Devoir surveillé n°2

Durée : 3 heures, calculatrices et documents interdits

Présentation

On s'intéresse dans ce sujet aux montres multisport, en développement croissant depuis les dix dernières années. Ces montres permettent aux sportifs amateurs ou professionnels d'enregistrer diverses données physiques et physiologiques. Elles permettent maintenant de connaître sa position GPS, d'enregistrer un trajet et tous les paramètres associés, de suivre un trajet prédéfini. Les plus sophistiquées intègrent également un altimètre, un baromètre, un thermomètre, une boussole, un lecteur MP3, etc.
On se limite dans ce sujet à :

- l'acquisition et le traitement des données GPS sur un parcours : **partie I** ;
- l'acquisition du rythme cardiaque : **partie II** ;
- le partage des activités : **partie III**.

Dans tout le sujet, on supposera que les bibliothèques `numpy` et `matplotlib.pyplot` sont importées par `from numpy import *` et `from matplotlib.pyplot import *`

Partie I - Acquisition et traitement des données GPS

I.1 - Introduction

Dans la montre, un module électronique capte les signaux GPS à l'aide d'une antenne et décode ces signaux pour générer une chaîne de caractères constituée de plusieurs lignes. On ne s'intéresse qu'au traitement de cette chaîne de caractères qui est transmise par le module GPS. Les données envoyées par le récepteur GPS suivent la norme NMEA083 définie dans l'**annexe 1**. On suppose que la variable `donneesGPS` est une chaîne de caractères reçue par le récepteur. Cette chaîne contient plusieurs lignes dont le séparateur est le caractère spécial `'\n'`. Pour la lisibilité, cette chaîne est affichée avec les sauts de ligne.

```
"$GPRMC,092828.00,A,4754.68988,N,00154.69147,E,0.444070619,A*7C\n
$GPVTGTM,0.444,N,0.822,K,A*2F\n
$GPGGA,092828.00,4754.68988,N,00154.69147,E,1,04,2.61,84.3,M,46.5,M*64\n
$GPGSA,A,3,25,29,24,32,3.99,2.61,3.02*0E\n
$GPGSV,3,1,11,02,32,07504,22,06,15,03712,47,072,16*4A\n
$GPGSV,3,2,11,14,38,276,11,24,18,141,19,25,80,353,24,29,56,196,30*7F\n
$GPGSV,3,3,11,31,34,306,25,32,35,250,36,33,32,202,27*4F\n
$GPGLL,4754.68986,N,00154.69154,E,092828.00,A,A*6B\n"
```

I.2 - Récupération des données brutes

L'objectif est de récupérer l'heure, la latitude, la longitude et l'altitude dans les données de la trame GPGGA (voir **annexe 1**). Cette procédure va se décomposer en 3 étapes :

- récupérer la trame GPGGA dans la série de trames reçues ;
- vérifier la validité de la trame ;
- extraire les données.

Pour répondre aux questions suivantes, il est conseillé d'utiliser les propriétés et fonctions sur les caractères et chaînes de caractères données en **annexe 2**.
Pour récupérer la trame GPGGA, on a écrit la fonction `recupererGPGGA(chaine)` (donnée sur le **Document Réponse**) qui prend en argument la chaîne de caractères `chaine` et qui renvoie la ligne correspondant à la trame GPGGA si elle existe sous forme de liste découpée suivant les virgules. Sinon elle renvoie une liste vide.
La fonction `indice_GPGGA(listeChaine)` renvoie l'indice de l'élément contenant '$GPGGA' de `listeChaine`.

Q1. Expliquer les lignes 2 à 8 de cette fonction.

Q2. Écrire une fonction `indice_GPGGA(listeChaine)` prenant en argument une liste de chaînes de caractères et qui renvoie l'indice de la position de la chaîne contenant '`$GPGGA`' quand elle existe ou la longueur de `listeChaine` sinon. Vous utiliserez une boucle `while`.

