Lycée Faidherbe, 2020-21

Devoirs d'informatique

PC2

Version du 21 février 2021

Table des matières

DS I

MOYENNISATION

Les 3 parties de ce devoir sont indépendantes.

- La première partie porte sur de l'algorithmique simple de première année
- La deuxième partie étudie une modélisation de chimie, elle correspond au programme du second semestre de sup.
- La troisième partie reprend des calculs d'algorithmes en introduisant quelques calculs de complexité.

Il est rappelé qu'il est toujours possible d'utiliser une fonction demandée dans une question pour l'écriture d'autres solutions **même si on n'a pas écrit cette fonction.**

1 Droites d'approximations

1.1 Fonctions classiques

Exercice 1 — Moyenne
Écrire une fonction `moyenne(X)` *qui calcule la moyenne des termes de la liste* X *:*
`moyenne(X)` $= \dfrac{1}{n}\displaystyle\sum_{i=0}^{n}$ `X[i]` *avec* $n =$ `len(X)`.

La **variance** d'une liste est définie par $\text{var}(X) = \dfrac{1}{n}\displaystyle\sum_{i=0}^{n}\big(\texttt{X[i]} - m_X\big)^2$ avec m_X égal à la moyenne de la liste et $n =$ `len(X)`.

Exercice 2 — Variance
Écrire une fonction `var(X)` *qui calcule la variance de la liste* X.

1.2 Droite de régression

Si X et Y sont deux listes de même longueur, la **covariance** de X est Y est
$\text{cov}(X,Y) = \dfrac{1}{n}\displaystyle\sum_{i=0}^{n}\big(\texttt{X[i]} - m_X\big)\big(\texttt{Y[i]} - m_Y\big)$ avec m_X égal à la moyenne de la liste X, m_Y égal à la moyenne de la liste Y et n est la longueur commune de X et de Y.
On pourra noter que $\text{var}(X) = \text{cov}(X,X)$

Exercice 3 — Covariance
Écrire une fonction `cov(X, Y)` *qui calcule la covariance des listes* X *et* Y.

Si X et Y sont deux listes de même longueur, leur **produit** est la liste Z de même longueur telle que `Z[i] = X[i]*Y[i]` pour tout indice i compris entre 0 et $n-1$.

Exercice 4 — Produit
Écrire une fonction `produit(X, Y)` *qui calcule la liste produit des listes X et Y.*

On admet que, si Z est le produit des listes X et Y, alors $\text{cov}(X,Y) = m_Z - m_X.m_Y$ où m_X, m_Y et m_Z sont les moyennes respectivement des listes X, Y et Z.

Exercice 5 — Calcul alternatif
Écrire une fonction `cov(X, Y)` *qui calcule la covariance des listes X et Y sans utiliser de boucles mais en employant les fonctions* `moyenne` *et* `produit`*.*

On admet que, X et Y sont deux listes de même longueur, alors la **droite de régression** approchant Y par rapport à X est la droite d'équation $y = ax + b$ avec $m_Y = a.m_X + b$ et $a.\text{var}(X) = \text{cov}(X,Y)$.

Exercice 6 — Droite de régression
Écrire une fonction `regression(X, Y)` *qui calcule les coefficientsa et b de la droite de régression de X à Y.*

1.3 Méthode de Wald

Il existe d'autres droites qui permettent d'approcher les variations de Y par rapport à X.
La méthode de Wald consiste à

- séparer les listes en deux parties quasi-égales : X donne Xg et Xd et Y donne Yg et Yd,
- calculer les moyennes des quatre parties,
- déterminer la droite $y = ax + b$ telle que
  `moyenne(Yg) = a*moyenne(Xg) + b` et `moyenne(Yd) = a*moyenne(Xd) + b`

On suppose que la liste $X = [x_0, x_1, x_2, \ldots, x_{n-1}]$ est croissante.
Si X comporte n éléments et si $m = \left\lfloor \frac{n}{2} \right\rfloor$ (`m = n//2`), on définit
$Xg = [x_0, x_1, x_2, \ldots, x_{m-1}]$ et $Xd = [x_m, x_{m+1}, \ldots, x_{n-1}]$.
On sépare alors Y selon le même indice : `Yg = Y[0 : m]` et `Yd = Y[m : n]`.
Ainsi les sous-suites de droite ont autant d'éléments que les sous-suites de gauche ou un de plus.

Exercice 7 — Séparation
Écrire en Python une fonction `separer(U)` *qui détermine les deux listes définies en séparant comme ci-dessus.*

Exercice 8 — Droite de Wald
Écrire en Python une fonction `wald(X,Y)` *qui, à partir de deux listes de même longueur* `X` *et* `Y`*, calcule le couple* `(a,b)` *des coefficients a et b déterminés ci-dessus.*

La troisième partie donnera d'autres calculs de droites.

