DS4 - INFORMATIQUE
DETECTION D'OBSTACLES PAR UN SONAR DE SOUS-MARIN
CORRIGE

```
## Q1 - Avec arange
dt=Tf/(N-1)                  # Incrément de temps en secondes ; N-1 intervalles
temps=arange(0,Tf+dt/2,dt) # liste des instants en secondes (Tf+dt/2 exclu)

# ou linspace
temps=linspace(0,Tf,N)

## Autre option avec une boucle
temps=zeros(N)
dt=Tf/(N-1)
for i in range(N):
      temps[i] = i*dt

## Q2 - Avec une fonction intermédiaire
def chirp(temps,f0,Deltafe,T,E0):
      def fe(t):
            return(f0+t*Deltafe/T)

      e=zeros(len(temps)) # e : vecteur nul de même dimension que le vecteur temps
      for i in range(len(temps)):
            if temps[i]<=T:
                  e[i]=E0*sin(2*pi*fe(temps[i])*temps[i])
      return(e)
```

Q3 - Utilisation de la fonction stft

```
n=256 # Pour l'exemple
f,t,S=stft(s,fs=fe,window="hamming",nperseg=n,noverlap=n//2)
```

- Nombre de fft réalisées = len(t) = N//(n//2) avec N le nombre de frames du signal initial et n le nombre de frames de chaque segment analysé avec recouvrement de n//2
- Nombre de fréquences de la FFT : len(f) = N // (n//2)
- Nombre d'amplitudes pour chaque fréquence de la FFT : S.shape = (N//(n//2), N//(n//2))

Remarque : la fonction sftf de scipy donne, par défaut, un nombre de fréquences et d'amplitudes de n//2+1. S a une dimension adaptée aussi.

Q4

Le spectrogramme permet de filtrer le bruit en faisant ressortir uniquement les fréquences à hautes amplitudes. On reconnait bien ici la forme d'un signal modulé de 400 à 600Hz sur une durée de 0,1s. Il faudra vérifier que le matériau réfléchissant (acier ou béton) modifie significativement le spectrogramme afin de l'utiliser comme moyen de différenciation.

