Concours Blanc - Février 2016 : Correction

Le sujet consiste en l'analyse et de données GPS (Global Positioning System) acquises par un récepteur équipant un véhicule et de données acquises par un radar fixe.

Les données GPS sont étudiées dans la seconde partie, alors que les données radar sont étudiées dans une troisième partie.

1 Récepteur GPS

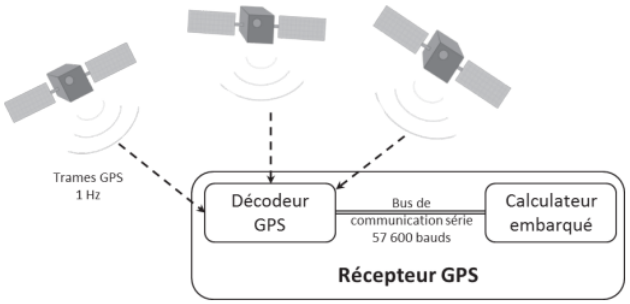
Les données GPS sont obtenues via un dispositif de réception installé à bord et fonctionnant à une fréquence allant de 1 à 4 Hertz. Chacune des mesures reçues contient (entre autre) des informations de positionnement (latitude, longitude) et de vitesse. Le décodeur de trames inclus dans le récepteur GPS envoie les informations au calculateur embarqué sur un bus de communication suivant un protocole particulier. Dans le cadre de ce sujet, le protocole retenu est le protocole NMEA 0183 (National Marine Electronics Association), utilisant des trames de type GPRMC (Recommended minimum specific GPS/Transit data) et utilisable avec la plupart des récepteurs GPS. Le format des trames GPRMC est défini en annexe 1.

1.1 Acquisition des trames NMEA via une liaison série

L'objectif de cette première partie est d'enregistrer les trames GPS reçues par le décodeur dans un fichier texte nommé `tramesNMEA.txt`.

Le décodeur GPS reçoit des trames provenant des satellites à la fréquence de 1 Hertz. Elles sont ensuite transmises sur un port série dans le but d'être traitées par le calculateur embarqué.

La figure ci-dessous présente le principe de fonctionnement du système étudié :



1.1.1 Liaison série

Le décodeur GPS envoie sur la liaison série de type RS-232C une trame GPRMC.

De manière générale, une liaison série permet de véhiculer, sur un nombre réduit de supports physiques, des informations élémentaires les unes à la suite des autres.

Dans le protocole RS-232C, pour chaque octet à transmettre, 1 bit de start, 8 bits de données (correspondant à l'octet à transmettre), puis 1 bit de stop sont envoyés sur le bus de communication.

Ainsi, pour chacun des octets d'une trame GPRMC, 10 bits sont transmis. Le signal de communication est bivalent, chaque bit est codé par un signal électrique haut ou bas dont la durée dépend du débit choisi pour la transmission. Au débit de 57 600 bauds choisi ici, 57 600 bits sont donc transmis à chaque seconde.

Nous faisons l'hypothèse que la liaison est fiable, c'est-à-dire que l'on considère qu'il n'y a pas de panne du récepteur ou de déconnexion physique de la liaison série.

On considère une trame de 76 caractères codés en ASCII. On donne en annexe 1 un exemple de trame et en annexe 2 la table ASCII.

Question 1. Au vu de la problématique du sujet, préciser combien d'octets sont nécessaires dans une trame GPS de type GPRMC pour encoder la latitude et la longitude avec leurs précisions Nord/Sud ou Est/Ouest et la vitesse en nœuds. Dans le dénombrement, les séparateurs de champs (virgules) ne seront pas comptabilisés. Votre réponse devra être justifiée.

Les caractères sont codés en ASCII : il suffit donc de compter les octets utiles, étant donné que un caractère compte comme un octet. Le sujet spécifie que les virgules ne doivent pas être comptées. Rien n'est spécifié pour les points apparaissant

dans les champs latitude et longitude. Dans le corrigé nous les compterons, mais il paraît naturel d'accepter une réponse où il ne seraient pas comptés.

La latitude est codée sous la forme ddmm.mmmm et compte donc pour 9 octets (8 si on choisit de ne pas compter les points). L'orientation compte pour un octet.

La longitude est codée sous dddmm.mmmm et compte donc 10 octets (9 si on choisit de ne pas compter les points). L'orientation compte pour un octet.

La vitesse compte pour 4 octets (3 si on choisit de ne pas compter les points).

Au total il y a donc besoin de 25 octets pour encoder les données demandées (22 si on a choisi de ne pas compter les points).

Question 2. Quelle est la durée de transmission d'une trame de 76 caractères sur le bus RS-232C ? On pourra laisser le résultat sous forme de fraction.

Le débit de transmission du bus série est-il suffisant pour traiter en temps réel les trames provenant des satellites ?

Le décodeur GPS reçoit des trames provenant des satellites à la fréquence de 1 Hertz, il reçoit donc une trame par seconde. Une trame faisant 76 caractères, elle sera représentée par 76 octets. Pour chacun de ces derniers 10 bits sont transmis ce qui fait qu'il faut transmettre 760 bits pour une trame.

Ceci correspond à une durée de $\frac{760}{57600} \approx 0,013$s.