2 Phosgène

Le phosgène $COCl_2$ est gaz qui a servi de gaz de combat pendant la première guerre mondiale.
Il est utilisé maintenant dans de nombreuses synthèses, en particulier des isocyanates.
A haute température il se décompose selon $COCl_2 \longrightarrow CO + Cl_2$.
L'étude du mécanisme indique que la réaction est d'ordre 1 en $COCl_2$ et d'ordre $\frac{1}{2}$ en Cl_2.
On note $y(t)$ la concentration en Cl_2 au temps t.
La concentration de phosgène introduite dans le gaz n'est pas constante : elle est donnée par une loi $\alpha(t)$. On aboutit alors à l'équation différentielle

$$(E) \qquad y'(t) = k\big(\alpha(t) - y\big)\sqrt{y} = \varphi(y,t)$$

2.1 Premières fonctions

Exercice 9
Écrire une fonction `listeT(a, b, n)` *qui calcule une liste de longueur n dont les valeurs sont équiréparties entre a et b.*

`listeT(1, 3, 5)` doit renvoyer `[1.0, 1.5, 2.0, 2.5, 3.0]`.
On remarquera que le pas est $\frac{b-a}{n-1}$.

On suppose que la fonction α est définie par la formule

$$\alpha(t) = \begin{cases} \alpha_0 + ct & \text{pour } t < T_s \\ \alpha_0 + c.T_s & \text{pour } t \geqslant T_s \end{cases}$$

Les constantes $k = 2,4.10^{-2}$ mole$^{-1/2}$ L$^{1/2}$ s^{-1}, $\alpha_0 = 10^{-8}$ mole L^{-1}, $c = 0.2$ L^{-1} s^{-1}
et $T_s = 10$ s (le seuil) sont données par les variables globales

```
k = 2.4e-2
alpha0 = 1e-8
c = 0.2
T_s = 10
```

On utilisera les variables dans les fonctions à écrire plutôt que leurs valeurs.

Exercice 10
Écrire la fonction `alpha(t)` *qui calcule la valeur de* $\alpha(t)$ *en fonction du temps t.*

Exercice 11
Écrire la fonction `phi(x, t)` *qui définit l'équation* (E) *; elle traduit* φ *en python.*

2.2 Méthode d'Euler

On rappelle que la résolution de l'équation nécessite
- la fonction `phi(y, t)`,
- une liste `T` de valeurs de t,
- la condition initiale `y0` qui est la valeur de la solution recherchée au temps `T[0]`.

La solution sera donnée sous la forme d'une liste `Y` de même taille que `T` et telle que `Y[i]` approche la valeur de la solution en `T[i]`.

Exercice 12
Écrire la fonction `euler(phi, y0, T)` *qui calcule une solution en utilisant la méthode d'Euler.*

Exercice 13
Donner les instructions qui permettent de calculer une solution sur l'intervalle de temps $[0; 200]$ *avec une condition initiale* $y_0 = \frac{1}{2}\alpha_0$. *On choisira une liste de temps avec 1000 points.*

Exercice 14
Pourquoi a-t-on choisi une condition initiale non nulle ? On pourra trouver une solution très simple si $y_0 = 0$.

2.3 Schéma du point médian

On suppose dans cette partie que les temps t_i sont tels que le pas $t_{i+1} - t_i$ est constant ; on note h sa valeur. On notera que c'est le cas pour les listes `listeT(a, b, n)` où on a $h = \frac{b-a}{n-1}$
On rappelle que le schéma d'Euler consiste, pour une fonction f de classe $\mathcal{C}^1$,
à approcher $f(t_{i+1})$ par $f(t_i) + (t_{i+1} - t_i)f'(t_i) = f(t_i) + hf'(t_i)$.

Exercice 15 — Des maths simples
En écrivant un développement limité à l'ordre 2 de f *pour calculer* $f(t_{i+1})$ *en fonction de* $f(t_i)$, $f'(t_i)$ *et* $f''(t_i)$, *montrer que l'erreur commise par l'approximation du schéma d'Euler peut s'écrire*

$$f(t_{i+1}) - f(t_i) - hf'(t_i) = A_2h^2 + o\big(h^2\big)$$

Exercice 16 — Un peu de calcul mathématique
En écrivant deux développementx limitéx à l'ordre 3 de f *pour calculer* $f(t_{i+1})$ *et* $f(t_{i-1})$ *en fonction des valeurs en* t_i, *montrer qu'on a*

$$f(t_{i+1}) - f(t_{i-1}) - 2hf'(t_i) = A_3h^3 + o\big(h^3\big)$$

On rappelle qu'on a $t_{i-1} - t_i = -h$

Ainsi l'approximation de $f(t_{i+1})$ par $f(t_{i-1}) + 2hf'(t_i)$ semble permettre des calculs plus précis pour des petites valeurs du pas.
On notera cependant que cette méthode ne peut pas s'appliquer pour évaluer $f(t_1)$.
On propose donc le schéma de calcul suivant pour approcher les valeurs des $f(t_i)$ par y_i.
- On a la valeur de y_0.
- On calcule y_1 par le schéma d'Euler : $y_1 = y_0 + hf'(t_0) = y_0 + h\varphi(y_0, t_0)$.
- Pour i prenant les valeurs de 1 à $n - 2$, on calcule
- $y_{i+1} = y_{i-1} + 2h\varphi(y_i, t_i)$.