Q3. Donner l'instruction à écrire pour récupérer la trame GPGGA de la variable `donneesGPS` et la stocker dans la variable `listeGPGGABrute`.

I.3 - Test de validité de la trame GPGGA

Pour que la trame soit valide, il faut vérifier que :

- les valeurs soient présentes ;
  par exemple, la trame "$GPGGA,092811.00,0,00,99.99*65\n" n'est pas valide ;
- le coefficient de précision soit inférieur à 5 ;
- la somme de contrôle soit valide. La somme de contrôle est obtenue par une opération "ou exclusif" entre une représentation binaire de chaque caractère entre `$`' et `*`' (ces deux caractères étant exclus). Sa valeur dans la trame est donnée en base hexadécimale.

On suppose dans cette sous-partie que la trame, constituée de 14 éléments, a été récupérée et stockée dans `listeGPGGABrute` soit ici :
`listeGPGGABrute= ['GPGGA', '092828.02', '4754.68988', 'N', '00154.69147', 'E', '1', '04', '2.61', '84.3', 'M', '46.5', 'M', '*64']`
Pour les questions suivantes, trame correspond à une variable ayant la même forme que `listeGPGGABrute`.

Q4. Écrire une fonction `testPresence(trame)` qui prend en argument la liste de chaînes de caractères `trame` correspondant à une trame GPGGA et qui renvoie `True` si les données sont présentes et non vides, `False` sinon.

Q5. Écrire une fonction `testPrecision(trame)` qui prend en argument la liste de chaînes de caractères `trame` correspondant à une trame GPGGA et qui renvoie `True` si la précision est inférieure à 5, `False` sinon.

On donne dans le **Document Réponse** une fonction `testSC(trame)` qui prend la liste de chaînes de caractères `trame` correspondant à une trame GPGGA et qui renvoie `True` si la trame est valide, `False` sinon.

Q6. Donner la valeur renvoyée par la fonction `testSC` appliquée à la liste `['GPG','*A0']`.
Donner notamment la valeur de la variable `csHex`. On donne les résultats suivants :
`bin(ord('G'))` renvoie `'0b01000111'`, `bin(ord('P'))` renvoie `'0b01010000'`,
`bin(ord('*'))` renvoie `'0b00101010'`, `bin(ord('A'))` renvoie `'0b01000001'`,
`bin(ord('0'))` renvoie `'0b00110000'`.

I.4 - Récupération des données

La trame étant valide, on recherche maintenant à récupérer les données qui nous intéressent en les stockant dans le format adéquat :

- l'horaire exprimé en secondes ;

- la latitude exprimée en degrés (0 correspond à l'équateur, elle varie de -90^o à 90^o). Au Nord l'angle sera compté positivement et au Sud négativement ;
- la longitude exprimée en degrés (0 correspond au méridien de Greenwich, elle varie de -180^o à 180^o). L'Ouest sera compté positivement et l'Est négativement ;
- l'altitude exprimée en mètres.

Pour la latitude, comme la longitude, les minutes d'angle sont toujours représentées avec 8 caractères (dont la virgule) alors que les degrés sont représentés avec 2 ou 3 caractères.

Q7. Écrire une fonction `convHoraire(chHoraire)` qui prend en argument la chaîne de caractères `chHoraire` représentant l'horaire avec le format de la norme GPGGA (`092828.00` correspondant à 09 h 28 min 28,00 secondes) et qui renvoie la valeur de l'horaire exprimé en secondes.

Q8. Écrire une fonction `convAngle(chAngle,chCard)` qui prend en argument la chaîne de caractères `chAngle` correspondant à la latitude ou la longitude, ainsi que la chaîne de caractères `chCard` correspondant à la direction (N/S ou E/O) et qui renvoie le flottant signé correspondant à la valeur de l'angle en degré (il faudra convertir les minutes d'angle en valeur décimale).

Q9. Compléter l'ébauche de la fonction `recupDonnees(trame)` qui prend en argument `trame` correspondant à une trame GPGGA et qui renvoie une liste de 4 réels contenant (Heure, Latitude, Longitude, Altitude).