Le débit de transmission du bus série est donc (amplement) suffisant.

1.1.2 Lecture des trames GPS

On souhaite utiliser une bibliothèque permettant la gestion des entrées/sorties sur une liaison série. Un extrait de la spécification de cette bibliothèque est donné ci-après en Python.

```
def initSerie(port,bauds):
"""
Initialise un port série et retourne
un identifiant de port série idport
Entrées :
  * port : int, entier >=0
  * bauds : int, débit en bauds par sec
Sortie :
  * idport : int, l'identifiant du port
"""
```

```
def lireSerie(idport):
"""
Attend l'arrivée d'un octet sur le port série
et le retourne sous forme d'un entier
(code ASCII)
Entrée :
  * idport : int, l'identifiant du port
Sortie :
  * car : int, valeur de l'octet lu sur le port série
"""
```

Question 3. Donner l'instruction permettant d'initialiser le port série numéroté « 0 » à une vitesse de 57 600 bauds. L'identifiant du port sera stocké dans une variable nommée `identifiant`.

En se servant de la documentation fournie : `identifiant = initSerie(0,57600)`

Question 4. En utilisant les annexes 1 et 2, déterminer le dernier caractère de la trame.

Préciser quel est son code ASCII sous forme décimale.

Le dernier caractère de la trame est <LF> de code ASCII 10 (sous forme décimale).

Question 5. À partir des informations contenues dans l'annexe 3, indiquer de quelle manière obtenir un caractère à partir de son code ASCII exprimé sous forme décimale.

L'annexe vient rappeler une instruction Python déjà connue : `chr`.

Plus précisément si `code` est un code ASCII exprimé sous forme décimale, `chr(code)` est un caractère correspondant à son code.

Question 6. On donne ci-dessous l'algorithme en pseudo-code de la fonction permettant de lire et de retourner une trame à partir d'un port série préalablement ouvert.

Algorithme : lecture et affichage d'une trame
Fonction lireTrame(Paramètre à définir)
    Entrée : à définir
    Sortie : à définir
    trame ← chaîne de caractères vide
    lu ← 0
    Tant que lu ≠ nombre entier correspondant au caractère de fin de trame en ASCII faire
        lu ← lire octet série correspondant à un caractère
        ajouter en fin de trame le caractère correspondant à lu
    Fin Tant que
    Afficher (trame)
    Retourner (trame)
Fin Fonction lireTrame

Implémenter cette fonction en langage Python en vous servant de l'annexe 3.

Il s'agit là d'une simple traduction du pseudo-code, en se servant des questions précédentes. L'argument de la fonction sera l'identifiant du port, tel que retourné par la fonction d'ouverture du port.

```
def lireTrame(idport):
      trame = ""
      lu = 0
      while lu != 10: # correspond au code ASCII de la fin de trame
            lu = lireSerie(idport) # permet de lire la caractère
            trame += chr(lu) # convertit en caractère et ajouté (de préférence avec +=) à trame
      print(trame) # comme spécifié par l'énoncé
      return trame
```

1.1.3 Enregistrement des trames

Nous souhaitons maintenant enregistrer toutes les trames envoyées par le récepteur GPS dans un fichier texte nommé situé dans le dossier courant. Le fichier texte sera ouvert en début de programme de sorte à être vidé s'il existait déjà auparavant. Ici, le récepteur GPS envoyant des trames en continu, le programme ne se terminera que lorsqu'il sera explicitement fermé par l'utilisateur. Pour implémenter ce comportement, on supposera que le programme continuera à stocker des données tant qu'un booléen `GPS` est vrai et que ce booléen est géré par le matériel du port série.

L'annexe 3 présente des fonctions en langages Python permettant de manipuler les fichiers.

Question 7. En utilisant les fonctions précédentes, écrire en Python le programme principal qui, après initialisation du port de la liaison série, permet de lire les trames sur le port série et d'enregistrer ces dernières, telles qu'elles sont reçues, dans le fichier `tramesNMEA.txt`, au format texte.

Il s'agit là d'une mini-synthèse du travail précédent :

```
identifiant = initSerie(0,57600)
GPS = True
fichier = open('tramesNMEA.txt','w') # on se contente des arguments nécessaires !
while GPS: # plutôt que GPS == True qui est lourd
      trame = lireTrame(identifiant)
      fichier.write(trame)
fichier.close() # sans quoi l'écriture n'a pas lieu physiquement.
```

1.2 Exploitation du fichier de trames

L'objectif de cette partie est d'extraire et d'enregistrer une trame dans une structure de données cohérente afin de pouvoir exploiter aisément les mesures.

On fait l'hypothèse que les trames transmises ne contiennent pas d'erreur.

1.2.1 Extraction d'une trame

La description d'une trame GPMRC, donnée en annexe 1, indique que les valeurs des différentes informations de la trame sont comprises entre les caractères "$" et "*".

Nous souhaitons disposer d'une fonction susceptible de retourner la position de la première occurrence d'un caractère ASCII (exemple d'utilisation de la fonction : rechercher la position de la première occurrence de « $ » dans une trame).

Question 8. Écrire la fonction `cherchePremiereOccurence(chaine,car)` qui retourne l'indice de la première occurrence du caractère `car` dans la chaîne de caractères `chaine`. Il convient de choisir et de justifier ce que la fonction retourne dans le cas où le caractère recherché n'est pas présent dans la chaîne.