Exercice 17
Écrire la fonction `median(phi, y0, T)` *qui calcule une solution approchée en utilisant ce schéma. On devra calculer le pas* h **avant** *la boucle* `for` *:* $h = t_1 - t_0$.

La figure ci-après compare les solutions obtenues par les deux méthodes. On constate une amélioration de l'approximation avec, cependant, une instabilité des valeurs visible autour de $t = 200$. Cette instabilité ne permet pas la généralisation de cette méthode.

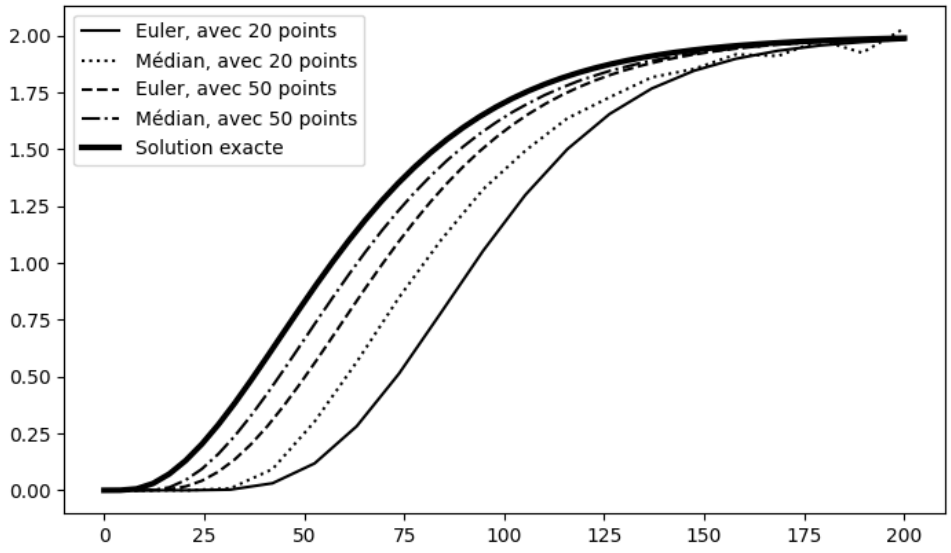


Figure I.1 – Solutions comparées

2.4 Dépassement

On demande parfois le moment où la concentration atteint une valeur donnée.
On suppose que la liste `Y` est triée par ordre croissant.
On se donne un seuil s strictement compris entre les valeurs extrêmes de la liste `Y`, par exemple $s = 1.2$ dans les solutions ci-dessus et on cherche l'indice i tel que `Y[i] <= s < Y[i+1]`

Exercice 18
Écrire une fonction `indiceSeuil(Y, s)` *qui calcule cet indice.*

Exercice 19
Modifier en une fonction `tempsSeuil(T, Y, s)` *qui renvoie le temps* `T[i]` *où se produit le dépassement.*

Si on ne suppose pas que le seuil est franchi on peut vouloir cependant avoir un résultat.

Exercice 20
Modifier la fonction `indiceSeuil(T, Y, s)` *pour qu'elle renvoie*

- *l'indice i tel que* `Y[i] <= s < Y[i+1]` *si cet indice existe,*
- *-1 si on a* `s < Y[0]`,
- *n-1 si on a* `Y[n-1] > = s` *(n est la longueur de la liste* `Y`*.)*

3 Médiane

On revient à la problématique de la première partie pour rechercher une droite qui approche au mieux un ensemble de points défini par les liste `X` et `Y` ; pour simplifier les écriture on notera x_i (respectivement y_i) la valeur de `X[i]` (respectivement de `Y[i]`).
La méthode de Wald décrite ci-dessus comporte le calcul des moyennes mais on sait que la moyenne, dans des résultats expérimentaux, peut être est perturbée par la présence de points aberrants.
Un moyen classique de gérer ces points est de considérer la médiane plutôt que la moyenne.

Définition :
La médiane d'une liste de points $U = (u_0u_1, u_2, \ldots, u_{n-1})$ est une valeur qui partage la liste en deux parties quasi- égales.

