INFORMATIQUE BANQUE PT 2018 : CORRIGE

Analyse d'un 100m à l'aide d'un cinémomètre à effet Doppler-Fizeau

3. Traitement des données recueillies

3.1. Analyse du déroulement de la course

3.1.1. Détermination de l'instant d'arrivée

Q. 12. a) On a, par définition de la vitesse : $v(t) = \frac{dx}{dt}$ => dx = v(t) dt => $\int_{x_i}^{x_{i+1}} dx = \int_{t_i}^{t_{i+1}} v(t)\,dt$

=> $\boxed{x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} v(t)\,dt}$

Q. 12. b) Par la méthode des trapèzes : $\int_{t_i}^{t_{i+1}} v(t)\,dt = \left(\frac{v_{i+1} + v_i}{2}\right).(t_{i+1} - t_i)$

=> $\boxed{x_{i+1} = x_i + \left(\frac{v_{i+1} + v_i}{2}\right).(t_{i+1} - t_i)}$

Q. 12. c)

```
def inte(LV, LT):
    LX = [0]     # LX sera la liste des positions estimées
    for i in range(len(LV) - 1):
        LX.append(LX[i] + (LV[i+1] + LV[i]) * (LT[i+1] - LT[i]) / 2)
    return LX
```

Q. 13)

```
LXexp = inte(LVexp, LT)
import matplotlib.pyplot as p
p.figure()        # crée une fenêtre de trace vide
p.plot(LT, LXexp, '.')
A = [100, 100]
T = [0, 10]
p.plot(T, A, ':')          # ':' dotted line style (cf annexe)
p.xlim(0, 10)              # fixe les bornes de l'axe x
p.ylim(0, 120)             # fixe les bornes de l'axe y
p.xlabel('Temps (s)')
p.ylabel('Position (m)')
p.legend(['Position estimee', 'Arrivee'], loc='lower right')
p.show()        # affichage de la fenêtre
```

Q. 14. a) La pente vaut $\frac{x_{iA} - x_{iA-1}}{t_{iA} - t_{iA-1}} = \frac{d - x_{iA-1}}{T - t_{iA-1}}$ => $\boxed{T = t_{iA-1} + \frac{t_{iA} - t_{iA-1}}{x_{iA} - x_{iA-1}} . (d - x_{iA-1})}$

Q. 14. b)

```
def arrivee(LX, LT, d):
    i = 0
    while i < len(LX) and LX[i] < d:
        i += 1
    if i == len(LX):
        return False
    else:
        T = LT[i-1] + (LT[i] - LT[i-1]) / (LX[i] - LX[i-1]) * (d - LX[i-1])
        return T
```

Q. 14. c)

```
LTexp = LT              # il semble qu'il y ait une confusion au niveau des notations
print(arrivee(LXexp, LTexp, 100))
```

3.1.2. Délimitation des phases de la course

Q. 15) On a, par définition de l'accélération : $a = \frac{dv}{dt}$ => $a = \frac{v_{k+1} - v_k}{t_{k+1} - t_k}$

Donc la grandeur physique estimée est l'accélération.
Approximation utilisée : on procède par interpolation linéaire : on suppose que la vitesse varie linéairement entre deux instants de mesure.

Q. 16. a) len(f(LVexp, LTexp)) = n – 1 car LY est initialement vide et on ajoute (len(LT) – 1) termes.

Q. 16. b)

```
LTmodif = []
for k in range (len(LTexp) - 1):
    LTmodif.append(LTexp[k])
LY = f(LVexp, LTexp)
p.figure()      # crée une fenêtre de trace vide
p.plot(LTmodif, LY, '.')
p.show()        # affichage de la fenêtre
```

Q. 17)

```
def instants(LY, LT):
    moyenne = sum(LY) / len(LY)     # valeur moyenne de LY
    sup = 1.5 * moyenne
    inf = 0.5 * moyenne

# recherche de t1, début de la phase à vitesse constante
    i = 0
    while (i < len(LY)) and not(inf < LY[i] < sup):
        i += 1
    if i == len(LY):
        t1 = -1
    else:
        t1 = LT[i]

# recherche de t2, début de la phase de décélération
    if LY[-1] > inf:
        t2 = -1
    else:
```

```
    i = 0
    while LY[len(LY) - i - 1] < inf:
        i += 1
    t2 = LT[len(LY) - i]

  return t1, t2
```

Q. 18) Complexité :
calcul de moyenne : O(n)
calcul de t1 : 1ère boucle while : O(n)
calcul de t2 : 2ème boucle while : O(n)
Ces calculs n'étant pas « imbriqués », la complexité est en O(n).