Il s'agit d'une question du cours de première année ! On convient dans ce corrigé de renvoyer `None` si le caractère cherché n'est pas présent dans la chaîne de caractères considéré. On se sert du fait que `return` interrompt l'exécution d'une fonction, la boucle `for` programmée équivaut donc à une boucle `while`. On choisit évidemment un parcours par indice de la chaîne de caractères.

```
def cherchePremiereOccurence(chaine,car):
      for i in range(len(chaine)): # itération sur les indices
            if chaine[i] == car:
                  return i
      return None # optionnel.
```

On cherche à concevoir l'algorithme de la fonction `extraireTrame(NMEA)`.

Cette fonction doit permettre d'extraire et de retourner, au format chaîne de caractères, le contenu d'une trame (nommée `NMEA`) situé strictement entre les caractères "$" et "*" en s'assurant au préalable que ces caractères sont bien présents dans la chaîne. La trame extraite se nomme `NMEA_extraite`.

A titre d'exemple, le contenu de la trame `NMEA_extraite` issue de la fonction `extraireTrame(NMEA)`, basée sur la trame donnée en exemple dans l'annexe 1 est de la forme :

GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A

Question 9. Indiquer quelles sont les conditions nécessaires et suffisantes que doit respecter une trame quelconque afin de pouvoir être extraite grâce à la fonction `extraireTrame(NMEA)`.

Une trame quelconque doit contenir les caractères "$" et "*" et ce dans cet ordre pour pouvoir être extraite par la fonction `extraireTrame(NMEA)`.

On peut imaginer une situation d'une chaîne fictive de la forme "*$trame*" qui soit une trame permettant une extraction. Ainsi il n'est pas forcément nécessaire que la première occurrence de "*" soit avant celle de "$". Concrètement, il est nécessaire et suffisant que :

- la chaîne contienne une occurrence de "$" ;
- la chaîne contienne une occurrence de "*" après celle de "$".

Question 10. À partir des fonctions proposées dans cette partie, établir une implémentation de la fonction `extraireTrame()` dans laquelle la chaîne d'entrée se nomme `NMEA` et la chaîne de sortie `NMEA_extraite`. Dans le cas où la chaîne ne peut pas être extraite, la fonction doit retourner une chaîne vide.

En reprenant les conditions nécessaires et suffisantes de la question précédente :

```
def extraireTrame(NMEA):
      a = cherchePremiereOccurence(NMEA,"$")
      b = cherchePremiereOccurence(NMEA,"*")
      while not b is None: # la comparaison à None se fait à l'aide du test d'adressage is (cas particulier)
            if b>a:
                  return NMEA[a+1:b] # slicing ; b est exclu et on exclu a à la main
            b = cherchePremiereOccurence(NMEA,"*")
      return "" # on traite le cas particulier.
```

1.2.2 Structure de données

Dans le cadre des traitements ultérieurs, il nous faut représenter une mesure GPS, que nous nommerons par la suite `pointGPS`, par sa latitude, sa longitude et sa vitesse.

Dans la trame GPS, les longitudes et latitudes sont données en degrés et minutes. Cette représentation ne facilitant pas les calculs, il est nécessaire de les convertir en minutes (1 degré étant égal à 60 minutes). Par ailleurs, on opte pour la convention suivante :

- pour la latitude, le signe positif est choisi pour le Nord ;
- pour la longitude, le signe positif est choisi pour l' Est.

Ainsi par exemple, si la trame contient la latitude «0314.2500» suivie de l'indicateur «S», la séquence «03» correspond à 3 degrés soit 180 minutes d'angle, la séquence «14.2500» correspond à 14,25 minutes d'angle et l'indicateur «S» implique que la latitude doit être précédée du signe moins. Ainsi, pour la latitude, on stockera -194.25 dans le `pointGPS` . Cette conversion est assurée par la fonction `conversionLongLat2Min` dont l'aide est fournie ici :

```
def conversionLongLat2Min(ll,orientation):
"""
Conversion de la latitude ou de la longitude en minutes
(nombre entier positif ou négatif)
Entrées :
  * ll : str, latitude ou longitude sous la forme ddmm.mmmm
  *orientation : str, vaut "S","N","W" ou "E"
Sortie :
  * res : float,résultat en minutes d'angle
"""
```

Afin de stocker un on utilise une liste qui présente la structure interne suivante :
`pointGPS = [latitude, longitude, vitesse]`

Cette liste sera renseignée par les latitudes, longitudes et les vitesses extraites des différentes trames envoyées par le décodeur et traitées par les fonctions précédentes.

Il existe deux principales méthodes pour extraire ces données, une première repose sur le comptage des séparateurs et peut s'adapter à des longueurs variables pour un même champ. La seconde, plus simple à mettre en œuvre, repose sur l'hypothèse que toutes les trames présentent exactement la même structure et que toutes les longueurs de champs restent identiques entre les différentes trames reçues.

Dans la suite du sujet, nous ferons l'hypothèse que toutes les longueurs de champs sont toujours identiques dans les différentes trames à traiter.