3.1 Listes sans points doubles

Dans le cas d'une liste de points **distincts** on trie la liste en $(v_0, v_1, \ldots, v_{n-1}$

1. Si n est impair, $n = 2p + 1$, la médiane est v_p : il y a p éléments qui lui sont strictement inférieurs et p éléments qui lui sont strictement supérieurs.
2. si, n est pair, $n = 2p$, la médiane peut être
   v_{p-1} : il y a $p - 1$ éléments strictement inférieurs et p éléments strictement supérieurs
   ou v_p : il y a p éléments strictement inférieurs et $p - 1$ éléments strictement supérieurs.
   On choisira v_p comme médiane dans ce cas.

On ne suppose pas que les listes sont triées.

Exercice 21
Montrer que, pour une liste de points distincts et de longueur n, il y a `n//2` *éléments strictement supérieurs à la médiane et* `(n-1)//2` *éléments strictement inférieurs à la médiane.*

Pour trouver la médiane d'une liste de points distincts il suffit donc de trouver l'élément de la liste qui admet exactement `(n-1)//2` éléments strictement inférieurs dans la liste.

Exercice 22
Écrire une fonction `rang(x, L)` *qui renvoie le nombre d'éléments de la liste L qui sont strictement inférieurs à x. On pourra parcourir la liste et augmenter un compteur pour chaque élément de la liste strictement inférieur à x.*
Combien de comparaisons sont-elles effectuées ?

Exercice 23
En déduire une fonction `mediane(L)` *qui renvoie la médiane d'une liste supposée non vide et composée d'éléments distincts.*
Combien de comparaisons sont-elles effectuées dans le meilleur des cas (la médiane est `L[0]`*) et dans le pire des cas ?*

3.2 Listes générales

Dans cette partie les listes peuvent contenir des points doubles ; on rappelle que la liste n'est pas supposée triée. Le nombre d'éléments dans la liste est noté n et on pose $n_1 = \lfloor \frac{n-1}{2} \rfloor$ (`n1 = (n-1)//2`) et $n_2 = \lfloor \frac{n}{2} \rfloor$ (`n2 = n//2`).
On admet que dans le cas il existe une valeur unique x dans la liste qui verifie (P) où (P) est la propriété
Il existe au plus n_1 éléments strictement inférieurs à x dans la liste et il existe au plus n_2 éléments strictement supérieurs à x dans la liste.
Il se peut que propriété (P) soit vérifiée pour plusieurs indices dans la liste.

Exercice 24
Écrire une fonction `avantApres(x, L)` *qui renvoie le couple* (p, q) *avec*
p *le nombre d'éléments de la liste* L *qui sont strictement inférieurs à* x,
q *le nombre d'éléments de la liste* L *qui sont strictement supérieurs à* x.

Exercice 25
Écrire une fonction `test_mediane(x, L)` *qui renvoie* `True` *ou* `False` *selon que* x *vérifie ou non la propriété* (P).

Exercice 26
En déduire une fonction `mediane(L)` *qui renvoie la médiane d'une liste supposée non vide.*

3.3 Calcul plus rapide

On généralise le calcul de la médiane : on va chercher l'élément d'indice k dans l'ordre croissant dans une liste.
Voici un algorithme rapide qui pourrait être utilisé pour calculer la médiane en cherchant pour $k = \lfloor \frac{n}{2} \rfloor$.

```
def choisir(k, liste):
    """On choisit le k-ième élément
    dans une liste de n éléménts avec n >=k"""
    copier liste dans L
    tant qu'on n'a pas trouvé
        choisir un élément au hasard dans L noté c
        séparer la liste L en petit, egal et grand selon c
        calculer la longueur de petit : p
        calculer la longueur de egal : q
        si p > k : copier petit dans L
        si p <= k < p + q : renvoyer c
        si p + q <= k : copier grand dans L
                        et poser k = k - p - q
```

La séparation de la liste selon c place les éléments de la liste `L` dans 3 listes :
- `petit` contient les éléments `L[i]` strictement inférieurs à c
- `egal` contient les éléments `L[i]` égaux à c
- `grand` contient les éléments `L[i]` strictement supérieurs à c

Exercice 27
Écrire une fonction `petitTrir(L, c)` *qui renvoie les 3 listes ci-dessus.*

On rappelle que la copie de liste se fait en utilisant la fonction `deepcopy`.

```
from copy import deepcopy
```

Exercice 28
Écrire une fonction `choisir(k, liste` *qui recherche l'élément de rang* k *dans la liste en appliquant l'algorithme ci-dessus.*

3.4 Modèle robuste de Tukey

La méthode de Tukey repose, par amélioration successives, sur les idées de Wald.
Pour calculer la droite "robuste" :

