). En renvoyant ce milieu, c'est-à-dire $\frac{u+v}{2}$, l'algorithme renvoie bien une approximation à 10^{-8} près de ℓ.

ALG-006

On appelle *nombre parfait* tout entier naturel non nul qui est égal à la somme de ses diviseurs stricts, c'est-à-dire de ses diviseurs autres que lui-même.

Par exemple, 26 n'est pas parfait, car ses diviseurs stricts sont 1, 2 et 13, et $1+2+13=16\neq 28$. Mais 28 est parfait, car ses diviseurs stricts sont 1, 2, 4, 7 et 14, et $1+2+4+7+14=28$.

QUESTION Écrire une fonction Python `parfait(n)` prenant en entrée un entier naturel non nul `n`, et renvoyant un booléen donnant la valeur de vérité de l'assertion « `n` est parfait ».

QUESTION Écrire les éventuels variants et invariants permettant de montrer que cette fonction renvoie le bon résultat.

ALG-007

QUESTION Écrire un script Python permettant de calculer le plus petit entier naturel n tel que $n!>123456789$.

QUESTION Écrire les éventuels variants et invariants de boucle permettant de montrer que le code Python écrit précédemment donne le bon résultat.

QUESTION Montrer que les variants et/ou invariants donnés à la question précédente sont bien des variants/invariants de boucle et justifier que le script écrit termine et donne le bon résultat.

ALG-008

On s'intéresse au problème du codage d'une suite de bits (représentée sous la forme d'un tableau de 0 ou 1), de manière à pouvoir réparer une erreur de transmission. On fixe un entier naturel non nul `k`. Un tableau de bits `b` sera codé avec un niveau de redondance `k` en répétant chaque bit $2k+1$ fois. Pour décoder un tableau avec un niveau de redondance `k`, on le découpe en blocs de $2k+1$ bits. Dans chaque bloc, on effectue un « vote » et l'on considère la valeur majoritaire.

Exemple : Avec $k=2$ (et donc un niveau de redondance de 2), le tableau

$$b=[0,1,0]$$

sera codé en

$$c=[\underbrace{0,0,0,0,0}_{5\text{ bits}},\underbrace{1,1,1,1,1}_{5\text{ bits}},\underbrace{0,0,0,0,0}_{5\text{ bits}}].$$

Imaginons qu'après transmission, le tableau reçu soit

$$c'=[\underbrace{0,0,0,1,0}_{1\text{ erreur}},\underbrace{0,1,1,0,1}_{2\text{ erreurs}},\underbrace{0,1,0,1,1}_{3\text{ erreurs}}].$$

On le décode alors en

$$b'=[0,1,1].$$

QUESTION Écrire une fonction `code(b,k)` renvoyant le tableau codant un tableau `b`, avec un niveau de redondance `k`.

QUESTION Écrire une fonction `decode(c,k)` renvoyant le tableau décodant un tableau `c`, avec un niveau de redondance `k`.

ALG-008-cor

```
def code(b,k):
    """Code le tableau de bits b avec niveau de redondance k
       Préconditions : b tableau de 0 / 1
                       k entier"""
    n = len(b)
    c = [0] * (n*(2*k+1))
    for i in range(n):
        if b[i] == 1 :
            for j in range(2*k+1) :
                c[i*(2*k+1)+j] = 1
            # On pouvait aussi écrire :
            # c[i(2k+1):(i+1)(2k+1)] = [1](2k+1)
    return c

def decode(c,k):
    """Décode le tableau de bits c avec niveau de redondance k
       Préconditions : c tableau de 0 / 1
                       len(c) multiple de 2k+1
                       k entier"""
    m = len(c)
    n = m // (2*k+1)
    b = [0]*n
    for i in range(n):
        v = 0
        for j in range(2*k+1):
            v = v + c[i*(2*k+1)+j]
        # v : somme des bits dans le ie bloc de c.
        # v > k ssi il y a une majorité de 1 dans ce bloc.
        if v > k :
            b[i] = 1
    return b
```

ALG-009

On considère un fichier `points.csv` contenant n lignes, chaque ligne contenant n entiers parmi 0 ou 1, séparés par des virgules. Ce fichier représente donc un tableau de nombres. Par exemple, le fichier

```
0,1,0,0
1,1,0,1
1,0,1,0
0,0,0,0
```

sera représenté (en `Python`) par le tableau bidimensionnel `t` (construit comme un tableau de tableaux) :

```
t = [ [0,1,0,0],
      [1,1,0,1],
      [1,0,1,0],
      [0,0,0,0] ]
```

QUESTION Écrire une fonction `lit_fichier(nom_de_fichier)` qui, à un tel fichier `nom_de_fichier`, renvoie le tableau associé.

On voit ce tableau comme décrivant des points dans le plan. Étant donné un tel tableau `t`, on considère que l'on a un point aux coordonnées `(i,j)` si et seulement si `t[i][j]` vaut 1. Par exemple, avec le tableau précédents, la liste `L` des points décrits est

```
L = [ (0,1) , (1,0) , (1,1) , (1,3) , (2,0) , (2,2) ]
```

QUESTION Écrire une fonction `lit_tableau(t)` qui, à un tel tableau bidimensionnel `t`, renvoie la liste des points décrits.

QUESTION Complexité ou invariants?

QUESTION Écrire une fonction `d(a,b)` qui, pour deux couples d'entiers `a`, `b`, dont nous noterons les coordonnées repectivement (x_a, y_a) et (x_b, y_b), renvoie la valeur $(x_a - x_b)^2 + (y_a - y_b)^2$.

On veut maintenant, étant donné un entier naturel k non nul et un couple d'entiers `c`, trouver les k couples de la liste de points les plus proches de `c`. S'il y a égalité entre plusieurs points, on n'en garde que k.

Par exemple [...]

On considère le code suivant.

```
def kNN(L,c,k,d):
    """k plus proches voisins du point c dans L
       d : fonction de distance"""
    v = []
    for j in range(L) :
        a = L[j]
        if len(v) < k :
            v.append(a)
        if d(a,c) < d(v[-1],c) :
            v[-1] = a
        i = len(v) - 1
        while i >= 1 and v[i] < v[i-1] :
            v[i-1],v[i] = v[i],v[i-1]
            i = i-1
    return v
```

QUESTION Donner l'invariant pour la boucle while et faire montrer que c'en est un.

QUESTION Donner l'invariant pour la boucle for.

QUESTION Complexité.

ALG-010

On s'intéresse au problème d'insertion d'un nombre dans un tableau de nombres trié par ordre croissant.

Soit `t` $= [t_0, \dots, t_{n-1}]$ un tableau de nombres trié par ordre croissant, c'est-à-dire que

$$t_0 \leq t_1 \leq \cdots \leq t_{n-1}.$$

On dit qu'un nombre x s'insère en position $i \in [\![0, n-1[\![$ dans le tableau `t` si $t_i \leq x \leq t_{i+1}$. Si $x < t_0$, alors x s'insère en position -1 dans `t` et, si $x > t_{n-1}$, alors x s'insère en position $n-1$ dans `t`.

QUESTION Écrire une fonction `position_insertion(t` prenant en argument un tableau de nombres `t` trié par ordre croissant et un nombre `x` et renvoyant une position où `x` s'insère dans `t`.

ALG-011

Soit $n \geq 2$ un entier. Un diviseur strict de n est un entier $1 \leq d \leq n-1$ qui divise n (c'est-à-dire que le reste de la division euclidienne de n par d est nul).

Deux entiers n_1 et n_2 sont dits *amicaux* si la somme des diviseurs stricts de n_1 vaut n_2 et si la somme des diviseurs stricts de n_2 vaut n_1.

QUESTION Écrire une fonction `amicaux(n,m)` qui prend en argument deux entiers naturels `n` et `m` et renvoie la valeur de vérité de « `n` et `m` sont amicaux ».

On pourra écrire une fonction auxiliaire, au besoin.

ARCHITECTURE

CHAINES

STR-000

Écrire une fonction prenant en paramètre deux chaînes de caractères `texte` et `mot`, et recherchant la première occurence de `mot` dans `texte` (la fonction retournera l'indice de la première lettre de l'occurrence de `mot` dans `texte`.

STR-001

Une chaîne de caractères $s_0 s_1 \dots s_{n-1}$ est un *palindrome* si elle est « symétrique » : $\forall k \in \{0, \dots, n-1\}, s_k = s_{n-1-k}$.

QUESTION Écrire une fonction `est_pal(s)` qui, à une chaîne de caractères `s`, renvoie le booléen `True` si `s` est un palindrome et `False` sinon.

QUESTION Écrire une fonction `max_pal(s)` qui, à une chaîne de caractères `s`, renvoie la chaîne `p` où `p` est la plus grande sous chaîne palindromique centrée (c'est la plus grande sous-chaîne palindromique de la forme $s_k s_{k+1} \dots s_{n-1-k}$).