I.5 - Sauvegarde d'une activité

Lors d'une activité, on récupère toutes les secondes :

- les données du GPS : horaire (en seconde), latitude (en degré), longitude (en degré) et altitude (en mètre) ;
- la fréquence cardiaque.

Toutes les secondes, ces 5 données sont enregistrées dans un fichier texte, en les séparant par une virgule. Chaque ligne du fichier correspond à un "point" de mesure.

Exemple d'une ligne du fichier texte : `"34108,47.911498,001.911526,0084.3,105\n"`.

L'horaire, en secondes, est un entier représenté avec 5 caractères. On garde 6 chiffres après la virgule pour les angles en degrés. La partie entière de l'altitude est représentée avec 4 caractères et on prend une précision au décimètre. La fréquence cardiaque est un entier représenté avec 3 caractères. Tous les caractères sont stockés sur 1 octet (ascii).

Q10. On suppose que l'on réalise une activité d'une heure. Donner l'ordre de grandeur de la taille mémoire en octets du fichier texte généré. Dans le cahier des charges, on impose que la montre peut sauvegarder 200 h d'activités. Donner l'ordre de grandeur de la taille mémoire nécessaire au stockage des données pour répondre au cahier des charges.

La mémoire des montres est de l'ordre de 20 Mo.

Q11. Proposer une solution permettant de ne pas dépasser la mémoire disponible.

Partie II - Acquisition du rythme cardiaque

Lors d'une activité sportive, les montres multisport permettent l'enregistrement et la consultation du rythme cardiaque. Cela permet au sportif de contrôler son effort et d'optimiser ses performances. Cette donnée est intéressante durant les compétitions et les entraînements.

La première méthode utilisée est d'enregistrer l'ElectroCardioGramme (ECG). On mesure l'activité électrique du cur à l'aide d'électrodes. Cette méthode nécessite d'installer une ceinture cardiothoracique qui mesure et envoie les données à la montre par Bluetooth ou ANT+ en fonction des marques.

La seconde méthode est la PhotoPlestysmoGraphie (PPG) ou Oxymétrie de pouls. L'hémoglobine, lorsqu'elle est chargée en oxygène, absorbe davantage les infrarouges que lorsqu'elle n'est pas chargée en

oxygène. On peut donc avec un émetteur et un récepteur infrarouge mesurer en temps réel la saturation en oxygène du sang et en déduire le rythme cardiaque. En centre hospitalier, le capteur est positionné au bout d'un doigt. L'épaisseur de peau étant faible et la zone très vascularisée, on utilise des longueurs d'onde "optimales" de l'ordre de 800 nm (lumière rouge). Dans les montres, la prise de mesure s'effectue au poignet. La peau y étant plus épaisse, on utilise des longueurs d'onde de l'ordre de 570 nm (lumière verte) qui pénètrent mieux les tissus.

Le capteur PPG renvoie une valeur de longueur d'onde avec un temps d'échantillonnage $T_e = 0,02$ s. Tous les T_e, on enregistre la valeur de ce signal (sans unité) dans une liste notée `signal` et le temps (en s) dans une liste notée `temps`. Chaque élément de la liste `signal` est un nombre entier, image de la quantité d'infrarouge absorbée.

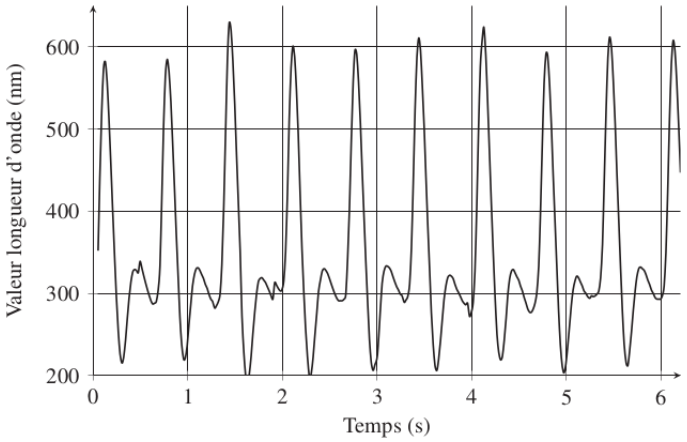


FIGURE 1 – Mesure PPG : tracé de la liste `signal` en fonction de la liste `temps`

Un essai d'une durée d'environ 6 secondes permet de tracer la courbe de la **figure 1**.