On rappelle qu'en appliquant la fonction (définie dans la partie précédente) à la trame donnée en annexe 1, on obtient :
GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A

Question 11. À partir de la trame extraite issue de l'annexe 1 et rappelée ci-dessus compléter le tableau donnant l'index de début et l'index de fin des informations nécessaires à l'extraction de la latitude (y compris l'orientation), de la longitude (y compris l'orientation) et de la vitesse.

	Latitude	Orientation	Longitude	Orientation	Vitesse
Index de début	19	29	31	42	44
Index de fin	27	29	40	42	47

Remarque : les index de fin des informations sont considérés comme faisant partie de l'information (il s'agit donc de la convention inverse de celle prise par `range`.

Question 12. Écrire la fonction `creationPointGPS(NMEA_extraite)` fonction qui prend en argument une chaîne de caractères, contenu de la trame entre les caractères $ et * et qui renvoie un `pointGPS`, qui est une liste de la forme `[latitude, longitude, vitesse]`. On utilisera pour cela la fonction ainsi que les informations données précédemment.

```
def creationPointGPS(NMEA_extraite):
      lat,lato = NMEA_extraite[19:28],NMEA_extraite[29]
      long,longo = NMEA_extraite[31:41],NMEA_extraite[42]
      vit = NMEA_extraite[44:48]
      pointGPS = [conversionLongLat2Min(lat,lato), conversionLongLat2Min(long,longo),float(vit)]
      return pointGPS
```

1.2.3 Affichage du fichier

Nous disposons maintenant de toutes les fonctions qui permettent d'exploiter le fichier de trames créé dans la première partie (`tramesNMEA.txt`). Nous souhaitons à présent afficher successivement chacun des `pointGPS` présents dans le fichier sur une console texte.

Pour la gestion des fichiers, on pourra utiliser les fonctions `close()`, `open()`, `readline()` et `write()`, dont les descriptions sont proposées en annexe 3. On précise qu'il n'est pas indispensable d'utiliser toutes ces fonctions.

Question 13. Le fichier contient les éléments correspondant aux `pointGPS`. Établir une fonction sans argument qui affiche sur la console chacun de ces éléments, en vue de l'analyse par la gendarmerie des endroits où sont commis les excès de vitesse.. Chaque ligne de la console devra être de la forme suivante :

```
latitude_Pt_1    longitude_Pt_1    vitesse_Pt_1
latitude_Pt_2    longitude_Pt_2    vitesse_Pt_2
```

Chaque espace correspond à une tabulation.

Nous utiliserons dans ce corrigé une version plus proche de ce qui a été vu durant l'année, avec l'usage de la méthode `read` sur les fichiers et de la méthode `split` sur les chaînes de caractères :

```
def affiche():
      f = open("tramesNMEA.txt","r")
      s = f.read()
      f.close()
      ListeTrames.split(chr(10)) # Les trames se terminent par un caractère de code ASCII 10
      ListePoints = [ExtraireTrame(trame) for trame in ListeTrames]
      for point in ListePoints:
            pointGPS = creationPointGPS(point)
            print(str(pointGPS[0]) + '\t' + str(pointGPS[1]) + '\t' + str(pointGPS[2]) + '\n'
            # La tabulation est créé à l'aide de \ t ; pointGPS est une liste de flottants.
```

1.3 Exploitation des données

Pour une analyse plus simple des excès de vitesse commis, les données présentes dans chacun des `pointGPS` présents dans le fichier de trames créé dans la première partie (`tramesNMEA.txt`) doivent être triés.

Question 14. Rappeler les trois méthodes de tris au programme d'informatique et leurs complexités dans le meilleur et le pire des cas.

Les deux questions qui suivent sont des questions de cours :

Le tri par insertion a une complexité dans le pire des cas en $\mathcal{O}(n^2)$ et dans le meilleur des cas en $\mathcal{O}(n)$.
Le tri rapide a une complexité dans le pire des cas en $\mathcal{O}(n^2)$ et dans le meilleur des cas en $\mathcal{O}(n \log n)$.
Enfin, le tri fusion a une complexité dans le pire et dans le pire des cas en $\mathcal{O}(n \log n)$.

Question 15. Écrire une fonction `tri_rapide(L)` qui effectue le tri rapide d'une liste quelconque de nombres.

```
def tri_rapide(L):
      if len(L) <= 1:
            return L
      else:
            pivot = L[0]
            Lg = [L[i] if L[i]<=pivot for i in range(1,len(L))]
            Ld = [L[i] if L[i]>pivot for i in range(1,len(L))]
            return tri_rapide(Lg) + [pivot]+tri_rapide(Ld)
```

Question 16. Quelles adaptations seraient nécessaires aux fonctions précédentes pour que l'affichage de la dernière partie se fasse dans l'ordre des vitesses décroissantes ?

Deux adaptations seraient nécessaires :
- Tout d'abord, dans la fonction de la question 13, on créerait une liste de listes de points GPS, qui seraient des flottants, et on éviterait donc le parcours direct de `ListePoints`.
- Ensuite, on effectuerait un tri de `ListePoints` selon sa dernière composante (et par ordre décroissant) : il faudrait pour cela adapter la fonction de tri rapide, pivot serait posé égal à L[0][2] ; Lg serait constitué des L[i] tels que L[i][2]>pivot et Ld des L[i] tels que L[i][2]<= pivot.