COMPLEXITE

COM-000

Dans ce problème, on considère des tableaux d'entiers relatifs `t=[t0,t1,...,tn-1]`, et on appelle *tranche* de `t` toute sous-tableau non vide `[ti,ti+1,...,tj-1]` d'entiers consécutifs de ce tableau (avec $0 \leq i < j \leq n$) qu'on notera désormais `t[i:j]`.

À toute tranche `t[i:j]` on associe la somme `s[i : j]` $= \sum_{k=i}^{j-1} t\_k$ des éléments qui la composent. Le but de ce problème est de déterminer un algorithme efficace pour déterminer la valeur minimale des sommes des tranches de `t`.

L'algorithme naïf

1. Définir une fonction `somme` prenant en paramètre un tableau `t` et deux entiers i et j, et retournant la somme `s[i : j]`.
2. En déduire une fonction `tranche_min1` prenant en paramètre un tableau `t` et retournant la somme minimale d'une tranche de `t`.

Cycle 01

3. Montrer que la complexité de cet algorithme est en $\Theta(n^3)$, c'est-à-dire qu'il existe deux constantes $a, b \in \mathbb{R}_+^*$ telles que si `t` a n éléments, alors le nombre d'opérations effectuées dans le calcul de `tranche_min1(t)` est compris entre an^3 et bn^3.

0.2 Un algorithme de coût quadratique

1. Définir, sans utiliser la fonction `somme`, une fonction `mintranche` prenant en paramètres un tableau `t` et un entier `i`, et calculant la valeur minimale de la somme d'une tranche de `t` dont le premier élément est t_i , en parcourant une seule fois la liste a à partir de l'indice `i`.
2. En déduire une fonction `tranche_min2` permettant de déterminer la somme minimale des tranches de `t`, en temps quadratique, c'est-à-dire que la complexité de cet algorithme est en $\Theta(n^2)$. On justifiera que la complexité est précisément en $\Theta(n^2)$.

0.3 Un algorithme de coût linéaire

Étant donnée un tableau `t`, on note m_i la somme minimale d'une tranche quelconque du tableau `t[0 : i]`, et c_i la somme minimale d'une tranche de `t[0 : i]` se terminant par t_{i-1}.

Montrer que $c_{i+1} = \min(c_i + t_i, t_i)$ et $m_{i+1} = \min(m_i, c_{i+1})$, et en déduire une fonction `tranche_min3` de coût linéaire (c'est-à-dire dont la complexité est en $\Theta(n)$), calculant la somme minimale d'une tranche de `t`.

COM-001

Un enjeu scientifique et technologique actuel est de savoir traiter des problèmes mettant en jeu un nombre très important de données. Un point crucial est souvent de pouvoir manipuler des matrices de très grandes dimensions, ce qui est *a priori* très coûteux, en temps de calcul et en mémoire. On peut cependant souvent considérer que les matrices manipulées ne contiennent que « peu » d'éléments non nuls : c'est ce que l'on appelle les matrices *creuses*.

Nous nous intéressons ici à l'implémentation de deux algorithmes d'addition de matrices : l'un pour une représentation classique des matrices, l'autre pour une représentation des matrices creuses.

Soit $n \in \mathbb{N}^*$, on note $\mathcal{M}_n(\mathbb{R})$ l'ensemble des matrices carrées d'ordre n, à coefficients dans $\mathbb{R}$.

On représentera classiquement une matrice $M \in \mathcal{M}_n(\mathbb{R})$ par un tableau à double entrées. En Python, cela sera un tableau (type `list`) de longueur n, chaque élément de ce tableau représentant une ligne de M. Chaque élément de ce tableau est donc un tableau de longueur n, dont tous les éléments sont des nombres (types `int` ou `float`).

Cette même matrice M sera représentée de manière creuse en ne décrivant que ses cases non vides par un tableau de triplets `(i,j,x)`, où `x` est l'élément de M situé sur la i^e ligne et la j^e colonne. On pourra supposer que les éléments non nuls de M sont ainsi décrits ligne par ligne.

▪ **Exemple** La matrice $\begin{pmatrix} 1 & 0 & 5 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ sera représentée classiquement par le tableau

```
[ [1,0,5] , [0,-2,0] , [0,0,0] ]
```

et de manière creuse par le tableau

```
[ (0,0,1) , (0,2,5) , (1,1,-2) ].
```

▪

QUESTION Écrire une fonction `add(M,N)` prenant en argument deux représentations classiques `M` et `N` de deux matrices M et N (carrées, de même ordre) et renvoyant la représentation classique de $M+N$. On prendra soin d'écrire une fonction « optimale » en terme de complexité, spatiale et temporelle.

QUESTION Écrire une fonction `add_creuse(M,N)` prenant en argument deux représentations creuses `M` et `N` de deux matrices M et N (carrées, de même ordre) et renvoyant la représentation creuse de $M+N$. On prendra soin d'écrire une fonction « optimale » en terme de complexité, spatiale et temporelle.

QUESTION On suppose que M et N sont carrées, d'ordre n, représentées classiquement par `M` et `N`. Évaluer asymptotiquement la complexité temporelle de la fonction `add(M,N)`.

QUESTION On suppose que M et N sont carrées et contient chacune au plus p élements non nuls, représentées de manière creuse par `M` et `N`. Évaluer asymptotiquement la complexité temporelle de la fonction `add_creuse(M,N)`.

Pour simplifier, on ne justifiera pas que les éléments obtenus sont disposés dans le bon ordre.

QUESTION Discuter du choix de la représentation pertinente à utiliser pour additionner deux matrices.

COM-001-cor

QUESTION Voici un algorithme répondant à la question.

```
def add(M,N):
    """M+N, représentation classique"""
    n = len(M)
    S = [[0]*n for i in range(n)] # M+N
    for i in range(n):
        # Invariant : S[:i] = (M+N)[:i]
        for j in range(n):
            # Invariant : S[:i+1][:j] = (M+N)[:i+1][
            S[i][j] = M[i][j] + N[i][j]
            # Invariant : S[:i+1][:j+1] = (M+N)[:i+1
        # Invariant : S[:i+1] = (M+N)[:i+1]
    return S
```

Montrons que `add(M,N)` renvoie bien le bon résultat. On a supposé que `M` et `N` sont de même dimension.

Ligne 4 `S` représente bien une matrice carrée d'ordre `n`.

On a $S[:0] = []$ et $S[:0][:0] = []$, donc l'invariant de boucle de la ligne 4 est bien vérifié en début de boucle.

Ligne 6 On suppose que, pour tout $0 \leq a < i$ et $0 \leq b < j$, $S[a][b] = M[a][b]+N[a][b]$. Notamment, $S[i][:0] = []$, donc l'invariant de boucle de la ligne 6 est bien vérifié en début de boucle.

Ligne 8 On suppose que, pour tout $0 \leq b < j$, $S[i][b] = M[i][b]+N[i][b]$.

Ligne 9 Directement, $S[i][j] = M[i][j]+N[i][j]$.

Ligne 10 Ainsi, $S[:i+1][:j+1] = (M+N)[:i+1][:j+1]$.

Ligne 11 En fin de boucle, $j = n-1$, donc $S[:i+1] = (M+N)[:i+1]$.

Ligne 12 En fin de boucle, $i = n-1$, donc $S = M+N$. L'algorithme renvoie bien le bon résultat.

QUESTION Voici un algorithme répondant à la question.

```
1  def lexico(X,Y):
2      """(a,b) < (c,d) pour l'ordre lexicographique
3          avec X = (a,b,x), Y = (a,b,y)"""
4      a,b,_ = X
5      c,d,_ = Y
6      return (a < c) or (a==c and b < d)
7
8  def add_creuse(M,N):
9      """M+N, représentation creuse"""
10     m,n = len(M), len(N)
11     iM, iN = 0,0 # Indices des prochains éléments de M, N à ajouter
12     S = [] # M+N
13     while (iM < m) or (iN < n):
14         # Invariant : M[:iM] et N[:iN] ont été ajoutés dans S
15         # Variant : m+n-iM-iN
16         if (iM == m) or (iN < n and lexico(N[iN],M[iM])):
17             S.append(N[iN])
18             iN += 1
19         elif (iN == n) or lexico(M[iM],N[iN]):
20             S.append(M[iM])
21             iM += 1
22         else:
23             if M[iM][2]+N[iN][2] != 0:
24                 S.append((M[iM][0],M[iM][1],M[iM][2]+N[iN][2]))
25             iM +=1
26             iN +=1
27     return S
```

Montrons que `add_creuse(M,N)` renvoie bien le bon résultat.

Fonction lexico `lexico((i,j,x),(a,b,y))` renvoie un booléen indiquant si `(i,j)` est strictement plus petit que `(a,b)` dans l'ordre lexicographique.

Lignes 11-12 `M[:iM]` et `N[:iN]` sont vides donc `S` (vide) contient bien la somme des éléments de `M[:iM]` et `N[:iN]`.