3.2. Modélisation dynamique de la course

3.2.1. Identification et validation du modèle

Q. 19)

```
import numpy as np
P = np.polyfit(Lvexp, Laexp, 2)
A = P[2]
B = P[1]
C = P[0]
```

Q. 20. a) On a, par définition de l'accélération : $a = \frac{dv}{dt}$ et $a = A + B\,v(t) + C\,v(t)^2$

$$\Rightarrow \frac{v_{i+1} - v_i}{t_{i+1} - t_i} = A + B\,v_i + C\,v_i^2$$

$$\Rightarrow \boxed{v_{i+1} = v_i + \left(A + B\,v_i + C\,v_i^2\right).\left(t_{i+1} - t_i\right)}$$

Q. 20. b)

```
def simu(LT, P):
    LVsimu = [0]        # initialization de la liste
    for i in range(len(LT) - 1):
        LVsimu.append(LVsimu[i] + (P[2] + P[1] * LVsimu[i] + P[0] * LVsimu[i]**2) * (LT[i+1] - LT[i]))
    return LVsimu
```

Q. 21) L'expression de la quantité affichée à l'écran est (avec n le nombre de mesures) :

$$\sqrt{\frac{\sum_{i=0}^{n-1}\left(v_{i\,sim} - v_{i\,exp}\right)^2}{\sum_{i=0}^{n-1}\left(v_{i\,exp}\right)^2}}$$

Cette quantité mesure, quantitativement, l'écart entre les mesures et la simulation. En effet, cette grandeur adimensionnée permet d'évaluer l'écart entre la série des $v_{i\,sim}$ (simulation) et la série des $v_{i\,exp}$ (mesures) :

- Si les deux séries sont identiques : $(v_{i\,sim} - v_{i\,exp}) = 0 \quad \forall$ i. La quantité affichée vaut 0.
- Si les deux séries sont très différentes : les $(v_{i\,sim} - v_{i\,exp})$ sont grands. La quantité affichée est grande.

3.2.2. Exploitation du modèle pour estimer les efforts sur le coureur

Q. 22) Modification de l'algorithme :

```
def composantes(LV, LA, P):
    a, b = P[0], P[1]
    LA0, LA1, LA2 = [], [], []
    for k in range(len(LA)):
        LA2.append(a * LV[k]**2)
        LA1.append(b * LV[k])
        LA0.append(LA[k] - LA1[k] - LA2[k])
    return (LA0, LA1, LA2)
```

Il y a deux erreurs dans l'algorithme proposé :

- Attention, on augmente la taille de la liste à chaque itération (on ajoute terme après terme). Exemple : for k = 2 => LA2[k] n'est pas défini car len(LA2) = 2
- LA1 + b * LV[k] ???? LA1 est une liste et b * LV[k] est un flottant !!

Ordre dans lequel sont renvoyées les trois composantes recherchées :

- En 1er : LA0 : liste des f_i => propulsion
- En 2ème : LA1 : liste des $P_1\ v_i = b\ v_i = B\ v_i$ => traînée de frottement
- En 3ème : LA2 : liste des $P_0\ v_i^2 = a\ v_i^2 = C\ v_i^2$ => traînée de pression (ou de forme)

3.2.3. Bilan énergétique de la course

Q. 23) Avant propos : par la méthode des trapèzes : $\int_{t_i}^{t_{i+1}} a(t)\,v(t)\,dt = \left(\frac{a_{i+1}\,v_{i+1} + a_i\,v_i}{2}\right).(t_{i+1} - t_i)$

```
def travail(LA, LV, LT, d):
    LX = inte(LV, LT)
    T = arrivee(LX, LT, d)
    k = 0
    W = 0
    while LT[k] < T:
        W += (LA[k+1] * LV[k+1] + LA[k] * LV[k]) * (LT[k+1] - LT[k]) / 2
        k += 1
    return W
```

3.3. Stockage et mise en forme des données

3.3.1. Mise en œuvre de la base de données

Q. 24) Chaque coureur n'a pu courir qu'une seule fois une certain épreuve ! Donc le couple d'identifiants id_coureur et id_epreuve constitue bien une clé primaire pour cette table.