1. on sépare les listes en trois parties quasi-égales selon les indices.
   X est séparée en X_1, X_2 et X_3, Y est séparée en Y_1, Y_2 et Y_3 avec X_1 et Y_1 de même longueur ainsi que X_2 et Y_2 et que X_3 et Y_3
2. On calcule les médianes des 6 parties, $x_1, x_2, x_3, y_1, y_2, y_3$. .
3. On considère la droite passant par (x_1, x_1) et (x_3, y_3) d'équation $y = ax + b_1$.
4. On considère la droite de même pente a passant par (x_2, y_2) d'équation $y = ax + b_2$.
5. La droite de Tukey admet $y = ax + b$ pour équation avec $b = \dfrac{2b_1 + b_2}{2}$.

Exercice 29
Écrire une fonction `separer3(U)` *qui détermine trois listes définies en séparant les indices pour donner des listes dont les longueurs sont égales ou diffèrent de 1.*

Exercice 30
En déduire une fonction `robuste((X,Y)` *qui, à partir de deux listes de même longueur* `X` *et* `Y`*, calcule le couple* `(a,b)` *des coefficients a et b déterminés ci-dessus.*

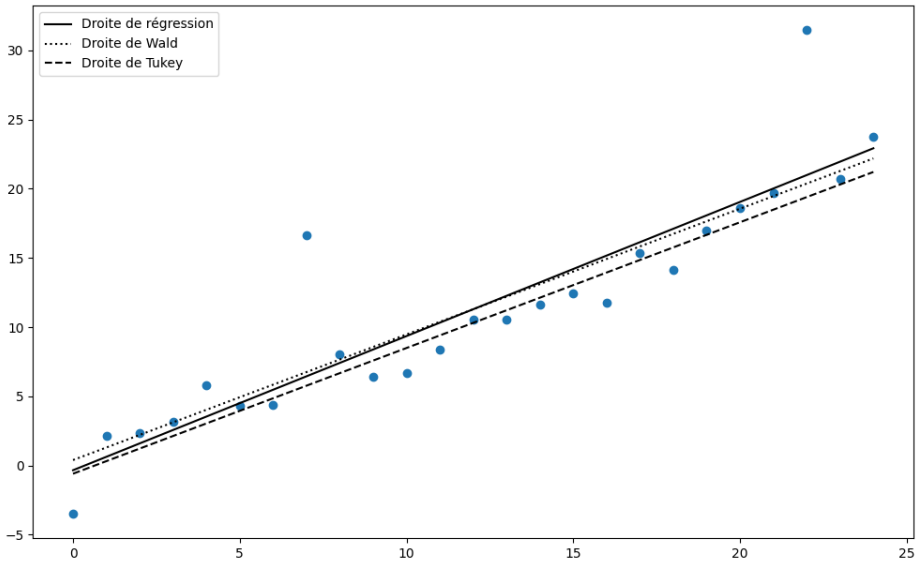


Figure I.2 – Exemple des 3 droites

4 Solutions

Solution de l'exercice 1 -

```
def moyenne(X):
    n = len(X)
    somme = 0
    for i in range(n):
        somme = somme + X[i]
    return somme/n
```

Solution de l'exercice 2 -

```
def var(X):
    n = len(X)
    mX = moyenne(X)
    somme = 0
    for i in range(n):
        somme = somme + (X[i] - mX)**2
    return somme/n
```

Solution de l'exercice 3 -

```
def cov(X, Y):
    n = len(X)
    mX = moyenne(X)
    mY = moyenne(Y)
    somme = 0
    for i in range(n):
        somme = somme + (X[i] - mX)*(Y[i] - mY)
    return somme/n
```

Solution de l'exercice 4 -

```
def produit(X, Y):
    n = len(X)
    Z = [0]*n
    for i in range(n):
        Z[i] = X[i]*Y[i]
    return Z
```

Solution de l'exercice 5 -

```
def cov(X, Y):
    n = len(X)
    mX = moyenne(X)
    mY = moyenne(Y)
    Z = produit(X, Y)
    mZ = moyenne(Z)
    return mZ - mX*mY
```

Solution de l'exercice 6 -

```
def regression(X, Y):
    a = cov(X, Y)/var(X)
    b = moyenne(Y) - a*moyenne(X)
    return a, b
```

Solution de l'exercice 7 -

```
def separer2(U):
    n = len(U)
    m = n//2
    return U[ : m], U[m : ]
```

Solution de l'exercice 8 -

```
def wald(X, Y):
    Xg, Xd = separer2(X)
    Yg, Yd = separer2(Y)
    mXg = moyenne(Xg)
    mXd = moyenne(Xd)
    mYg = moyenne(Yg)
    mYd = moyenne(Yd)
    a = (mYd - mYg)/(mXd - mXg)
    b = mYd - a*mYd
    return a, b
```