2 Intermède sur les bases de données

Dans cette partie, nous nous intéressons à une base de données constituée à partir de données GPS relevées.

Celles-ci sont stockées dans une base de données contenant deux tables, *Véhicules* et *Vitesses*.

Les structures de ces deux tables sont les suivantes :

Véhicules
id_vehicule
Propriétaire
Immatriculation

Vitesses
id_vitesse
releve
max
id_vehicule
date

Tous les attributs sont de type numérique sauf propriétaires, et immatriculation qui sont des chaînes de caractères.
La table *Véhicules* comporte un attribut **id_vehicule**, qui est une clé primaire pour un véhicule donné.
La table *Vitesses* comporte attribut **id_vitesse**, qui est une clé primaire pour une mesure donnée.
Voici des exemples d'enregistrements pour cette base de données :

Véhicules		
id_vehicule	propriétaire	immatriculation
1	Alain B	6451-ABC-13
2	Claude D	7654-ZZ-75
3	Estelle F	7544-DFE-69

Vitesses				
id_vitesse	releve	max	id_vehicule	date
1	145	130	1	01/09/2011
2	152	110	1	01/02/2014
3	108	130	1	17/03/2015
4	120	70	2	08/01/2016

Question 17. Écrire une requête SQL affichant le nom de tous les propriétaires enregistrés dans la base de données.

```
SELECT Propriétaire
FROM Véhicules;
```

Question 18. Une vitesse est appelée un « grand excès de vitesse » lorsque la vitesse relevée (stockée dans le champ **releve**) est au moins égale à plus de 50 kilomètres par heure à la vitesse autorisée (stockée dans le champ **max**).
Écrire une requête SQL affichant le nombre de grand excès de vitesses contenus dans la base de données.

On se sert ici de la fonction d'agrégation `COUNT` dans la requête :

```
SELECT COUNT(*)
FROM Vitesses
WHERE releve-50 >= max;
```

Question 19. Écrire une requête SQL affichant tous les numéros d'immatriculation de véhicules ayant commis des excès de vitesse.

Nous avons ici besoin d'accéder aux deux tables présentes dans la base, nous allons donc utiliser une jointure :

```
SELECT immatriculation
FROM Véhicules JOIN Vitesses ON Véhicules.id_vehicule = Vitesses.id_vehicule
WHERE releve > max;
```

Question 20. Écrire une requête SQL affichant le nombre de conducteurs ayant commis plus de trois grands excès de vitesse.

La fonction d'agrégation porte cette fois-ci sur une sélection : nous allons donc nous servir des clauses `GROUP BY` et `HAVING` de SQL :

```
SELECT COUNT(*)
FROM Véhicules JOIN Vitesses ON Véhicules.id_vehicule = Vitesses.id_vehicule
GROUP BY Véhicules.id_vehicule
HAVING COUNT(releve-max>50) >= 3 ;
```

3 Étude d'un radar

Dans cette partie, nous nous intéressons à un dispositif de mesure de la vitesse : le radar (pour RAdio Detecting And Ranging).

Le principe utilisé par les radars est voisin de celui de la réflexion des ondes sonores. Le radar utilise des impulsions d'énergie électromagnétique. Le signal, de haute fréquence, est émis en direction d'une cible. Une petite partie de l'énergie transmise est réfléchie par la cible dans la direction du radar. Cette énergie renvoyée par la cible jusqu'au radar est appelée écho, exactement comme lorsque l'on considère les ondes sonores. Un radar utilise l'écho afin de déterminer la direction et la distance de l'objet qui a réfléchi son signal.

Un radar Doppler pulsé est un radar capable, non seulement de donner le cap, la distance et l'altitude d'une cible, mais aussi de mesurer sa vitesse radiale, et donc de distinguer les cibles en mouvement des objets permanents. Pour cela il utilise l'effet Doppler. Il émet des impulsions cohérentes par une antenne ou un transducteur. Celles-ci sont superposées à la fréquence porteuse du signal radar. Entre chaque impulsion, l'antenne et le circuit électronique sont mis à l'écoute de l'impulsion de retour. On calcule la distance entre le radar et les cibles par la relation suivante (le facteur $frac12$ tient compte que t correspond au temps d'aller-retour du signal) :

Distance $= c\,\dfrac{\Delta t}{2}$ (c = vitesse de la lumière).

Les radars Doppler pulsés modernes possèdent un étage de démodulation du signal entrant qui centre la fréquence sur zéro, en retirant la fréquence porteuse, avant l'échantillonnage numérique des échos. Le signal résultant contient non seulement l'intensité de l'écho mais aussi la variation de la fréquence causée par l'effet Doppler d'une cible s'éloignant ou se rapprochant du radar.

Entre chaque impulsion, le processeur radar fait un échantillonnage du signal retourné au radar. Nous disposerons donc de N mesures du signal (qui est une énergie électromagnétique captée et démodulée) et nous allons chercher dans un premier temps à représenter le spectre du signal traité, puis dans un second temps à trouver la fréquence maximale.

La Transformée de Fourier Discrète (TFD) permet seulement d'évaluer une représentation spectrale discrète (spectre échantillonné) d'un signal discret (signal échantillonné) sur une fenêtre de temps finie (échantillonnage borné dans le temps).