Ligne 14 Supposons que tous les éléments de `M[:iM]` et `N[:iN]` ont été ajoutés (leur somme se trouve dans `S`). Supposons de plus que tous les éléments de `M[:iM]` et `N[:iN]` sont strictement plus petits que tous les éléments de `M[iM:]` et `N[iN:]`. On a trois cas possibles.

- Le prochain élément à ajouter est `N[iN]`, à une case contenant un zéro de M.
- Le prochain élément à ajouter est `M[iM]`, à une case contenant un zéro de N.
- Le prochain élément à ajouter correspond au cas où l'on ajoute deux cases non nulle de M et de N, de mêmes indices : $M[iM]$ et $N[iN]$.

Lignes 15-17 On est dans le premier cas, soit $iM = m$, dans ce cas $M[iM:]$ est vide, soit $N[iN]$ est strictement plus petit que $M[iM]$ lexicographiquement. Dans les deux cas, le prochain élément à ajouter est $N[iN]$ (avec un zéro de M) et l'invariant de la ligne 14 est vérifié en $(iM, iN+1)$.

Ligne 18 On a donc $iM < m$ et, si $iN < n$, on a $M[iM] \leq N[iN]$ lexicographiquement.

Lignes 18-20 On est dans le deuxième cas, soit $iN = n$, dans ce cas $N[iN:]$ est vide, soit $M[iM]$ est strictement plus petit que $N[iN]$ lexicographiquement. Dans les deux cas, le prochain élément à ajouter est $M[iM]$ (avec un zéro de N) et l'invariant de la ligne 14 est vérifié en $(iM+1, iN)$.

Ligne 21 On a donc $iM < m$ et $iN < n$, avec $M[iM][:2] = N[iN][:2]$. On est donc dans le troisième cas, on additionne deux cases non nulles de M et de N de mêmes indices.

Lignes 22-23 Si la somme des cases est non nulle, on la rajoute à S, sinon cette case « disparaît ».

Lignes 24-25 L'invariant de la ligne 14 est vérifié en $(iM+1, iN+1)$.

Ligne 26 À la sortie de la boucle `while`, on a donc $m = iM$ et $n = iN$, donc M et N ont bien été ajoutés dans S. Le résultat donné est donc correct.

Variant Dans tous les cas, la quantité $m+n-iM-iN$ a diminué d'au moins 1 (et d'au plus deux). De plus, $iM < m$ ou $iN < n$ si et seulement si $m+n-iM-iN > 0$, donc la boucle `while` termine.

QUESTION On considère le modèle de complexité usuel.

Comptons les opérations effectuées, on donne pour chacune une domination asymptotique du nombre d'opérations élémentaires effectuées, en fonction de n.

- Un calcul de taille de tableau par la fonction `len` : $O(1)$.
- Création d'un tableau de taille n : n fois le temps de création de chaque élément de S. Chaque élément de S est créé comme un tableau de taille n et se fait en temps $O(n)$. Ainsi, S est créé en temps $O(n)$.
- Une création d'objet `range` : $O(1)$.
- Dans chaque tour de la boucle indicée par `i` (n tours en tout) :
  - Une création d'objet `range` : $O(1)$.
  - Dans chaque tour de la boucle indicée par `j` (n tours en tout) : deux lectures dans un tableau, une addition de nombres, une écriture dans un tableau : $O(1)$

Ainsi, le nombre d'opérations élémentaires est en $O(n^2)$. Remarquons que l'on ne pouvait pas faire mieux (asymptotiquement) : chaque case de M doit être parcourue et M possède n^2 cases.

Notons A_n le nombre d'opérations élémentaires effectuées pour calculer `add(M,N)`. Il existe donc une constante K telle que, pour tout $n \in \mathbb{N}$,

$$n^2 \leq A_n \leq K \times n^2.$$

QUESTION On considère le modèle de complexité

usuel.

Comptons les opérations effectuées, on donne pour chacune une domination asymptotique du nombre d'opérations élémentaires effectuées, en fonction de $p = \max(m,n)$.

- Remarquons qu'un appel de `lexico` se fait en temps $O(1)$.
- On a deux calculs de longueurs par la fonction `len` et cinq affectations : $O(1)$.
- Chaque tour dans la boucle `while` se fait en $O(1)$: au plus 5 comparaisons, 2 appels à `lexico`, 12 accès à des tableaux, 1 ajout à une liste et deux incrémentations.
- Comme, à chaque tour de boucle, $m+n-iM-iN$ est réduit de 1 ou deux, on a au plus $m+n \leq 2p$ tours de boucle.

Ainsi, le nombre d'opérations élémentaires est en $O(p^2)$. Remarquons que l'on ne pouvait pas faire mieux (asymptotiquement) : chaque case non vide de M (et de N) doit être parcourue et M (ou N) possède p cases.

Notons C_p le nombre d'opérations élémentaires effectuées pour calculer `add_creuse(M,N)`. Il existe donc une constante K' telle que, pour tout $p \in \mathbb{N}$,

$$p \leq C_p \leq K' \times p.$$

QUESTION On discute ici de complexités asymptotiques, or nous avons deux variables sur lesquelles jouer : n et p. Supposons que p varie en fonction de n (on le note $p(n)$). Notamment, $\dfrac{p(n)}{n^2}$ est la proportion de cases non vides de M et de N.

Avec la constante K' trouvée précédemment,

$$\forall n \in \mathbb{N}, \frac{C_{p(n)}}{A_n} \leq K' \times \frac{p(n)}{n^2}.$$

Ainsi, dès que $\dfrac{p(n)}{n^2} \leq \dfrac{1}{K'}$, on a $C_{p(n)} \leq A_n$. Il existe donc une proportion de cases non vides en deçà de laquelle il est toujours préférable d'utiliser la représentation creuse des matrices.

De même, il existe une proportion de cases non vides au dessus de laquelle il est toujours préférable d'utiliser la représentation classique des matrices.

Notre analyse ne nous permet pas d'être plus précis.

COM-002

On considère la suite u à valeurs dans $[\![0;64\,007[\![$ définie par

$$u_0 = 42, \quad \forall n \in \mathbb{N},\ u_{n+1} = 15\,091\,u_n\ [64\,007],$$

ainsi que, pour tout $n \in \mathbb{N}$, $S_n = \sum_{k=0}^{n} u_k$.

On propose l'algorithme suivant pour calculer les valeurs de S.

```
def u(n):
    """u_n, n : entier naturel"""
    v = 42
    # Inv : v = u_0
    for k in range(n):
        # Inv : v = u_k
        v = 15091 * v % 64007
        # Inv : v = 15091*u_k % 64007 = u_k+1
    # Inv : au dernier tour, k = n-1, donc v = u_n
    return v

def S(n):
    """u_n, n : entier naturel"""
    s = u(0)
    # Inv : s = S_0
    for k in range(n):
        # Inv : s = S_k
        s = s + u(k+1)
        # Inv : v = S_k+u_k+1 = S_k+1
    # Inv : au dernier tour, k = n-1, donc s = S_n
    return s
```

1. Étudier les complexités des fonctions `u` et `S`, en fonction de n.
2. Écrire une fonction donnant la valeur de S_n en temps $O(n)$.

COM-003

On considère la suite u définie par

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

On se considère la fonction suivante, permettant de calculer les valeurs de u.

```
def u(n):
    """u_n, n : entier naturel"""
    v = 2
    # Inv : v = u_0
    for k in range(n):
        # Inv : v = u_k
        v = v*v
        # Inv : v = u_k**2 = u_k+1
    # Inv : au dernier tour, k = n-1, donc v = u_n
    return v
```

Pour étudier le temps d'exécution d'une fonction, on pourra utiliser le morceau de code suivant.

```
import timeit

REPEAT=3

def duree(f, x):
  """Calcule le temps mis par Python pour calculer f
  Cette fonction effectue en fait le calcul de f(x) 
  et garde la valeur la plus petite
  (l'idée est d'éliminer les éventuelles perturbatio
  par d'autres processus tournant sur la machine)"""
  t = timeit.Timer(stmt=lambda : f(x))
  time = min(t.repeat(REPEAT, number=1))
  return time
```

1. Étudier en fonction de n la complexité asymptotique de la fonction `u`, dans le modèle standard.
2. Tracer les temps de calculs de u_k pour $k \in [\![0;30[\![$ par la fonction `u`. Discuter le résultat.

Indication : on pourra utiliser une échelle semi-logarithmique.

3. Proposer un modèle de complexité plus réaliste et étudier dans ce modèle n la complexité asymptotique de la fonction `u`. On pourra déterminer explicitement u_n.

COM-004

On considère la fonction `Python` suivante.

```
def mystere(t):
    """t : tableau d'entiers"""
    m = 0
    for i in range(len(t)):
        for j in range(i+1,len(t)):
            if t[j] - t [i] > m :
                m = t[j]-t[i]
    return m
```

QUESTION Que contient le résultat renvoyé par `mystere(t)` ? Justifier.

QUESTION Quelle est la complexité de la fonction `mystere(t)`, en fonction de la longueur de `t`, que l'on notera n ?