On observe que la valeur du signal infrarouge se décompose en deux valeurs :

- une valeur quasi constante, de l'ordre de 300 sur la **figure 1**, qui correspond à l'absorption due aux tissus, au flux veineux et à une partie constante du flux artériel ;
- une partie variable due exclusivement à la variation du flux artériel.

II.1 - Application d'un filtre passe-bas

Pour atténuer certaines perturbations, on applique un filtre passe-bas. Il est caractérisé par l'équation différentielle :

$$\frac{ds(t)}{dt} + 2\pi f_c\, s(t) = 2\pi f_c\, e(t) \tag{1}$$

où $e(t)$ est le signal d'entrée du filtre, $s(t)$ le signal en sortie du filtre et f_c la fréquence de coupure du filtre.

On peut intégrer l'équation (1) sur un intervalle $[t, t + T_e]$ et poser :

$$s(t + dt) = s(t) + 2\pi f_c \int_t^{t+T_e} (e(u) - s(u))\, du \tag{2}$$

On obtient un signal discrétisé avec une fréquence d'échantillonnage $f_e = 1/T_e$.

On note $e_k = e(t_k)$ et $s_k = s(t_k)$ les valeurs respectives des signaux d'entrée e et de sortie s à l'instant t_k.

Q12. Montrer qu'on obtient, après discrétisation et en appliquant la méthode des trapèzes pour calculer une valeur approchée de l'intégrale dans l'équation (2), la relation de récurrence suivante :

$$s_{k+1} = \frac{(1-G)s_k + G(e_{k+1} + e_k)}{1+G} \tag{3}$$

où G est une constante dont vous donnerez l'expression en fonction des paramètres f_c et f_e.

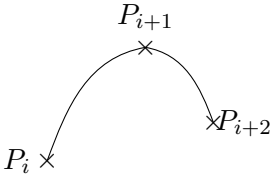
Q13. Écrire la fonction `filtrage(e,G)` d'argument `e` tableau du signal d'entrée, `G` la constante définie précédemment et qui renvoie le tableau de valeurs du signal filtré.

II.2 - Méthode 1 - Récupération des pics

Le signal ayant été filtré, une première méthode consiste à récupérer les instants des pics, puis d'en déduire la fréquence cardiaque à partir du temps entre deux pics consécutifs.

Pour repérer un pic, nous allons prendre trois points de mesures consécutifs P_i, P_{i+1} et P_{i+2}. On considère que l'on trouve un pic si :

- la valeur en P_{i+1} est supérieure à la valeur en P_i ;
- la valeur en P_{i+1} est supérieure à la valeur en P_{i+2} ;
- les valeurs des trois points sont supérieures à une valeur seuil minimale (par exemple 550 sur la **figure 1**) afin de ne trouver que des extrema " valides ".



Pour récupérer chaque pic, il est inutile de travailler sur l'ensemble du signal, mais il faut prendre suffisamment de points pour être certain d'avoir un pic. La fréquence cardiaque au repos usuelle étant de 60 pulsations par minute, soit un pic par seconde, on fait le choix de travailler, par sécurité, sur un intervalle de 3 s, ce qui correspond à des fenêtres de 150 points avec notre temps d'échantillonnage de $0,02$ s. Cette partie du signal sera notée `extSignal`.

Q14. Compléter le test du `while` de la fonction `detectionPics(extSignal,seuil)` qui prend en argument `extSignal`, liste des données du signal partiel avec 150 points de mesure, ainsi que la valeur minimale `seuil` et qui renvoie l'indice du premier "pic" dans la partie `extSignal` du signal traité.