La transformée de Fourier discrète est l'ensemble des $S(k)$ donnés par la formule :

$$S(k) = \sum_{n=0}^{N-1} s(n)\mathrm{e}^{-2j\pi k\frac{n}{N}}$$

où $j^2 = -1$ et $0 \leqslant k < N$, et les mesures $s(n)$ avec $0 \leqslant n < N$ sont connues.

Question 21. Écrire une fonction **module(a,b)** retournant le module du complexe $a + jb$.

```
def module(a,b):
    return (a**2+b*2)**0.5
```

Question 22. Écrire une fonction **trans_fourier(s)** retournant les parties réelles et imaginaires de $S(k)$ défini comme la composante de la transformée de Fourier discrète du signal s, représenté par une liste.

Il y a ici une petite ambiguïté quant à la nature des objets à renvoyer ; nous faisons le choix de renvoyer deux listes. Le choix de la lisibilité (par rapport au nombre de lignes) a par ailleurs été fait ici :

```
from math import cos, sin, pi
def trans_fourier(s):
      SR, SI = [], [] # listes des composantes réelles et imaginaires des transformées de Fourier
      N = len(s)
      for k in range(N):
            sr = 0
            si = 0
            for n in range(N):
                  sr = sr + s[n]*cos(2*pi*k*n/N)
                  si = si + s[n]*sin(2*pi*k*n/N)
            SR.append(sr)
            SI.append(si)
      return SR, SI
```

Question 23. Écrire une fonction **trans_fourier2(s)** retournant le module de la transformée de Fourier discrète de s sous la forme d'une liste contenant les modules des termes $s(k)$ en utilisant des appels aux fonctions précédentes.

Une petite erreur d'énoncé s'est glissée ici : les deux fonctions ont le même nom. La correction rectifie ce point :

```
def trans_fourier2(s):
      S = []
      SR, SI = trans_fourier(s)
      for i in range(len(SR)):
            S.append(module(SR[i],SI[i]))
      return S
```

Question 24. Écrire une suite d'instructions permettant d'afficher graphiquement le spectre de s calculé à l'aide de la TFD (donc les $s(k)$ en fonction de k).

```
import matplotlib.pyplot as plt

A = list(range(n))
B = trans_fourier2(s)
plt.plot(A,B)
plt.xlabel("fréquences")
plt.ylabel("intensité spectrale")
```

Question 25. Écrire une fonction qui prend en argument une liste s de mesures et renvoie le maximum des modules de la transformée de Fourier discrète de s.

Il suffit de reprendre la fonction précédente :

```
def maxmodfourier(s):
      return max(trans_fourier2(s))
```

Remarque : il aurait été intéressant de renvoyer la fréquence plutôt que l'intensité du maximum.

Question 26. Déterminer la complexité du calcul de la transformée de Fourier par la fonction **trans_fourier(s)** en fonction de N

On reprend la fonction `trans_fourier(s)` ; elle comporte une double boucle imbriquée, répétées chacune N fois, et un nombre constant d'opérations à l'intérieur de celle-ci : la complexité de la transformée de Fourier discrète est en $\mathcal{O}(n^2)$.

La complexité obtenue ici est un frein à l'application de cette technique en temps réel, nécessaire en pratique lors de mesures au radar.

Il existe un autre algorithmique pour déterminer la TFD d'un signal, appelé Transformée de Fourier Rapide (*Fast Fourier Transform*, couramment abrégé comme FFT). Cet algorithme célèbre a été inventé par Cooley et Tukey, ingénieurs dans le centre de recherche d'IBM au début des années 1960.

Pour expliquer cet algorithme, nous utiliserons la récursivité en montrant que le calcul d'une transformée de Fourier de taille N se ramène au calcul de deux transformée de Fourier de taille $N/2$ suivi de $N/2$ multiplications. On veut calculer pour $k = 0, \ldots, N-1$:

$$S(k) = \sum_{t=0}^{N-1} s(t) \exp\left(-2\pi j \frac{k.t}{N}\right)$$

On pose $t = 2n$ si t est pair et $t = 2n+1$ si t est impair. $S(k)$ s'écrit alors, en posant $M = \frac{N}{2}$:

$$S(k) = \sum_{n=0}^{M-1} s(2n) \exp\left(-2\pi j \frac{k.2n}{N}\right) + \sum_{n=0}^{M-1} s(2n+1) \exp\left(-2\pi j \frac{k.(2n+1)}{N}\right) \quad (*)$$

Nommons les séquences :

$t = 0, \ldots, 2M-1 : s_{2M}(t) = s(t)$

$n = 0, \ldots, M-1 : s^o_M(n) = s(2n)$

$n = 0, \ldots, M-1 : s^i_M(n) = s(2n+1)$

$k = 0, \ldots, 2M-1 : S_{2M}(k) = S(k)$

Avec ces notations, l'équation (*) devient pour $k = 0, \ldots, 2M-1$

$$S_{2M}(k) = \left[\sum_{n=0}^{M-1} s^o_M(n) \exp\left(-2\pi j \frac{k.n}{M}\right)\right] + \exp\left(-\pi j \frac{k}{M}\right) \left[\sum_{n=0}^{M-1} s^i_M(n) \exp\left(-2\pi j \frac{k.n}{N}\right)\right]$$

Finalement, si $0 \leqslant k \leqslant M-1$:

$S_{2M}(k) = S^o_M(k) + \exp\left(-\pi j \frac{k}{M}\right) S^i_M(k)$

et

$S_{2M}(k+M) = S^o_M(k) - \exp\left(-\pi j \frac{k}{M}\right) S^i_M(k)$

Question 27. Écrire une fonction `testpuissance2(L)` qui prend en argument une liste `L` quelconque et qui renvoie un booléen indiquant si la longueur de cette liste est une puissance de 2.