Solution de l'exercice 9 -

```
def listeT(a, b, n):
    pas = (b-a)/(n-1)
    T = [0]*n
    for k in range(n):
        T[k] = a + k*pas
    return T
```

Solution de l'exercice 10 -

```
def phi(x, t):
    return k*(alpha*t - x)*x**0.5
```

Solution de l'exercice 11 -

```
def alpha(t):
    if t < Ts:
        return alpha0 + c*t
    else:
        return alpha0 + c*Ts
```

Solution de l'exercice 12 -

```
def euler(phi, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        pas = T[i+1] - T[i]
        pente = phi(Y[i], T[i])
        Y[i+1] = Y[i] + pas*pente
    return Y
```

Solution de l'exercice 13 -

```
y0 = alpha0/2
t_max = 200
n = 1000
T = listeT(0, t_max, n)
Y = euler(phi, y0, T)
```

Solution de l'exercice 14 - Pour $y_0 = 0$, une solution est la fonction nulle, on veut voir évoluer le système.

Solution de l'exercice 15 - $f(t_{i+1}) - f(t_i) - hf'(t_i) = \frac{f''(t_i)}{2}h^2 + o\big(h^2\big)$

Solution de l'exercice 16 - $f(t_{i+1}) = f(t_i) + hf'(t_i) + \frac{f''(t_i)}{2}h^2 + \frac{f^{(3)}(t_i)}{6}h^3 + o\big(h^3\big)$
$f(t_{i+1}) = f(t_i) - hf'(t_i) + \frac{f''(t_i)}{2}h^2 - \frac{f^{(3)}(t_i)}{6}h^3 + o\big(h^3\big)$
d'où $f(t_{i+1}) - f(t_{i-1}) = 2hf'(t_i) + \frac{f^{(3)}(t_i)}{3}h^3 + o\big(h^3\big)$.

Solution de l'exercice 17 -

```
def median(phi, y0, T):
    n = len(T)
    Y = [0]*n
    h = T[1] - T[0]
    Y[0] = y0
    Y[1] = y0 + h*phi(y0, T[0])
    for i in range(2, n-1):
        pente = phi(Y[i], T[i])
        Y[i+1] = Y[i-1] + 2*h*pente
    return Y
```

Solution de l'exercice 18 - On balaye la liste.

```
def indiceSeuil(Y, k):
    i = 0
    while liste[i+1] <= s:
        i = i+1
    return i
```

Solution de l'exercice 19 -

```
def tempsSeuil(T, Y, s):
    i = 0
    while liste[i+1] <= s:
        i = i+1
    return T[i]
```

Solution de l'exercice 20 -

```
def indiceSeuil(T, Y, s):
    i = -1
    while i < n-1 and liste[i+1] <= s:
        i = i+1
    return i
```

Solution de l'exercice 21 - Pour $n = 2p$, `p = n//2` et `p - 1 = (n-1)//2`,
pour $n = 2p + 1$, `p = n//2` et `p = (n-1)//2`.
On trouve bien les valeurs souhaitées.

Solution de l'exercice 22 -

```
def rang(x, L):
    n = len(L)
    compteur = 0
    for i in range(n):
        if liste[i] < x:
            compteur = compteur + 1
    return compteur
```

Solution de l'exercice 23 - Deux possibilités :

```
def mediane(L):
    n = len(L)
    i = 0
    while rang(L[i], L) != (n-1)//2:
        i = i + 1
    return L[i]
```

Solution de l'exercice 23 -

```
def mediane(L):
    n = len(L)
    for i in range(n):
        if rang(L[i], L) == (n-1)//2:
            return L[i]
```

Solution de l'exercice 24 -

```
def avantApres(x, L):
    n = len(L)
    avant = 0
    apres = 0
    for i in range(n):
        if liste[i] < x:
            avant = avant + 1
        if liste[i] > x:
            apres = apres + 1
    return avant, apres
```

Solution de l'exercice 25 -

```
def test_mediane(x, L):
    n = len(L)
    n1 = (n-1)//2
    n2 = n//2
    p, q = avantApres(x, L)
    return p <= n1 and q <= n2
```

Solution de l'exercice 26 -

```
def mediane(L):
    n = len(L)
    i = 0
    while not test_mediane(L[i], L):
        i = i + 1
    return L[i]
```

Solution de l'exercice 27 -

```
def petitTri(L, c):
    n = len(L)
    petit = []
    egal  = []
    grand = []
    for i in range(n):
        x = L[i]
        if x < c:
            petit.append(x)
        elif x == c:
            egal.append(x)
        else:
            grand.append(x)
    return petit, egal, grand
```

Solution de l'exercice 28 -

```
def choisir(k, liste):
    L = deepcopy(liste)
    while True:
        petit,egalgrand = separer(L, L[0])
        p = len(petit)
        q = len(grand)
        if k < p:
            L = deepcopy(petit)
        elif k < p + q:
            return L[0]
        else:
            L = deepcopy(grand)
            k = k - p - q
```

Solution de l'exercice 29 - On choisit de séparer en 3 parties de taille

- m, m et m pour $n = 3m$,
- m, m et $m+1$ pour $n = 3m+1$,
- $m+1$, $m+1$ et m pour $n = 3m+2$.