Q15. Commenter précisément la fonction `pulsationCardiaque(signal,Te,seuil)` qui prend en argument `signal`, liste des données du signal complet, `Te` le temps d'échantillonnage en seconde et `seuil` la valeur du seuil de détection. Expliciter ce que représente la valeur renvoyée.

II.3 - Méthode 2 - Transformée de Fourier discrète

Une seconde méthode consiste à effectuer une étude fréquentielle du signal. Pour cela, on applique aux données une Transformation de Fourier Discrète (TFD).

Pour un signal s de N échantillons obtenus à une fréquence f_e, la TFD du signal, notée $S(k)$, est donnée par :

$$S(k) = \sum_{n=0}^{N-1} s_n e^{-2i\pi k \frac{n}{N}} \qquad \text{pour } 0 \leqslant k < N \tag{4}$$

Les termes $S(k)$ représentent une approximation de la transformée de Fourier du signal s, aux N fréquences $f_k = \dfrac{k}{N} f_e$. $S(k)$ est un nombre complexe dont seul le module nous intéresse.

Q16. Écrire une fonction `TFD(signal,Te)` qui prend en argument `signal`, liste des données du signal et `Te` le temps d'échantillonnage et qui renvoie la liste des fréquences et la liste des modules de $|S(k)|$. La fonction `abs(z)` renvoie la valeur du module du complexe `z` (fonctionne sur un tableau). En Python, le complexe $1 + 2i$ s'écrit `1+2j` et le complexe $a + ib$ s'écrit `a+b*1j`.

Pour le signal de la **figure 1**, nous obtenons le diagramme des fréquences de la **figure 2** réalisé avec un temps d'échantillonnage de $0,02$ s.

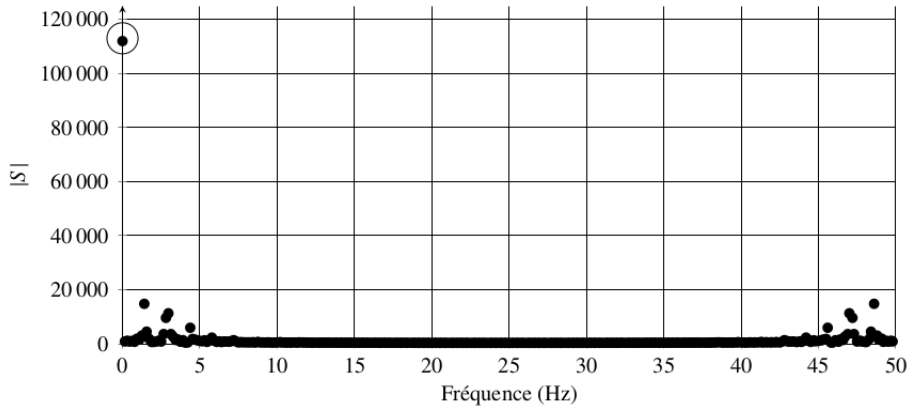


FIGURE 2 – Spectre après application de la TFD

Q17. Justifier la présence d'une valeur très élevée pour une fréquence nulle.

Le rythme cardiaque a une pulsation "normale" comprise entre 50 et 200 pulsations par minute. Nous allons prendre une marge en considérant l'amplitude de fréquences de pulsations dans l'intervalle $[30, 220]$ pulsations par minute.

Q18. Modifier la fonction `TFD` précédente pour qu'elle ne calcule que les valeurs de $S(k)$ comprises entre ces deux fréquences.

On obtient alors le spectre de la **figure 3**.