Il est implicite que la puissance de 2 considérée est nécessairement positive (on parle de longueur de liste, entière par définition). Dès lors il suffit de tester si les divisions successives de la longueur de la liste par 2 restent entières, avec comme condition d'arrêt que le quotient finisse par être égal à 1 :

```
def testpuissance2(L):
      n = len(L):
      while n! = 1:
            if n%2 == 1:
                  return False
            n = n//2
      return True
```

Question 28. Écrire une fonction `TransformeeFourierRapide(s)` qui prend en argument une liste de mesures `s` de taille 2^p et qui calcule à l'aide de la transformée de Fourier rapide les parties réelles et imaginaires de $S(k)$ défini comme la composante de la transformée de Fourier discrète du signal s, représenté par une liste.

Il est naturel de travailler avec des nombres complexes :

```
 def FFT(s):
   N = len(s)
   if N == 1:
         return s
   P, I = FFT(s[0::2]), FFT(s[1::2])
   S = [0 for i in range(N)]
   omega = complex(cos(2*pi/N),sin(2*pi/N))
   x = 1
   for j in range(N//2):
         S[j] = P[j]+x*I[j]
         S[j+N//2] = P[j] - x*I[j]
         x *= omega
   return S
```

Question 29. Montrer que la complexité de la transformée de Fourier rapide est en $\mathcal{O}(N\log(N))$.

À chaque exécution, il y a $\mathcal{O}(n)$ opérations élémentaires qui sont effectuées, où n est la taille de la liste considérée.

À chaque niveau d'appel récursif, des transformées de Fourier rapides sont effectuées sur des listes dont la taille totale est N. Ainsi, chaque niveau d'appel récursif coûte $\mathcal{O}(N)$ opérations élémentaires.

Il reste à déterminer le nombre de niveaux d'appels récursifs. À chaque niveau d'appel récursif, les listes sur lesquelles sont calculées les FFT ont une taille qui est divisée par 2. Ainsi au k-ème niveau d'appel récursif, les listes sont de taille 2^{p-k}. Or l'algorithme s'arrête lorsque les listes considérées sont de taille 1. Ceci signifique que l'on a $2^{p-k}=1$ ou encore $p-k=0$, donc $k=p$. Comme $2^p=N$ on a $p=\log_2(N)$ niveaux d'appels récursifs.

Au final, la complexité de la FFT est bien en $\mathcal{O}(N\log(N))$.

Question 30. Quelles adaptations seraient nécessaire à l'algorithme de la transformée de Fourier rapide pour traiter le cas d'une liste dont la longueur n'est pas une puissance de 2 ?

L'algorithme présenté ici ne suffit plus. Le problème est très loin d'être trivial, et d'autres algorithmes de FFT existent (algorithme de Vinograd, algorithme de Rader...), s'appuyant sur d'autres idées arithmétiques. Face à ce genre de questions ouvertes (et de dernière question de concours), il convient de ne pas être trop gourmand, et de rester honnête et humble tout en essayant de proposer une généralisation, qui sera nécessairement valorisée de part le fait qu'elle départagera les meilleurs candidats.

On pourrait par exemple proposer d'étendre la formulation $P(X)=P_0(X^2)+XP_1(X^2)$ en

$$P(X)=P_0(X^k)+XP_1(X^k)+X^2P_2(X^k)+\cdots+X^{k-1}P_{k-1}(X^k)$$

et de transformer ainsi une FFT de taille km en k FFT de taille m. Il s'agit de l'algorithme PFA, pour `Prime Factor Algorithm`.

Notons que l'algorithme FFT, qui permet notamment une multiplication d'entiers plus rapide que l'algorithme « naïf » de l'école primaire, reste aujourd'hui l'objet de recherches actives et d'optimisations variées, également liées à l'évolution du matériel informatique (taille des caches, parallélisme etc...).

4 Annexe 1 : Le protocole NMEA 0183, type GPRMC

Dans le protocole NMEA 0183, chaque trame est constituée de chaînes de caractères, terminées par un retour chariot et un saut de ligne. Les caractères constituant la trame ont un code ASCII compris entre 20 et 127 mais sont codés sur un octet. Toute trame commence par les caractères $GP, suivis de trois caractères définissant le type de trame. Chaque trame de type GPRMC est donc préfixée par $GPRMC.

On donne ci-dessous un exemple de trame :

$GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A*65<CR><LF>

Les trames (cf exemple ci-dessus), sont constituées de 13 champs texte, séparés par des virgules et terminés par une somme de contrôle de parité, résultat d'un *ou exclusif* de chaque octet compris strictement entre le caractère initial "$" et le caractère "*". L'interprétation de chaque champ dépend alors de sa position.