Q. 25)

```
SELECT nom, date
FROM epreuves
WHERE distance = 100
```

Q. 26) Cette requête donne, pour tous les “100m” enregistrés dans la base parcourus en moins de 12s, les quadruplets :
- le temps, ou instant d’arrivée, en s
- l’instant initial de la phase à vitesse constante, en s
- l’instant initial de la phase de décélération, en s
- le travail massique de la force de propulsion, en J/kg.

Q. 27)
```
SELECT c.nom, c.prenom, e.nom, e.date, p.temps
FROM coureurs c, epreuves e, performances p
WHERE c.id = p.id_coureur AND e.id = p.id_epreuve AND e.distance = 100
```

Autre possibilité :

```
SELECT c.nom, c.prenom, e.nom, e.date, p.temps
FROM performances p
JOIN epreuves e ON e.id = p.id_epreuve
JOIN coureurs c ON c.id = p.id_coureur
WHERE e.distance = 100
```

3.3.2. Traitement des données récupérées

3.3.2.1. Evolution des performances des coureurs au fil du temps

Q. 28)
```
def performances(nom, prenom, T):
    L = []          # cette liste ne contiendra que les listes du coureur qui nous intéresse
    for k in range(len(T)):
        if T[k][0].lower() = nom.lower() and T[k][1].lower() = prenom.lower():
            L.append(T[k])
    date1 = [2000, 1, 1]
    jours = []
    temps = []
    for i in range(len(L)):
        date2 = L[i][3].split('-')
        for k in range(2):
            date2[k] = int(date2[k])
        jours.append(nb_jours(date1, date2))
        temps.append(L[i][4])
    return (jours, temps)
```

3.3.2.2. Extraction des dix meilleurs temps

Q. 29)
```
def top10(T):
    for k in range(10):
        temps_mini = T[k][4]
        indice_mini = k
        ligne_mini = T[k]
        for i in range(k : len(T)):
            if T[i][4] < temps_mini:
                temps_mini = T[i][4]
```

```
            indice_mini = i
            ligne_mini = T[i]
        T[indice_mini] = T[k]
        T[k] = ligne_mini
    Tbis = []
    for k in range(10):
        Tbis.append(T[k])
    return Tbis
```

Q. 30. a) On a une complexité en O(10 N) = $\boxed{O(N)}$, car on parcourt 10 fois la liste quasiment entièrement à chaque fois.

Q. 30. b) On rappelle des résultats de cours de 2ème année (cours sur les tris) :

Cas	Tri par insertion	Tri rapide (quicksort)	Tri fusion
Meilleur	O(N)	O(N logN)	O(N logN)
Moyen	$O(N^2)$	O(N logN)	O(N logN)
Pire	$O(N^2)$	$O(N^2)$	O(N logN)

Les algorithmes performants (tri rapide, tri fusion) ont une complexité en O(N logN) dans les cas moyens. Avec le tri par sélection, en ne cherchant que les 10 meilleurs temps (on n'est alors pas obligé de trier le tableau en entier !), la complexité est meilleure (en O(N)).

Q. 30. c) Si on triait entièrement T par le tri par sélection, alors la complexité serait en $\boxed{O(N^2)}$.

En effet, il y a deux boucles for imbriquées, l'une de longueur N, l'autre allant de k à N :

$$\sum_{j=1}^{N} j = \frac{N(N-1)}{2} = O(N^2)$$

La complexité serait alors moins bonne que pour un tri rapide ou un tri fusion (complexité en O(N logN)).