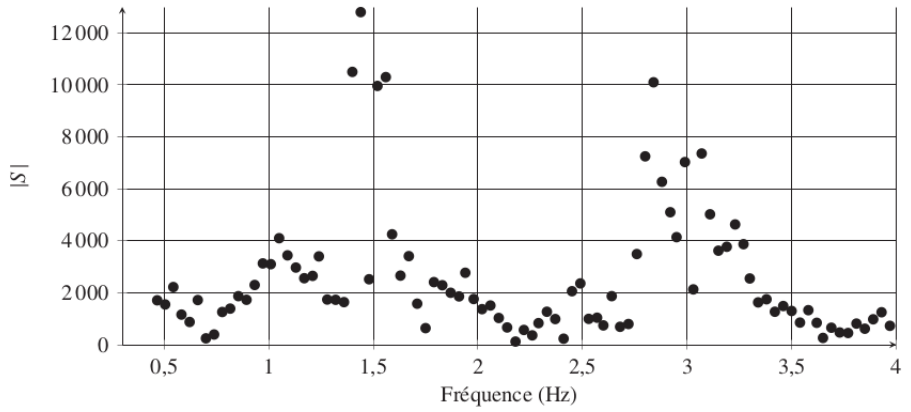


FIGURE 3 – Spectre issu de la TFD pour $f \in [0,2Hz, 4Hz]$

Pour déterminer la pulsation cardiaque toutes les secondes d'un signal, on calcule la fréquence cardiaque sur un intervalle de temps $\Delta t = T_f - T_d$, puis, on fait "glisser" cette fenêtre d'une seconde et on recalcule la fréquence cardiaque (**figure 4**).

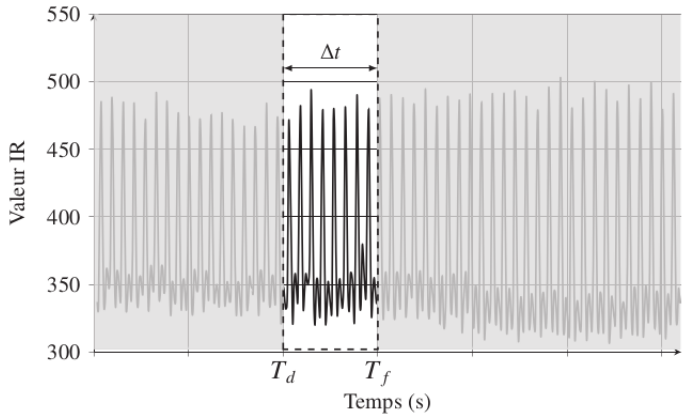


FIGURE 4 – Fenêtrage du signal

Pour appliquer ce principe, on propose le programme non commenté suivant où `Signal` contient la liste des valeurs de l'infrarouge durant le temps de l'activité.

```
# Acquisition des donn\'ees
Temps, Signal = acquisition( Donnees )

# fr\'equence d'\'echantillonnage
G = 0.628
fe = 50
# Filtrage du signal
SignalFiltre = filtrage( Signal, G )
#
indDepart = 0
indFin = 600
#
HR = []
nbPoints = len( SignalFiltre )
while indFin < nbPoints :
    partSignal = SignalFiltre[indDepart:indFin]
    freq, Sk = TFD( partSignal, fe )
    HR.append( 60 * freq[ SK.index( max(Sk) ) ] )   # ligne 69
    indDepart =                                     # ligne 70
    indFin =                                        # ligne 71
```

Q19. Donner l'intervalle Δt choisi par le programmeur. Expliquer la ligne 69 et compléter les lignes 70 et 71.

Dans la méthode précédente, on calcule la fréquence cardiaque par application de la TFD à toutes les valeurs contenues dans une fenêtre temporelle de largeur Δt sans traitement sur ces valeurs. On parle de fenêtre rectangulaire. Cependant, l'application de la TFD sur ces valeurs peut induire, par des effets de lobes secondaires, des erreurs sur la fréquence maximale et donc sur la pulsation cardiaque relevée.