Q5 : fonction w(t,f,eta)

```
def w(t,f,eta):
      t_eta=DT/2+eta*T
      f_eta=Df/2+eta*Df
      if abs(t-t_eta)<DT/2 and abs(f-f_eta)<Df/2:
            return(1)
      else:
            return(0)
```

```
#ou, si t et f sont des flottants (ou des tableaux numpy de même dimension).
def w(t,f,eta):
      t_eta=DT/2+eta*T
      f_eta=Df/2+eta*Df
      return abs(t-t_eta)<DT/2 and abs(f-f_eta)<Df/2

## Q6 : fonction enveloppe
def enveloppe(eta,S,p,dt,df):
      P_eta=0
      for i in range(S.shape[0]):       # ou range(len(t))
            for j in range(S.shape[1]): # ou range(len(f))
                  P_eta+=p[i,j]*S[i,j]*w(t[i],f[j],eta)*dt*df
      return P_eta

## Q7 : fonction normalisation(P)
def normalisation(P):
      return( (P-min(P)) / (max(P)-min(P)) )

## Q8 :
[6, 0.85, [3, 0.89, [3, 0.31, 1, 0], 1], [1, 0.67, [4, 0.8, 1, 1], 1]


## Q9 :
Soit une donnée non classée a :
a=[0.5,0.2,0.7,0.4,0.9,0.25,0.7,0.7,0.9,0.2]
a[6]=0.7<0.85
a[3]=0.4<0.89
a[3]=0.4>=0.31
Groupe 0 : l'obstacle semble être de la roche

## Q10 : fonction indices_aleatoires(m,p_var)
from random import randrange
def indices_aleatoires(m,p_var):
      liste_indices_aleatoires=[]  # Initialisation de la liste (vide pour
l'utilisation de append)
      for i in range(p_var):
            indice=randrange(m)
            while indice in liste_indices_aleatoires: # On boucle tant que l'indice
est déjà dans la liste
                  indice=randrange(m)
            liste_indices_aleatoires.append(indice)    # Une fois le nouvel indice
différent trouvé, on le rajoute à la liste
      return liste_indices_aleatoires

# Q11 : fonction [gauche,droite]=test_separation(ind, val, donnees)

def test_separation(ind, val, donnees):
      gauche, droite = [],[]
      for i in range(donnees.shape[0]):
            if donnees[i,ind]<val:
                  gauche.append(donnees[i,:])
            else:
                  droite.append(donnees[i,:])
      return [gauche,droite]
```

```
## Q12 : 5 instructions à compléter dans la fonction Gini_groupes
def Gini_groupes ( groupes ) :
      #nombre de données total
      n_donnees = len(groupes[0])+len(groupes[1])# instruction 1
      gini = 0.0 #somme pondérée des indices Gini de chaque groupe
      for donnees in groupes :
            taille = len(donnees) # taille d'un groupe
            if taille != 0 :
                  gini_gr = 0.0
                  for val in [ 0 , 1 ] :
                        p=0
                        for ligne in donnees :
                              if ligne[-1] == val :
                                    p=p+1  # instruction 2
                        p=p/taille # instruction 3
                        gini_gr += 1-p**2  # instruction 4
                  # ajout de gini_gr avec le poids relatif
                  gini += gini_gr * taille / n_donnees # instruction 5
      return gini

## Q13 : fonction feuille
def feuille(data):
      nb0,nb1=0,0
      for i in range(len(data)):
            if data[i][-1]==0:
                  nb0+=1
            else:
                  nb1+=1
      if nb0>nb1:
            return 0
      else:
            return 1

## Q14 :
def construit ( arbre , sep_max , taille_min , p_var , ind_rec ) :
      gauche , droite = arbre[2] , arbre[3]
      if gauche==[] or droite==[] : # condition 1
            valeur = feuille( gauche + droite )
            arbre[2] = valeur
            arbre[3] = valeur
            return # On ne retourne aucune valeur car arbre est modifié directement
dans la fonction

      if ind_rec==sep_max : # condition 2
            arbre[2] , arbre[3] = feuille( gauche ) , feuille( droite )
            return # ajouté pour ne pas traiter les cas suivants.

      if len(gauche)< taille_min: # condition 3
            arbre[2] = feuille( gauche )
      else :
            arbre[2] = separe( gauche , p_var )
            construit( arbre[2] , sep_max , taille_min , p_var , ind_rec +1)

      if len(droite)< taille_min: # condition 4
            arbre[3] = feuille ( droite )
      else :
            arbre[3] = separe( droite , p_var )
            construit( arbre[3] , sep_max , taille_min , p_var , ind_rec +1)
```

Q15 :

Le pourcentage de réussite ne dépasse pas 88% avec pourtant le maximum de données utilisés (150 données). Le temps de calcul double alors que le nombre de données est multiplié par 1,5. Les pourcentages de réussite semblent sensibles au jeu de données utilisé (comparaison des 3 tests de prédiction).

Q16 :

Il s'agit de parcourir un arbre selon la logique mise en œuvre en Q9.

```
def prediction(arbre,donnee):
      [ind,val,gauche,droite]=arbre
      if donnee[ind]<val:
            if isinstance(gauche,list):
                  return prediction(gauche,donnee) #instruction1
            else:
                  return gauche #instruction2
      else:
            if isinstance(droite,list):
                  return prediction(droite,donnee) #instruction3
            else:
                  return droite #instruction4
```

Q17 :

```
def random_forest(data_train,data_test,sep_max,taille_min,n_arbres,p_var)
      foret=construire_foret(data_train,  sep_max,  taille_min,  p_var,  ind_rec,
n_arbres)
      classes=[]
      for donnee in data_test:
          n = 0
          for arbre in foret:
              n = n + prediction(arbre,donnees)
          if n <= (n_arbres//2) :
              classes.append(0)
          else:
              classes.append(1)
      return classes
```