Position	Exemple	Interprétation
0	$GPRMC	Type de trame. Nous considérons uniquement le type RMC.
1	071139.988	Heure UTC de la mesure sous la forme hhmmss.sss (avec hh heures, mm minutes dans l'heure, ss.sss secondes et secondes décimales)
2	A	"A" pour mesure valide, "V" pour mesure invalide
3	4639.8232	Latitude donnée sous la forme ddmm.mmmm (avec dd degrés, mm.mmmm minutes et minutes décimales)
4	N	Complète le champ 3 pour préciser s'il s'agit de la latitude Nord ("N") ou Sud ("S")
5	00021.6810	Longitude donnée sous la forme dddmm.mmmm
6	E	Complète le champ 5 pour préciser s'il s'agit de la longitude Est ("E") ou Ouest ("W")
7	8.95	Vitesse en nœuds
8	184.14	Cap du déplacement par rapport au pôle Nord magnétique en degrés décimaux
9	141014	Date universelle donnée sous la forme jjmmaa (avec jj jour du mois, mm mois, aa année)
10	1.1	Variation magnétique sous la forme d.d en degrés décimaux.
11	W	Sens de la variation magnétique ("E" = Est, "W" = Ouest)
12	A	Mode de fonctionnement du récepteur
	*65	Somme de contrôle exprimée en base hexadécimale sur deux chiffres
fin	<CR><LF>	La trame se termine par deux caractères : un retour chariot (code ASCII 13) puis un saut de ligne (code ASCII 10).

5 Annexe 2 : Table ASCII

La table suivante donne les correspondances entre codes décimaux et caractères ASCII :

Décimal	Caractère	Description	Décimal	Caractère	Décimal	Caractère	Décimal	Caractère
0	NUL	Null	32	Space	64	@	96	‘
1	SOH	Start of heading	33	!	65	A	97	a
2	STX	Start of text	34	”	66	B	98	b
3	ETX	End of text	35	#	67	C	99	c
4	EOT	End of transmission	36	$	68	D	100	d
5	ENQ	Enquiry	37	%	69	E	101	e
6	ACQ	Acknowledge	38	&	70	F	102	f
7	BEL	Bell	39	’	71	G	103	g
8	BS	Backsapce	40	(	72	H	104	h
9	TAB	horizontal tab	41	)	73	I	105	i
10	LF	New line feed, new line	42	*	74	J	106	j
11	VT	Vertical tab	43	+	75	K	107	k
12	FF	NP form feed, new page	44	‘	76	L	108	l
13	CR	Carriage return	45	-	77	M	109	m
14	SO	Shift out	46	.	78	N	110	n
15	SI	Shift in	47	/	79	O	111	o
16	DLE	Data link espace	48	0	80	P	112	p
17	DC1	Device control 1	49	1	81	Q	113	q
18	DC2	Device control 2	50	2	82	R	114	r
19	DC3	Device control 3	51	3	83	S	115	s
20	DC4	Device control 4	52	4	84	T	116	t
21	NAK	Negative acknowledge	53	5	85	U	117	u
22	SYN	Synchronous idle	54	6	86	V	118	v
23	ETB	End of trans. block	55	7	87	W	119	w
24	CAN	Cancel	56	8	88	X	120	x
25	EM	End of medium	57	9	89	Y	121	y
26	SUB	Substitute	58	:	90	Z	122	z
27	ESC	Escape	59	;	91	[	123	{
28	FS	File separator	60	<	92	\	124	|
29	GS	Group separator	61	=	93	]	125	}
30	RS	Record separator	62	>	94	^	126	~
31	US	Unit separator	63	?	95	_	127	DEL

6 Annexe 3

Fonctions Python permettant de manipuler les codes ASCII des caractères :

`chr(i)` : Return the string representing a character whose Unicode codepoint is the integer i. For example, chr(97) returns the string ’a’. This is the inverse of ord(). The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). ValueError will be raised if i is outside that range.

Remarque : pour i compris entre 0 et 127, la fonction retourne le caractère correspondant au code ASCII.

`ord(i)` : Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, ord(’a’) returns the integer 97 and ord(’2020’) returns 8224. This is the inverse of chr().

Fonctions de traitement de fichiers :

`close()` : Flush and close this stream. This method has no effect if the file is already closed. Once the file is closed, any operation on the file (e.g. reading or writing) will raise a ValueError. As a convenience, it is allowed to call this method more than once ; only the first call, however, will have an effect.

`open(file,mode='r',buffering=-1,encoding=None,errors=None,newline=None,closefd=None,opener=None)` :
Open file and return a corresponding file object. If the file cannot be opened, an OSError is raised.
file is either a string or bytes object giving the pathname.
mode is an optional string that specifies the mode in which the file is opened. It defaults to ’r’ which means open for reading in text mode. Other common values are ’w’ for writing (truncating the file if it already exists), ’x’ for exclusive creation and ’a’ for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position).

`readline(size=1)` : Read until newline or EOF and return a single str. If the stream is already at EOF, an empty string is returned. If size is specified, at most size characters will be read.

`write(s)` : Write the string s to the stream and return the number of characters written.