COM-005

On considère la fonction `Python` suivante.

```
def mystere(u,v):
    """u,v : tableaux d'entiers"""
    m = 0
    p,q = len(u),len(v)
    for i in range(p) :
        for a in range(q) :
            k = 0
            while i+k < p and a+k < q and u[i+k] == v[a+k]:
                k = k+1
            if k > m :
                m = k
    return m
```

QUESTION Que contient le résultat renvoyé par `mystere(u,v)` ? Justifier.

QUESTION Quelle est la complexité de la fonction `mystere(u,v)`, en fonction du maximum des longueurs de `u` et `v`, que l'on notera n ?

COM-005-cor

QUESTION Pour la boucle while des lignes 8-9, $p-k$ est clairement un variant, donc cette boucle termine bien. La fonction `mystere` renvoie donc un résultat.

Toujours pour cette boucle while, « $u[i : i+k] = v[a : a+k]$ » est clairement un invariant. Ainsi, à la sortie de cette boucle, on a $u[i : i+k] = v[a : a+k]$ mais $u[i+k] \neq v[a+k]$. Ainsi, k est la taille de la plus longue tranche commune à u et v partant respectivement des positions i et a.

Un invariant pour la boucle for des lignes 5 à 11 est alors : « m est la taille de la plus longue tranche commune à u et v, dont le premier élément de la tranche de u est dans $u[: i]$ ».

Un invariant pour la boucle for des lignes 6 à 11 est alors : « m est la taille de la plus longue tranche commune à u et v, dont le premier élément de la tranche de u est dans $u[: i]$ ou bien le premier élément de la tranche de u est $u[i]$ et les premiers éléments de la tranche de v est dans $v[: a]$ ».

À la fin de ces boucles, m est donc la longueur de la plus grande sous-tranche commune à u et v : c'est ce que renvoie la fonction `mystere(u,v)`.

QUESTION On note n le maximum des longueurs de u et de v. On se place dans le modèle de complexité usuelle : les affectations, calculs de longueur de list, accès à un élément d'une liste et les comparaisons de nombres se font en temps $O(1)$.

Les lignes 3 et 4 s'effectuent donc en $O(1)$.

La ligne 9 s'effectue en temps $O(1)$. Il y a au plus n tours de boucle dans la boucle while des lignes 8-9, donc cette boucle s'effectue en $O(n)$.

Les lignes 7-10-11 s'effectuent en $O(1)$, donc le bloc des lignes 7 à 11 s'effectue en temps $O(1)+O(n)=O(n)$.

Il y a au plus n tours dans la boucle des lignes 6 à 11, donc cette boucle s'effectue en temps $n \times O(n) = O(n^2)$.

Il y a au plus n tours dans la boucle des lignes 5 à 11, donc cette boucle s'effectue en temps $n \times O(n^2) = O(n^3)$.

La fonction `mystere(u,v)` a donc une complexité en $O(n^3)$.

COM-006

On considère le code suivant, qui crée une liste aléatoire `L` et qui la trie ensuite par ordre croissant par la méthode du «tri par insertion».

```
from random import randrange

def liste_triee(n):
    """Renvoie une liste d'éléments de range(100), de
       triée par ordre croissant """
    L = []
    for i in range(n):
        L.append(randrange(100))
    for i in range(1,n):
        # Inv : L[:i] est triée par ordre croissant
        # Idée : on fait redescendre L[i] pour l'insé
        j = i
        while j >= 1 and L[j] < L[j-1]:
            # On échange L[j-1] et L[j]
            L[j], L[j-1] = L[j-1], L[j]
            j = j-1
    return L
```

Un appel de `randrange(100)` renvoie un nombre tiré aléatoirement et uniformément dans $[\![0, 100[\![$ et a une complexité en $O(1)$.

QUESTION Peut-on donner explicitement et exactement une complexité pour la boucle `while` des lignes 13 à 16 ? Que peut-on quand même proposer comme type de complexité pour cette boucle ?

Donner dans ce cas la complexité de cette boucle en fonction de `i`.

QUESTION Donner dans ce cas la complexité d'un appel de `liste_triee(n)` en fonction de `n`.

COM-006-cor

On se place dans le modèle de complexité usuel. Toutes les opérations de cette fonction sont en $O(1)$.

QUESTION Le nombre de tours de cette boucle n'est pas déterminé à l'avance, on ne peut donc mener un calcul de complexité exact. On peut cependant déterminer une complexité dans le pire des cas. Il y aura au plus i tours de boucle (un variant pour cette boucle est j). Chaque tour de boucle a une complexité en $O(1)$ (nombre borné d'opérations en $O(1)$). La complexité dans le pire des cas de cette boucle est en $O(i)$.

QUESTION En dehors des boucles, il y a une opération en $O(1)$ (affectation).

Boucle `for` des lignes 7-8 : chaque tour a une complexité en $O(1)$, il y a n tours de boucle, la boucle a donc une complexité en $O(n)$.

Boucle `for` des lignes 9-16 : la boucle while a une complexité en $O(i)$, chaque tour a donc une complexité en $O(i)+O(1)=O(i)$, il y a n tours de boucle, la boucle a donc une complexité en

$$O\left(\sum_{i=1}^{n} i\right) = O\left(\frac{n(n+1)}{2}\right) = O(n^2).$$

La complexité de cette fonction est donc

$$O(n^2)+O(n)+O(1)=O(n^2).$$

COM-007

Beaucoup de problèmes technologiques actuels mettent en jeu des données de grandes dimensions et font souvent intervenir de grandes matrices.

Beaucoup de ces matrices sont *creuses*, c'est-à-dire qu'elles ne contiennent que peu de valeurs non nulles. Nous allons développer une méthode de codage de telles matrices et étudier leur intérêt.

Dans tout ce problème, on s'interdira d'utiliser les opérations entre vecteurs et matrices `numpy` (méthode `.dot()` par exemple).

Dans tout ce problème, on utilisera le type `array` de `numpy` pour représenter des vecteurs. On pourra supposer que la ligne suivante a été écrite :

```
from numpy import array, zeros
```

Dans tout ce problème, on veillera à l'optimalité des fonctions écrites en termes de complexité temporelle asymptotique. Les réponses clairement sous-optimales seront pénalisées.

Soit une matrice réelle $A \in \mathcal{M}_{n,p}(\mathbb{R})$ de dimension $n \times p$ et possédant s coefficients non nuls. Ce coefficient s est appelé *niveau de remplissage* de la matrice A. Comme toujours en Python, on numérotera les lignes et les colonnes en partant de 0.

Une telle matrice se code usuellement comme un tableau de nombres de dimension $n \times p$.

Le codage CSR (Compress Sparse Raw) code la matrice A par trois listes V, L et C définies comme suit.

- La liste V contient toutes les valeurs des coefficients non nuls de A, en les listant ligne par ligne (de la première à la dernière ligne puis de la première à la dernière colonne). Ainsi, V est de longueur s.
- La liste L est de longueur $n+1$, L_0 vaut toujours 0 et pour chaque $1 \leq i \leq n$, L_i vaut le nombre de coefficients non nuls dans les i premières lignes de A (*i.e.* de la ligne d'indice 0 à celle d'indice $i-1$, inclu).
- La liste C contient les numéros de colonne de chaque coefficients non nuls de A, en les listant ligne par ligne (de la première à la dernière ligne puis de la première à la dernière colonne). Ainsi, C est de longueur s.

Par exemple, avec

$$A = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

on a $n=3$, $p=4$, $s=4$ et sa représentation CSR est donnée par les trois listes

$$V = [-1,2,4,-1],$$
$$L = [0,1,3,4],$$
$$C = [1,0,2,3].$$

On remarquera qu'ajouter une colonne nulle à droite de A ne change pas sa représentation CSR.

QUESTION Déterminer la représentation CSR de la matrice

$$A = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

QUESTION Donner une matrice dont la représentation CSR est

$$V = [2,-3,1,1],$$
$$L = [0,0,2,4],$$
$$C = [0,3,2,3].$$

QUESTION Soit (V,L,C) la représentation CSR de A, soit $0 \leq i \leq n-1$. Combien y a-t-il de coefficients non nuls sur la ligne n° i de A ? Donner deux expressions Python (en fonction de V, L, C et i) permettant d'obtenir respectivement la liste des valeurs de ces coefficients et la liste des indices colonnes de ces coefficients.

On rappelle la formule du produit matriciel : pour une matrice $A \in \mathcal{M}_{n,p}(\mathbb{R})$ de coefficients $a_{i,j}$ et un vecteur $X \in \mathcal{M}_{p,1}(\mathbb{R})$ de coefficients x_j, si $0 \leq i \leq n-1$, alors la i^{e} coordonnée de AX est

$$\sum_{j=0}^{p-1} a_{i,j} x_j.$$

QUESTION Écrire une fonction `coeff_prod(V,L,C,X,i)` renvoyant la i^{e} coordonnée du produit AX, où le triplet

(V,L,C) est la représentation CSR de A. On supposera que les dimension de A et X sont compatibles pour effectuer le produit AX.