Pour corriger ce problème, on peut appliquer un coefficient multiplicateur en réduisant l'influence des valeurs lorsque l'on s'approche des bords de la fenêtre. Un fenêtrage classique est la fenêtre dite **de Hann** dont on donne la fonction suivante :

```
def Hann(extSignal, fe) :
    t = arrange( len(extSignal) ) / fe
    hann = 1/2 - 1/2*cos( 2 * pi * t / t[-1] )
    return list( extSignal * hann )
```

Q20. Parmi les quatre propositions du **Document Réponse**, entourer la fonction correspondant à la définition de la fenêtre dite de Hann.

Q21. Expliquer comment modifier le programme donné à la **Q19** pour prendre en compte la fenêtre dite de Hann précédente.

Partie III - Partage des activités

Les fabricants de montres offrent la possibilité d'enregistrer les activités dans une base de données afin qu'elles soient accessibles sur PC ou tablettes et qu'elles puissent être partagées avec des amis.

La base de données est constituée des tables suivantes :

Table **activite**

- `Ida` : entier permettant d'identifier l'activité
- `Idm` : entier correspondant à l'identifiant du membre "propriétaire" de l'activité
- `Date` : correspondant à la date (type date) de l'activité
- `Type` : chaîne de caractères correspondant au type d'activité "course", "marche",
- `Distance` : entier correspondant à la distance parcourue en mètre de l'activité
- `Temps` : entier correspondant à la durée en secondes de l'activité
- `Fichier` : contient le lien vers le fichier de l'activité

Table **amis** qui établit une relation entre 2 membres

- `Idl` : entier, identifiant du lien
- `Membre1` : Idm d'un membre
- `Membre2` : Idm d'un second membre

Il ne peut pas y avoir de doublons dans la table amis : si A est le membre 1 et B le membre 2 d'une relation d'amitiés, la ligne membre 1=B et membre 2=A n'existe pas dans la table.

On considère pour les questions suivantes un membre dont l'identifiant est 1 (`Idm = 1`).

Q22. Écrire une requête `SQL` permettant de récupérer la liste des identifiants des activités du membre dont l'identifiant est 1.

Q23. Écrire une requête `SQL` permettant de donner la date, la distance parcourue et la vitesse moyenne en km/h des activités de type "course" du membre dont l'identifiant est 1.

On donne la requête suivante :

```
SELECT activite.Ida FROM activite
JOIN (SELECT membre1 AS idam1 FROM amis WHERE membre2 = 1
      UNION
      SELECT membre2 AS idam1 FROM amis WHERE membre1 = 1) AS amis1
ON activite.idm = amis1.idam1
WHERE Type = 'marche'
```

Q24. Décrire chaque instruction de la requête et expliquer ce qu'elle renvoie.

Annexe 1 - Norme NMEA 0183

La National Marine Electronics Association (NMEA) est une association américaine de fabricant de matériels électroniques maritimes. Elle a défini un protocole de communication entre équipements marins dont les GPS. Ainsi, un récepteur GPS envoie une série de phrases (trames) sur une communication série. Chaque trame est constituée de chaînes de caractères commençant par $' terminées par un retour chariot. Les caractères constituant la trame ont un code ASCII compris entre 20 et 127, mais sont codés sur un octet.
Un récepteur GPS de type NEO-6M transmet les trames suivantes toutes les secondes :

```
$GPGLL,4754.68986,N,00154.69154,E,092827.00,A,A*6B
$GPRMC,092828.00,A,4754.68988,N,00154.69147,E,0.444070619,A*7C
$GPVTGTM,0.444,N,0.822,K,A*2F
$GPGGA,092828.00,4754.68988,N,00154.69147,E,1,04,2.61,84.3,M,46.5,M*64
$GPGSA,A,3,25,29,24,32,3.99,2.61,3.02*0E
$GPGSV,3,1,11,02,32,07504,22,06,15,03712,47,072,16*4A
$GPGSV,3,2,11,14,38,276,11,24,18,141,19,25,80,353,24,29,56,196,30*7F
$GPGSV,3,3,11,31,34,306,25,32,35,250,36,33,32,202,27*4F
```