Donc les deux premières parties on `(n+1)//3` éléments

```
def separer3(U):
    n = len(M)
    m = (n+1)//3
    return U[ : m], U[m : 2*m], U[2*m : ]
```

Solution de l'exercice 30 -

```
def robuste(X, Y):
    Xg, Xc, Xd = separer3(X)
    Yg, Yc, Yd = separer3(Y)
    mXg = moyenne(Xg)
    mXc = moyenne(Xc)
    mXd = moyenne(Xd)
    mYg = moyenne(Yg)
    mYc = moyenne(Yc)
    mYd = moyenne(Yd)
    a = (mYd - mYg)/(mXd - mXg)
    b1 = mYd - a*mYd
    b2 = mYc - a*mYc
    return a, (b1 + b2)/2
```

DS II

Bases de données

1 Présentation

Les questions de ce devoir porteront sur la base de données étudiée en T.P.
On utilisera les tables Pays, Villes, Montagne et MontagnePays.
La table Pays contient les attributs

- **nom** est le nom du pays,
- **capitale** est le nom de la capitale du pays,
- **superficie** est la surface du pays,
- **population** est le nombre d'habitants du pays,
- **code** est le nom abrégé du pays qui sera utilisé dans les autres tables.

La table Ville contient les attributs

- **nom** est le nom de la ville,
- **population** est le nombre d'habitants de la ville,
- **pays** est le **code** du pays dans lequel est située la ville.

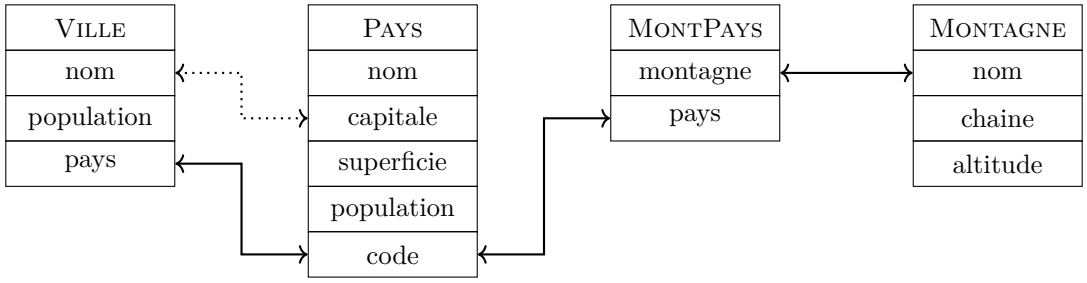
La table Montagne contient les attributs

- **nom** est le nom de la montagne,
- **chaine** est la chaîne de montagnes contenant la montagne (Alpes, Himalaya, ...),
- **altitude** est l'altitude du sommet.

La table MontPays relie les montagnes et les pays

- **montagne** est le nom de la montagne,
- **pays** est le **code** du pays dans lequel est située la montagne.

Une montagne peut être dans plusieurs pays et, bien sur un pays peut contenir plusieurs montagnes.
Les liaisons naturelles, qui peuvent servir dans les jointures, sont indiquées dans le diagramme ci-dessous.



2 Requêtes

2.1 Requêtes simples

Exercice 1
Écrire une requête qui renvoie le nom et l'altitude des montagnes de plus de 800 mètres d'altitude

Exercice 2
Écrire une requête qui renvoie le nom des villes de plus de 5 millions (5e6) d'habitants ainsi que leur population.

Exercice 3
Quels sont les pays dont la densité (population/superficie) est supérieure à 500 ?

2.2 Agrégations

Exercice 4
Quelle est la moyenne (`AVG`) de la population des "gros" pays, ceux ayant plus de 10 millions d'habitants ?

Exercice 5
Écrire une requête qui renvoie le nom des chaînes de montagne ainsi que le nombre de sommets leur appartenant.

Exercice 6
Quels sont les code des pays contenant au moins 5 villes de plus de 1 million d'habitants ?

2.3 Jointures

Exercice 7
Quels sont les pays dont la capitale a plus de 5 millions d'habitants ?

Exercice 8
Quelles sont les montagnes en France ? On donnera aussi leur altitude.

Exercice 9
Quels sont les pays qui contiennent des montagnes de plus de 8000 mètres ?