Donner la complexité asymptotique de cette fonction, en fonction de n, p et ℓ_i, où ℓ_i est le nombre d'éléments non nuls sur la i^e ligne de A.

QUESTION Écrire une fonction `prod(V,L,C,X)` renvoyant le produit AX, où le triplet (V,L,C) est la représentation CSR de A. On supposera que les dimension de A et X sont compatibles pour effectuer le produit AX.

Donner la complexité asymptotique de cette fonction, en fonction de n, p et s.

QUESTION Écrire une fonction `prod_naif(A,X)` renvoyant le produit AX, où A est codée usuellement comme un tableau à double dimension.

Donner la complexité asymptotique de cette fonction, en fonction de n, et p.

QUESTION En les comparant, discuter des deux complexités précédentes, notamment en fonction du niveau de remplissage s.

On prend maintenant pour exemple celui de la dérivation discrète, typique en traitement du signal. Pour un vecteur de longueur n

$$X=\begin{pmatrix} x_0 \\ \vdots \\ x_{n-1}\end{pmatrix}$$

on définit la dérivée discrète de X comme le vecteur Y de longueur n définit par :

$$y_0=x_1-x_0,\ y_{n-1}=x_{n-1}-x_{n-2} \text{ et } \forall i\in[\![1,n-1[\![,\ y_i=\frac{1}{2}(x_{i+1}-x_{i-1}).$$

QUESTION Déterminer une matrice A vérifiant $Y=AX$.

QUESTION Écrire une fonction `CSR_A(n)` renvoyant les trois vecteurs `V`, `L` et `C` codant A au format CSR.

Indication : on pourra écrire une fonction pour la création de chaque vecteur

COM-007-cor

QUESTION C'est

$$V=[1;3;-2]$$
$$L=[0;2;2;3;3]$$
$$C=[0;2;1]$$

QUESTION C'est

$$A=\begin{pmatrix}0&0&0&0\\2&0&0&-3\\0&0&1&1\end{pmatrix}.$$

QUESTION Comme il y a $L[i]$ éléments non nuls jusqu'à la ligne n^o $i-1$ et $L[i+1]$ éléments non nuls jusqu'à la ligne n^o i, il y a $L[i+1]-L[i]$ éléments non nuls sur la ligne n^o i. Dans V et C, ces éléments ont donc des indices allant de $L[i]$ (inclu) à $L[i+1]$ (exclu). On peut récupérer ces valeurs avec deux tranches :

```
V[L[i] : L[i+1]]
C[L[i] : L[i+1]]
\end{pyverbatim}
\question{}
\begin{pyverbatim}
def coeff_prod(V,L,C,X,i) :
    """ieme coefficient de AX
    V,L,C : représentation CSR de A"""
    S = 0
    for j in range(L[i],L[i+1]):
        S = S + X[C[j]] * V[j]
    return S
```

Toutes les opérations de cette fonction s'effectuent en temps constant. La boucle for effectue ℓ_i tours, chacun de complexité $O(1)$, donc est de complexité $O(\ell_i)$. En dehors de la boucle, il y a une affectation en $O(1)$. La complexité de cette fonction est donc en $O(1)+O(\ell_i)=O(\ell_i)$.

QUESTION

```
def prod(V,L,C,X) :
    n = len(L)-1
    Y = zeros(n,1)
    for i in range(n) :
        Y[i] = coeff_prod(V,L,C,X,i)
    return Y
```

Le calcul de n se fait en $O(1)$, la création de Y en $O(n)$. D'après la question précédente, chaque tour de boucle s'effectue en $O(\ell_i)$, donc la boucle a pour complexité $O(\sum_{i=0}^{p-1}\ell_i)=O(s)$. La fonction a donc pour complexité $O(n+s)$.

QUESTION

```
def prod_naif(A,X) :
    n,p = A.shape
    Y = zeros(n,1)
    for i in range(n) :
        S = 0
        for j in range(p) :
            S = S + A[i,j] * X[j]
    return Y
```

La complexité de cette fonction est en $O(np)$ (deux boucles imbriquées, l'une en $O(p)$, l'autre réalisant n tours, le calcul de n,p se faisant en $O(1)$ et la création de Y en $O(n)$).

QUESTION On voit facilement que la complexité du produit naïf est exactement de l'ordre de np.

On a toujours $s\leq np$. Toutefois, si s est négigeable devant np, alors la complexité de la multiplication sous format CSR est négligeable devant celle du produit naïf.

QUESTION On a directement

$$A=\begin{pmatrix}-1&1&0&\dots&\dots&0\\-.5&0&.5&0&&\vdots\\0&-.5&0&.5&\ddots&\vdots\\\vdots&\ddots&\ddots&\ddots&\ddots&0\\\vdots&&0&-.5&0&.5\\0&\dots&\dots&0&-1&1\end{pmatrix}$$

QUESTION Sur chaque ligne il y a 2 coefficients non nuls :

- sur la ligne n° 0, aux colonnes 0 et 1, avec pour valeurs -1 et 1 ;
- si $1 \leq i < n-1$, sur la ligne n° i, aux colonnes $i-1$ et $i+1$, avec pour valeurs $-.5$ et $.5$;
- sur la ligne n° $n-2$, aux colonnes $n-2$ et $n-1$, avec pour valeurs -1 et 1

On écrit donc

```
def V_A(n) :
    """Vecteur V de la représentation CSR de A"""
    V = zeros(2*n)
    V[0], V[1], V[2*n-2], V[2*n-1] = -1,1,-1,1
    for i in range(1,n-1):
        V[2*i], V[2*i+1] = -.5,.5
    return V

def L_A(n) :
    """Vecteur L de la représentation CSR de A"""
    L = zeros(n+1)
    for i in range(1,n+1) :
        L[i] = 2*i
    return L

def C_A(n) :
    """Vecteur C de la représentation CSR de A"""
    C = zeros(2*n)
    C[0], C[1], C[2*n-2], C[2*n-1] = 0,1,n-2,n-1
    for i in range(1,n-1):
        C[2*i], C[2*i+1] = i-1,i+1
    return C

def CSR_A(n) :
    """Représentation CSR de A"""
    return V_A(n), L_A(n), C_A(n)
```

EQUADIFFS

EQD-000

On considère une équation différentielle $(\mathscr{E}) : y' = F(y,t)$, où F est une fonction de $\mathbb{R}^n \times \mathbb{R}$ dans $\mathbb{R}^n$, et où l'inconnue y est une fonction de $\mathscr{C}^1(\mathbb{R},\mathbb{R}^n)$, avec la condition initiale $y(t_0) = y_0$.

QUESTION Décrire le principe de la méthode d'Euler. On donnera clairement la relation de récurrence qui est au coeur de cette méthode, en expliquant bien ce que représente la suite vérifiant cette relation de récurrence.

QUESTION Écrire en Python une fonction mettant en œuvre la méthode d'Euler et permettant de résoudre numériquement $(\mathscr{E})$ sur le segment $[a,b]$, où $a,b \in \mathbb{R}$, $a < b$. Vous écrirez une docstring décrivant tous les arguments de cette fonction. 1.1

On considère l'équation différentielle d'inconnue $y \in \mathscr{C}^2(\mathbb{R})$ et les conditions initiales suivantes :

$$y'' + \cos(t)y' - t^2 y = \mathrm{e}^t, \qquad y(0) = 4,\ y'(0) = 2. \quad (\mathscr{E})$$

QUESTION Donnez une fonction F et une variable X telle que $(\mathscr{E})$ soit équivalente à l'équation $X' = F(X,t)$. On précisera bien les ensembles de définition et d'arrivée de F, et l'ensemble auquel appartient la variable X, ainsi que la condition initiale à utiliser.

EQD-000

On considère une équation différentielle $(\mathscr{E}) : y' = F(y,t)$, où F est une fonction de $\mathbb{R}^n \times \mathbb{R}$ dans $\mathbb{R}^n$, et où 1.2 l'inconnue y est une fonction de $\mathscr{C}^1(\mathbb{R},\mathbb{R}^n)$, avec la condition initiale $y(t_0) = y_0$.

QUESTION Décrire le principe de la méthode d'Euler. On donnera clairement la relation de récurrence qui est au coeur de cette méthode, en expliquant bien ce que représente la suite vérifiant cette relation de récurrence.

QUESTION Écrire en Python une fonction mettant en œuvre la méthode d'Euler et permettant de résoudre numériquement $(\mathscr{E})$ sur le segment $[a,b]$, où $a,b \in \mathbb{R}$, $a < b$. Vous écrirez une docstring décrivant tous les arguments de cette fonction.

On considère l'équation différentielle d'inconnue $y \in \mathscr{C}^2(\mathbb{R})$ et les conditions initiales suivantes :

$$y'' + \cos(t)y' - t^2 y = \mathrm{e}^t, \qquad y(0) = 4,\ y'(0) = 2. \quad (\mathscr{E})$$

QUESTION Donnez une fonction F et une variable X telle que $(\mathscr{E})$ soit équivalente à l'équation $X' = F(X,t)$. On précisera bien les ensembles de définition et d'arrivée de F, et l'ensemble auquel appartient la variable X, ainsi que la condition initiale à utiliser.