Pour une montre multisport, nous n'utiliserons que la trame GGA qui contient les informations suivantes :
$GPGGA,092828.00,4754.68988,N,00154.69147,E,1,04,2.61,84.3,M,46.5,M*64

- GP : Type de système (GP : GPS, GA : Galileo, GL : Glonass, ...)
- GGA : Type de trame
- 092828.00 : Horaire UTC - 09 h 28 min 28,00 s
- 4754.68988 : Latitude : 47°54,68988'
- N : latitude Nord
- 00154.69147 : Longitude 001°54,69147'
- E : Longitude Est
- 1 : Type de positionnement
- 04 : Nombre de satellites
- 2.61 : Coefficient de précision
- 84.3,M : Altitude par rapport au niveau moyen des océans
- 46.5,M : Correction de la hauteur de la géoïde par rapport à l'ellipsoïde WGS84
- : Champ toujours vide
- *64 : Somme de contrôle

Annexe 2 - Caractères et chaînes de caractères en Python

Fonctions

`ord(char1)`	renvoie la valeur entière correspondant à `char1` dans la table ASCII	`ord('A') -> 65`
`chr(int1)`	renvoie le caractère correspondant à `int1` dans la table ASCII	`chr(65) -> 'A'`

Caractères d'échappement

- '\n' : saut de ligne
- '\t' : tabulation

Exemple :

```
>>> chaine = 'Ceci est un test.\nOn peut mettre une tabulation entre a et b : a\tb.
>>> print(chaine)
Ceci est un test.
On peut mettre une tabulation entre a et b : a	b.
```

Fonctions sur les chaînes de caractères

Une chaîne de caractères est gérée comme une liste.
Exemple : `chaine = "Concours Commun INP \n"`.

`chaine[i]`	Renvoie le ième terme de la chaine	`chaine[5] -> u`
`chaine[i:j]`	Renvoie la chaîne comprise entre le ième et le jième-1 terme	`chaine[9:15] -> Commun`
`chaine.split()`	Découpe la chaîne au caractère désigné, par défaut espace'.	`chaine.split('C') -> ['','oncours ', 'ommun INP \n']`
`bin(int1)`	Convertit un entier en la chaîne de caractères de son expression binaire, précédée de 0b' pour indiquer la base binaire	`bin(12) -> '0b1100'`
`hex(int1)`	Convertit un entier en la chaîne de caractères de son expression hexadécimale, précédée de 0x' pour indiquer la base héxadécimale	`hex(12) -> '0xc'` `hex(0b01010000) -> '0x50'`

Annexe 3 - Ou Exclusif

La fonction Ou Exclusif renvoie le ou exclusif bit à bit de deux entiers.
Exemple : $10 \veebar 15$ renvoie 5
ou en représentation binaire : `bin(0b1010 ⊻ 0b1111)` renvoie '0b0101'.

Annexe 4 - Fonctions sur les tableaux et les listes

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

	Python
tableau à une dimension	`L = [1,2,3] (liste)` `v = array( [1,2,3] ) (vecteur)`
créer un vecteur rempli de 0 de taille n	`zeros(n)`
accéder à un élément	`v[0] renvoie 1 (L[0] également)`
ajouter un élément	`L.append(5) uniquement sur les listes`
tableau à deux dimensions (matrice)	`M = array( ( [1,2,3], [3,4,5] ) )`
accéder à un élément	`M[1,2] donne 5`
extraire une portion de tableau (2 premières colonnes)	`M[:,0:2]`
extraire la colonne i	`M[:,i]`
extraire la ligne i	`M[i,:]`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`

Transformer une ligne en colonne	`a = array( [1 ,2 ,3] )`
	`b = a.reschape(3 ,1)`
	`print (b)`
	`>>> array( [ [1],[2],[3] ] )`

produit matrice-vecteur	`a = array( [ [1,2,3], [4,5,6], [7,8,9] ] )`
	`b = array( [1 ,2 ,3] )`
	`print( a.dot( b ) )`
	`>>> array( [14, 32, 50] )`