Exercice 10
Quels sont les pays qui contiennent une montagne de la chaîne des Alpes (`"Alps"`) ?

Exercice 11
Quels sont les pays qui contiennent au moins 5 montagnes de plus de 6000 mètres ?

2.4 Plus

Exercice 12
Quels sont les pays dont la population est supérieure à 10 fois la moyenne de la population des gros pays (question 4) ?

Exercice 13
Quels sont les pays pour lesquels la capitale n'est pas la ville la plus peuplée ?

3 Solutions

Solution de l'exercice 1 -

```
SELECT nom, altitude
FROM Montagne
WHERE altitude > 8000
```

Solution de l'exercice 2 -

```
SELECT nom, population
FROM Ville
WHERE population > 5e9
```

Solution de l'exercice 3 -

```
SELECT nom, population
FROM Ville
WHERE population > 5e9
```

Solution de l'exercice 4 -

```
SELECT AVG(population)
FROM Pays
WHERE population > 1e7
```

Solution de l'exercice 5 -

```
SELECT chaine, count()
FROM Montagne
GROUP BY chaines
```

Solution de l'exercice 6 -

```
SELECT pays, count() AS nb_villes
FROM Ville
Where population > 1e6
GROUP BY pays
HAVING nb_villes >= 5
```

Solution de l'exercice 7 -

```
SELECT p.nom, v.Population
FROM pays AS p JOIN ville AS v on p.capitale = v.Nom
where v.population > 5e6
```

Solution de l'exercice 8 -

```
SELECT m.nom, m.Altitude
FROM Pays AS p JOIN MontaPays AS mp ON p.code = mp.pays
               JOIN Montagnes AS m ON m.nom = mp.montagne
WHERE p.nom = "France"
```

Solution de l'exercice 9 -

```
SELECT p.nom, m.nom, m.Altitude
FROM Pays AS p JOIN MontPays AS mp ON p.code = mp.pays
               JOIN Montagne AS m ON m.nom = mp.montagne
WHERE m.altitude > 8000
```

Solution de l'exercice 10 -

```
SELECT DISTINCT p.nom
FROM Pays AS p JOIN MontPays AS mp ON p.code = mp.pays
               JOIN Montagne AS m ON m.nom = mp.montagne
WHERE m.chaine = "Alps"
```

Solution de l'exercice 11 -

```
SELECT p.nom, count() AS nombre
FROM Pays AS p JOIN MontPays AS mp ON p.code = mp.pays
               JOIN Montagne AS m ON m.nom = mp.montagne
WHERE m.altitude > 6000
GROUP BY p.nom
HAVING nombre >= 5
```

Solution de l'exercice 12 -

```
SELECT nom
FROM pays
WHERE population >= 10*(SELECT AVG(population)
                        FROM Pays
                        WHERE population > 1e7)
```

Solution de l'exercice 13 - On commence par créer la table des pays avec la population maximale d'une ville

```
SELECT p.nom AS pays, MAX(v.population) AS popmax
FROM Pays AS p JOIN Ville AS v ON v.pays = p.code
GROUP BY p.nom
```

Cette table va être jointe à `Pays` elle même jointe à `Villes`

```
SELECT p.nom, p.capitale, v.population, m.popmax
FROM Pays AS p
     JOIN Ville AS v ON v.nom = p.capitale AND v.pays = p.code
     JOIN (SELECT p.nom AS pays, MAX(v.population) AS popmax
           FROM Pays AS p JOIN Ville AS v ON v.pays = p.code
           GROUP BY p.nom) AS m ON m.pays = p.nom
WHERE v.population <> m.popmax
```

On doit ajouter la condition `AND v.pays = p.code` à la ligne 2 car il y a plusieurs villes qui ont le même nom.

Si on veut en plus, le nom de la ville la plus peuplée, c'est plus difficile.

```
1  SELECT p.nom, p.capitale, v.population AS population_c,
2         vm.ville_m, vm.population_m
3  FROM (SELECT m.pays AS pays, v.nom AS ville_m,
4               m.popmax AS population_m
5          FROM Ville AS v
6               JOIN (SELECT p.nom AS pays, p.code AS code,
7                     MAX(v.population) AS popmax
8                     FROM Pays AS p
9                          JOIN Ville AS v ON v.pays = p.code
10                    GROUP BY p.nom) AS m ON m.code = v.Pays
11         WHERE v.population = m.popmax) AS vm
12       JOIN Pays AS p ON vm.pays = p.nom
13       JOIN Ville AS v ON v.nom = p.capitale and v.pays = p.code
14 WHERE p.capitale <> vm.ville_m AND population_c IS NOT NULL
```

On doit ajouter la condition `AND population_c is not null` à la ligne 14 car il y a des capitales dont la population n'est pas connue