EQD-001

Autour de la dynamique gravitationnelle

Présentation

Modéliser les interactions physiques entre un grand nombre de constituants mène à l'écriture de systèmes différentiels pour lesquels, en dehors de quelques situations particulières, il n'existe aucune solution analytique. Les problèmes de dynamique gravitationnelle et de dynamique moléculaire en sont deux exemples. Afin d'analyser le comportement temporel de tels systèmes, l'informatique peut apporter une aide substantielle en permettant leur simulation numérique. L'objet de ce sujet est l'étude de solutions algorithmiques en vue de simuler une dynamique gravitationnelle afin, par exemple, de prédire une éclipse ou le passage d'une comète.

Quelques fonctions utilitaires

QUESTIONDonner la valeur des expressions python suivantes :

- $[1,2,3]+[4,5,6]$
- $2*[1,2,3]$

QUESTION Écrire une fonction python $smul$ à deux paramètres, un nombre et une liste de nombres, qui multiple chaque élément de la liste par le nombre et renvoie une deuxième liste sans modifier la première. Par exemple, *smul(2, [1, 2, 3])* renvera *[2, 4, 6]*.

QUESTION Déterminer la complexité de cet algorithme en fonction de la taille de la liste.

2 Étude de schémas numériques

Soient y une fonction de classe $\mathscr{C}^2$ sur $\mathbb{R}$ et t_{min} et t_{max} deux réels tels que $t_{min} < t_{max}$. On note I l'intervalle $[t_{min}, t_{max}]$. On s'intéresse à une équation différentielle du second ordre de la forme :

$$\forall t \in I \qquad y''(t) = f(y(t)), \tag{1}$$

où f est une fonction donnée, continue sur $\mathbb{R}$. De nombreux systèmes physiques peuvent être décrits par une équation de ce type.

On suppose connues les valeurs $y_0 = y(t_{min})$ et $z_0 = y'(t_{min})$.

2.1 Mise en forme du problème

Pour résoudre numériquement l'équation différentielle (1), on introduit la fonction $z : I \to \mathbb{R}$ définie par

$$\forall t \in I, z(t) = y'(t). \tag{2}$$

On considère l'équation :

$$Y' = F(Y,t) \tag{3}$$

QUESTION Pour quelle variable Y et quelle fonction F, l'équation (1) peut se mettre sous la forme de l'équation (3) ?

Soit n un entier strictement supérieur à 1 et $J_n = [\![0, n-1]\!]$. On pose $h = \dfrac{t_{max} - t_{min}}{n-1}$ et $\forall i \in J_n$, $t_i = t_{min} + i \cdot h$. On peut montrer que, pour tout entier $i \in [\![0, n-2]\!]$,

$$Y(t_{i+1}) = Y(t_i) + \int_{t_i}^{t_{i+1}} Y'(t)\,dt \tag{4}$$

La suite du problème exploite les notations introduites dans cette partie et présente deux méthodes numériques dans lesquelles l'intégrale précédente est remplacée par une valeur approchée.

2.2 Schéma d'Euler explicite

Dans le schéma d'Euler explicite, chaque terme sous le signe intégrale est remplacé par sa valeur prise en la borne inférieure.

QUESTION Ecrire une fonction $euler(F, tmin, tmax, Y_0, n)$ qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i\in J_n}$, $(z_i)_{i\in J_n}$ ainsi que la liste du temps $(t_i)_{i\in J_n}$.

Pour illustrer cette méthode, on considère l'équation différentielle,

$$\forall t \in I, \quad y''(t) = -\omega^2 y(t) \tag{5}$$

dans laquelle ω est un nombre réel.

QUESTION Expliciter en langage Python la fonction $F(Y,t)$ et la fonction $f(y)$ qui permettront de mettre en oeuvre ce problème.

La mise en oeuvre de la méthode d'Euler explicite génère le résultat graphique donné figure **??**. Dans un système d'unités adapté, les calculs ont été menés en prenant $y_0 = 3$, $z_0 = 0$, $t_{min} = 0$, $t_{max} = 3$, $\omega = 2\pi$ et $n = 100$.

QUESTION Écrire la suite d'instructions permettant à partir de la fonction F et après avoir défini la fonction $euler$ de la mettre en oeuvre et produire le tracé de la figure **??**.

Schéma de Verlet

Le physicien français Loup Verlet a proposé en 1967 un schéma numérique d'intégration d'une équation de la forme (1) dans lequel, en notant $f_i = f(y_i)$ et $f_{i+1} = f(y_{i+1})$, les relations de récurrence s'écrivent, pour tout entier $i \in [\![0, n-2]\!]$:

$$y_{i+1} = y_i + h z_i + \frac{h^2}{2} f_i \quad \text{et} \quad z_{i+1} = z_i + \frac{h}{2}\left(f_i + f_{i+1}\right). \tag{6}$$

QUESTION Écrire une fonction $verlet(f, tmin, tmax, Y$ qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$, ainsi que la liste du temps $(t_i)_{i\in J_n}$.

On reprend l'exemple de l'oscillateur harmonique défini par l'équation (5) et on compare les résultats obtenus à l'aide des schémas d'Euler et de Verlet.

QUESTION Écrire la suite d'instructions permettant à partir de la fonction f et après avoir défini la fonction $verlet$ de la mettre en oeuvre et produire le tracé de la figure **??**.

Comparaison qualitative des schémas numériques.

On rappelle que l'énergie pour un tel oscillateur est proportionnelle à :

$$\mathscr{E} = \frac{1}{2}(y')^2 + \frac{\omega^2}{2} y^2.$$

QUESTION Conclure sur la stabilité des deux schémas numériques.

EQD-001-cor

QUESTION

- [1,2,3,4,5,6];
- [1,2,3,1,2,3]

QUESTION

```
def smul(n,liste):
    """
    Multiplie chaque élément de la liste par un nomb
    """
    liste2=[]
    for i in liste:
        liste2.append(i*n)
    return liste2
```

Cycle 01

QUESTION

On effectue $len(liste)$ tours de boucle. La complexité pour chaque tour de boucle est en $O(1)$ (nombre borné/ fini d'opérations en $O(1)$) Ainsi la complexité est en $O(len(liste))$.

QUESTION

On peut poser :

$$Y = \begin{pmatrix} y(t) \\ z(t) \end{pmatrix}$$

et F peut se définir sous la forme :

$$F : \begin{pmatrix} y(t) \\ z(t) \end{pmatrix} \mapsto \begin{pmatrix} z(t) \\ f(y(t)) \end{pmatrix}$$

QUESTION

```
import numpy as np
def euler(F, tmin, tmax, Y0, n):
    """Solution de Y'=F(Y,t) sur [a,b], y(a) = y0, pas h"""
    Y = Y0
    t = tmin
    les_y = [Y0[0]]
    les_z = [Y0[1]]
    h=(tmax-tmin)/(n-1)
    les_t = np.arange(tmin,tmax,h)
    while t+h <= tmax:
        Y = Y + h * F(Y, t) # surtout pas += !
        les_y.append(Y[0])
        les_z.append(Y[1])
        t += h
    return les_t, les_y, les_z
```

QUESTION

Mathématiquement, on écrit

$$\begin{array}{rccc} f: & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & -\omega^2 x \end{array}$$

$$F : \begin{pmatrix} y \\ z \end{pmatrix} \longmapsto \begin{pmatrix} z \\ f(y) \end{pmatrix}.$$

Informatiquement, cela donne :

```
def f(x):
    return -w**2*x


def F(Y,t):
    y=Y[0]
    z=Y[1]
    return np.array([z,f(y)])
```

QUESTION

```
tmin=0
tmax=3
w=2*np.pi
n=100
y0=3
z0=0
vt,vy,vz=euler(F, tmin, tmax, np.array([y0,z0]), n)

plt.clf()
plt.plot(vy,vz,'ko-')
plt.xlabel('y(t)')
plt.ylabel('z(t)')
plt.grid()
plt.savefig('figure1.png')
```

QUESTION

Comme pour la méthode d'Euler, pour calculer tous les y_i et z_i, il est nécessaire de connaître : f, y_0, z_0, i et h.

```
def verlet(f, tmin, tmax, Y0, n):
    y,z=Y0[0],Y0[1]
    valeursy =[ y ]
    valeursz =[ z ]
    h=(tmax-tmin)/(n-1)
    h2sur2=h**2/2. # Hors de la boucle pour la compl
    les_t = np.arange(tmin,tmax,h)
    for k in range ( n-1 ):
        fi=f(valeursy[-1])
        y = y + h * valeursz[ -1]+h2sur2*fi
        z = z + h/2. * (fi+f(y))
        valeursy . append ( y )
        valeursz . append ( z )
    return [les_t,valeursy , valeursz ]
```

QUESTION

```
vt,vy,vz=verlet(f, tmin, tmax, np.array([y0,z0]), n)

plt.clf()
plt.plot(vy,vz,'ko-')
plt.xlabel('y(t)')
plt.ylabel('z(t)')
plt.axis([-15,20,-80,100])
plt.grid()
plt.savefig('fig2.png')
```

QUESTION Le schéma de Verlet semble plus stable que le schéma d'Euler. En effet l'équation différentielle traduit le comportement d'un système physique conservatif, c'est-à-dire que la quantité $\frac{(y')^2}{2} + \frac{\omega^2}{2} y^2$ doit être constante. On devrait alors avoir un portrait de phase décrivant une ellipse. On observe bien que c'est le cas du portrait de phase de Verlet mais pas celui obtenu avec la méthode d'Euler. La solution approchée donnée par la méthode d'Euler est absurde physiquement : il y aurait augmentation de l'énergie dans le système.

EQD-002

Etude d'un trafic routier

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

3.1 Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure **??**. Une file de longueur n est représentée par n cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure **??**) et sont indifférenciées.

QUESTION Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

QUESTION Donner une ou plusieurs instructions `Python` permettant de définir une liste A représentant la file de voitures illustrée par la Figure **??**.

QUESTION Soit L une liste représentant une file de longueur n et i un entier tel que $0 \leqslant i < n$. Définir en Python la fonction `occupe(L, i)` qui renvoie `True` lorsque la case d'indice i de la file est occupée par une voiture et `False` sinon.

QUESTION Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse.

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

QUESTION Que peut-on dire de la complexité de cette fonction ?

3.2 Base de données

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

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id_croisement_debut, id_croisement_fin)

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

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

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

QUESTION Que renvoie la requête SQL suivante ?

3.3 Simulation dynamique

On introduit les bibliothèques `Math` et `NumPy` à l'aide des lignes suivantes :

```
import math as m
import numpy as np
```

On s'intéresse maintenant à l'apparition des voitures en début de file. On se place alors dans le cas d'un parking lors d'une sortie d'usine entre 17 h et 19 h. Afin de simplifier le problème on considère qu'à $t = 0$, il est 17 h et à $t = 1$, il est 19 h.

Tous les employés de l'usine ne sortent pas au même moment. On donne sur la Figure **??** l'évolution de $Q(t)$ qui représente le nombre de véhicules quittant le stationnement par unité de temps, pour $t \in [0,1]$. On considère que cette fonction est nulle en dehors de cet intervalle. On appelle `db_stat_max` le maximum de Q entre 0 et 1.

$$\forall t \in [0,1],\ Q(t) = \texttt{db\_stat\_max} \cdot \mathrm{e}^{\left(1 - \frac{1}{4t(1-t)}\right)}$$

QUESTION Écrire une fonction `Q(t)` permettant d'obtenir, pour tout réel t, le nombre de véhicules quittant le stationnement par unité de temps. On considère `db_stat_max` comme une variable globale. Faire attention aux cas où `t = 0` et `t = 1`.

QUESTION Écrire une fonction `integrale(f, a, b, n)` permettant, avec la méthode des trapèzes, d'estimer l'intégrale de `f` entre `a` et `b` à partir de `n` points équirépartis.

QUESTION On appelle $N(t)$ le nombre de véhicules ayant quitté le stationnement jusqu'à l'instant t. Écrire une fonction `Nb(t, n)` renvoyant une valeur approchée de ce nombre, en utilisant n points.

La suite de cette partie permettant d'obtenir $N(t)$ par une autre méthode, la fonction `Nb` ne sera plus utilisée.

La vitesse de sortie des véhicules est limitée par le nombre de véhicules sortants. En effet, quand peu de véhicules sortent du parking, ces derniers peuvent circuler à vitesse maximale V_{max}. En revanche, lorsque beaucoup de véhicules sont en mouvement, ces derniers se ralentissent les uns les autres. On appelle K la concentration de véhicules cherchant à rejoindre la sortie du parking à un instant donné et K_{sat} le nombre de véhicules à partir duquel la vitesse de sortie est minimale (V_{min}).

L'expression de la vitesse des véhicules V en fonction de $K \in [0, K_{sat}]$ est donnée par

$$V(K) = \frac{V_{max} - V_{min}}{2} \cdot \left(1 + \cos\left(\pi \cdot \frac{K}{K_{sat}}\right)\right) + V_{min}.$$

QUESTION Écrire une fonction `V(K)` prenant en entrée un flottant `K` et renvoyant la valeur de la vitesse correspondante. On considère `Vmin`, `Vmax` et `Ksat` comme des variables globales.

La conservation du nombre total de véhicules conduit au système différentiel suivant :

$$\frac{\mathrm{d}N}{\mathrm{d}t}(t) = Q(t) \tag{7}$$

$$\frac{\mathrm{d}K}{\mathrm{d}t}(t) = Q(t) - K(t) \cdot V(K) \tag{8}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t}(t) = K(t) \cdot V(K) \tag{9}$$

où $S(t)$ désigne le nombre de véhicules sortis du parking à l'instant t.

QUESTION Rappeler la relation de récurrence de la méthode d'Euler explicite pour un problème défini par son équation différentielle $y'(t) = F(y(t), t)$ avec $h = \Delta t$ le pas de temps supposé constant.

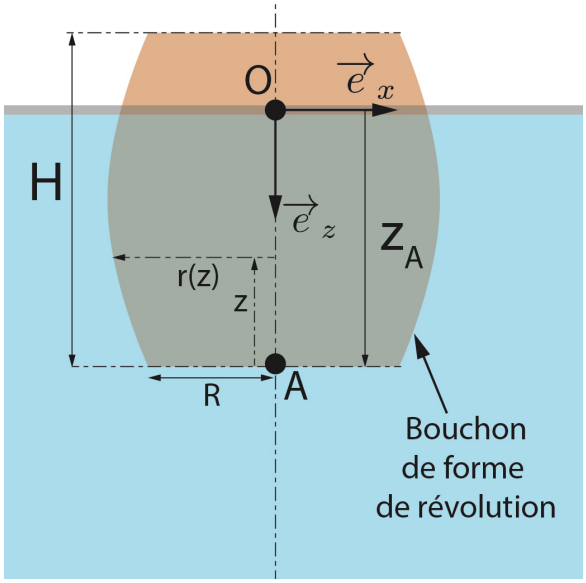
On considère que cette méthode est implémentée dans une fonction `odeint(F, Y0, T)` où `F` est la fonction associée au problème de Cauchy, `Y0` est un tableau

`NumPy` contenant les valeurs initiales de la fonction Y et `T` un tableau `NumPy` contenant les différents pas de temps.

QUESTION Commenter en quelques lignes les allures des courbes de la Figure **??**.

EQD-003

Ce sujet concerne la dynamique d'un bouchon en liège flottant dans un verre d'eau (voir figure suivante).



On note :

- R le rayon de la base du bouchon ;
- ρ_e la masse volumique de l'eau ;
- ρ_b la masse volumique du bouchon ;
- $z_A(t)$ la position verticale du bas du bouchon selon la direction $\vec{e}_z$ qui dépend du temps t par rapport à O (point situé sur la surface de l'eau) ;
- H la hauteur du bouchon.

Le bouchon possède une symétrie de révolution. Ainsi, son rayon, noté $r(z)$, dépend de la coordonnée z de la manière suivante :

$$\begin{array}{rccl} r: & [0,H] & \to & \mathbb{R}, \\ & z & \mapsto & R\cdot\left[1+0,1\cdot\sin\left(\pi\cdot\dfrac{z}{H}\right)\right]. \end{array}$$

Le volume immergé du bouchon dépend de la position z_A et est donné par la relation suivante :

$$V_i(z_A)=\begin{cases} 0 \quad \text{si} \quad z_A<0\,; \\ \pi\cdot\displaystyle\int_{z=0}^{z_A}(r(z))^2\,\mathrm{d}z \quad \text{si} \quad 0\le z_A\le H\,; \\ \pi\cdot\displaystyle\int_{z=0}^{H}(r(z))^2\,\mathrm{d}z \quad \text{si} \quad z_A>H. \end{cases}$$

On note V le volume total du bouchon correspondant à $V_i(H)$.

Le théorème de la résultante dynamique appliqué au bouchon selon la direction $\vec{e}_z$ donne :

$$M\cdot\frac{\mathrm{d}^2z_A}{\mathrm{d}t^2}(t)=P-F_p(z_A(t))+F_v\left(z_A(t),\frac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right). \quad (\mathscr{E})$$

Les grandeurs suivantes interviennent dans l'équation $(\mathscr{E})$.

- $M=\rho_b\cdot V$, la masse du bouchon (égale au produit de sa masse volumique avec son volume).
- $P=\rho_b\cdot V\cdot g$, le poids du bouchon.
- g, l'accélération de la pesanteur.
- F_p, la force de poussée. Elle est égale au produit de la masse volumique de l'eau (ρ_e) avec g et du volume immergé ($V_i(z_A(t))$), *i.e.*

$$F_p(z_A(t))=\rho_e\cdot g\cdot V_i(z_A(t)).$$

- $F_v\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)$ est la force de frottement visqueux. Elle est donnée par la formule

$$F_v\left(z_A(t),\frac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=-\frac{1}{2}\rho_e\cdot C_x\cdot S\left(z_A(t),\frac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)\cdot\left(\frac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)^2.$$

- $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)$ est la surface apparente du bouchon vis-à-vis du fluide et est définie de la manière suivante.
  - Si le bouchon est totalement hors de l'eau, cette surface est nulle.
  - Si le bouchon descend et est (au moins partiellement) immergé :
    si $0\le z<H/2$, $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=\pi\cdot r(z)^2$,
    si $z\ge H/2$, $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=\pi\cdot r(H/2)^2$.
  - Si le bouchon remonte et est (au moins partiellement) immergé :
    si $0\le z<H/2$, $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=0$.
    si $H/2\le z\le H$, la surface à prendre en compte est celle de la couronne : $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=-\pi\cdot\left(r(H/2)^2-r(z)^2\right)$
    si $z>H$, $S\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)=-\pi\cdot r(H/2)^2$.
    La force de frottement s'opposant au mouvement, si le bouchon remonte, la résultante de cette force selon la direction $\vec{e}_z$ est positive, d'où le signe $-$ placé ici.
- C_x est le coefficient de trainée aérodynamique.

Dans toute la suite de ce devoir, on pourra supposer que les grandeurs suivantes (et uniquement celles-ci) ont été définies.

```
R = 1E-2 # m, rayon minimum du bouchon de révolution
H = 4.5*1E-2 # m, hauteur du bouchon
rho_eau = 1000 # kg / m**3 masse volumique de l'eau
rho_b = 240 # kg / m**3, masse volumique du liège
g = 9.81 # m / s**2, accélération de la pesanteur en
Cx = 1 # Coefficient de trainée aérodynamique
N = 1000 # Nombre de trapèzes pour les calculs d'inté
```

QUESTION Écrire la fonction `T(f,a,b,N)` permettant de donner l'estimation de $\int_{z=a}^{b} f(z)\mathrm{d}z$ par la méthode des trapèzes, avec N trapèzes sur le segment $[a,b]$.

QUESTION Écrire une fonction `volume_immerge(z)` qui renvoie le volume immergé du bouchon en fonction de la profondeur du bas du bouchon (noté z) en utilisant la fonction définie à la question précédente.

QUESTION Écrire l'instruction permettant de calculer le volume total du bouchon, que l'on affectera à la variable `V`.

QUESTION Écrire une fonction `Fv(z,zp)` qui renvoie la force de frottement visqueux $F_v\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)$.

QUESTION Exprimer $\dfrac{\mathrm{d}^2 z_A}{\mathrm{d}t^2}(t)$ en fonction de ρ_e, ρ_b, $V_i(z_A(t))$, $F_v\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)$ et $z_A(t)$.

On souhaite résoudre cette équation différentielle (équation $\mathscr{E}$) avec pour conditions initiales :

$$\begin{cases} z_A(t=0) = -0,2 \\ \dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t=0) = 0 \end{cases} \qquad \text{(CI)}$$

QUESTION Définir l'expression de $Z(t)$, Z_0 et de $F(Z(t),t)$ pour que l'équation différentielle ($\mathscr{E}$) avec les conditions initiales (CI) soit équivalente au problème de Cauchy d'ordre 1 :

$$Z'(t) = F(Z(t),t) \quad \text{et} \quad Z(0) = Z_0. \qquad (\mathscr{F})$$

Écrire une suite d'instructions permettant de définir une telle fonction `F(Z,t)` ainsi que la condition initiale `Z0`.

QUESTION Écrire une fonction `euler(F, tmin, tmax, Z0, h)` prenant en argument la fonction `F` définie précédemment, `tmin` et `tmax` définissant l'intervalle de résolution, le vecteur `Z0` définissant les consitions initiales ainsi que `h` le pas de discrétisation temporelle et permettant de résoudre de manière approchée l'équation ($\mathscr{F}$) par la méthode d'Euler.

On donne sur la figure **??** le résultat de l'application de la méthode d'euler pour simuler le comportement de la chute libre d'un bouchon à partir de 20 cm au dessus du niveau de l'eau.

QUESTION Commenter brièvement la cohérence physique d'une telle courbe. Pouvez-vous fournir une explication et une solution au problème observé, tout en restant dans le cadre de la méthode d'Euler ?

EQD-003-cor

QUESTION

```
def T(f,a,b,N):
    h=(b-a)/N
    S=(f(a)+f(b))*0.5
    for k in range(1,N):
        S+=f(a+k*h)
    return S*h
```

QUESTION

```
from math import sin, pi

def r(z):
    return R*(1+0.1*sin(np.pi*z/H))

def r2(z):
    return r(z)**2

def volume_immerge(z):
    """Renvoie le volume immergé du bouchon.
    z : profondeur du bas du bouchon"""
    if z <= 0 :
        # bouchon hors de l'eau
        return 0
    elif z <= H :
        # bouchon immergé jusqu'à la hauteur z
        return pi*T(r2,0,z,N)
    else :
        # bouchon totalement immergé
        return pi*T(r2,0,H,N)
```

QUESTION

```
V = volume_immerge(H)
```

QUESTION

```
def Fv(z,zp):
    """renvoie la force de frottement visqueux"""
    if zp==0 or z<0:
        return 0
    else:
        if zp>0 :
            signe = 1
        else :
            signe = -1
        alpha = -signe*0.5*Cx*rho_eau*zp**2
        if zp>0 : # Le bouchon descend
            if 0<=z<H/2:
                return alpha*pi*r(z)**2
            else: # z >= H/2
                return alpha*pi*r(H/2)**2
        else s: #Le bouchon remonte
            if 0<=z<H/2:
                return 0
            elif H/2<=z<=H:
                return alpha*pi*(r(H/2)**2-r(z)**2)
            else : # z>H:
                return alpha*pi*r(H/2)**2
```

QUESTION

En combinant les différentes équations, on obtient :

$$\rho_b \cdot V \cdot \frac{\mathrm{d}^2 z_A}{\mathrm{d}t^2}(t) = -\rho_e \cdot g \cdot V_i(z_A(t)) + \rho_b \cdot g \cdot V + F_v\left(z_A(t),\frac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right)$$

En simplifiant les équations, on obtient :

$$\frac{\mathrm{d}^2 z_A}{\mathrm{d}t^2}(t) = \frac{F_v\left(z_A(t),\dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t)\right) - \rho_e \cdot g \cdot V_i(z_A(t))}{\rho_b \cdot V} + g$$

QUESTION
On peut poser

$$Z(t)=\begin{pmatrix} z_A(t) \\ \dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t) \end{pmatrix} \quad \text{et} \quad Z_0=\begin{pmatrix} -0,2 \\ 0 \end{pmatrix},.$$

Ainsi,

$$Z'(t)=\begin{pmatrix} \dfrac{\mathrm{d}z_A}{\mathrm{d}t}(t) \\ \dfrac{\mathrm{d}^2z_A}{\mathrm{d}t^2}(t) \end{pmatrix}$$

Avec

$$F\left(\begin{pmatrix} a \\ b \end{pmatrix},t\right)=\begin{pmatrix} b \\ g+\dfrac{1}{\rho_b V}\left(F_v(a,b)-\rho_e g V_i(a)\right) \end{pmatrix},$$

on a bien

$$Z'(t)=F(Z(t),t) \quad \text{et} \quad Z(0)=Z_0.$$

```
from numpy import array

def F(Z,t):
    """Avec frottement visqueux"""
    z,zp = Z
    zpp = (Fv(z,zp)- rho_eau*g*volume_immerge(z)) / (rho_b*volume_immerge(H)) + g
    return array([zp,zpp])

Z0 = array([-0.2,0])
```

QUESTION

```
def euler(F, tmin, tmax, Z0, h):
    """Solution de Z'=F(Z,t) sur [tmin,tmax], Z(tmin)
    Z = Z0
    t = tmin
    z_list = [Z0] # la liste des valeurs renvoyées
    t_list = [tmin] # la liste des temps
    while t+h <= tmax :
        # Variant : floor((tmax-t)/h)
        # Invariant : au tour k, z_list = [Z_0,...,Z
        Z = Z + h * F(Z, t)
        z_list.append(Z)
        t = t + h
        t_list.append(t)
    return t_list, z_list
```

QUESTION Le terme de frottement fluide induit une perte d'énergie pour le bouchon. Après la phase de chute libre, les oscillations du bouchon dans l'eau devraient donc décroître en amplitude. Ce n'est pas ce que l'on observe (la second oscillation a même une amplitude légèrement plus grande que la première !).

Une explication : le pas temporel choisi pour la méthode d'Euler est trop grand. Il suffit de le diminuer.