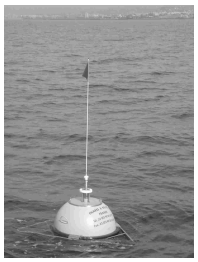
DS 06



Mesures de houle

Question de cours – Résolution d'équations différentielles

On cherche à résoudre l'équation différentielle $y' = y$ sur l'intervalle $[0,4]$ avec $y(0) = 1$.

Question 1 *Écrire la fonction de Cauchy* `f1(y:float,t:float)-> float` *associée à cette équation différentielle.*

Correction

```
def f(x,y):
    return y
```

Question 2 *Écrire la fonction* `euler(a:float,b:float,y0:float,h:float,f:function)-> list,list` *permettant de résoudre cette équation différentielle.*

Question 3 *Après avoir importé les bibliothèques nécessaires, donner les instructions permettant de tracer la solution de l'équation différentielle en fonction du temps. On prendra* $h = 0,001$.

Correction

```
def euler(a,b,y0,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y=y+h*f(x,y)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y
```

On cherche à résoudre l'équation différentielle $y'' + y = 0$ sur l'intervalle $[0,10]$ avec $y(0) = 0$ et $y'(0) = 1$.

Question 4 *Écrire la fonction de Cauchy* `f2(y:np.array,t:float)-> np.array` *associée à cette équation différentielle.*

Correction

```
def f(x,y):
    return (y[1],-y[0])
```

Question 5 *La définition de votre fonction* `f2` *peut-elle être utilisée avec la fonction* `euler` *? Si non, expliquer pourquoi et proposer des modifications.*

1 Introduction

2 Stockage interne des données

Question 6 *On suppose que chaque caractère est codé sur 8 bits. En ne tenant pas compte de la première ligne, déterminer le nombre d'octets correspondant à 20 minutes d'enregistrement à la fréquence d'échantillonnage de* 2 Hz.

Correction 8 caractères par lignes sur 8 bits = 8 octets par mesures. 20 minutes à 2 Hz = 20*60*2 = 2400 mesures soit 19200 octets en 20 minutes.

Question 7 *En déduire le nombre approximatif (un ordre de grandeur suffira) d'octets contenus dans le fichier correspondant à la campagne de mesures définie précédemment. Une carte mémoire de* 1 Go *est-elle suffisante ?*

Correction En 15 jours : $15 \times 24 \times 2 \times 19200 = 13\,824\,000$ octets soit 13,8 Mo. La carte mémoire est largement suffisante.

Question 8 *Si, dans un souci de réduction de la taille du fichier, on souhaitait ôter un chiffre significatif dans les mesures, quel gain relatif d'espace mémoire obtiendrait-on ?*

Correction Un chiffre de moins donne 1 octet de moins par mesure soit 12,5 % de moins (soit 1,8 Mo de moins).

Question 9 *Les données se trouvent dans le répertoire de travail sous forme d'un fichier* `donnees.txt`. *Proposer une suite d'instructions permettant de créer à partir de ce fichier une liste de flottants* `liste_niveaux` *contenant les valeurs du niveau de la mer. On prendra garde à ne pas insérer dans la liste la première ligne du fichier.*

Correction

```
def liste_niveaux():
    fichier = open('donnees.txt','r')
    ligne = fichier.readline()
    liste_niveaux = []
    while ligne:
        ligne = fichier.readline()
        if ligne:
            liste_niveaux.append(float(ligne))
    fichier.close()
    return liste_niveaux
```

Deux analyses sont effectuées sur les mesures : l'une est appelée« vague par vague », l'autre est appelée « spectrale ».

3 Analyse « vague par vague »

Question 10 *Pour le signal représenté sur la figure suivante, que valent approximativement H_1, H_2 et H_3 ? Que valent approximativement T_1 et T_2 ?*

Correction $H_1 = 6-(-3) = 9\,\text{m}$, $H_2 = 6,9-(-2) = 8,9\,\text{m}$, $H_2 = 5-(-1,2) = 6,2\,\text{m}$.
$T_1 = 15,5-3,75 = 11,75\,\text{s}$ et $T_2 = 28,25-15,5 = 12,75\,\text{s}$.

Question 11 *Proposer une fonction* `moyenne` *prenant en argument une liste non vide* `liste_niveaux`, *et retournant sa valeur moyenne.*

Correction

```
def moyenne(liste_niveaux):
    somme = 0
    for niveau in liste_niveaux:
        somme += niveau
    return somme / len(liste_niveaux)
```

Question 12 *Proposer une fonction* `integrale_precise` *prenant en argument une liste non vide* `liste_niveaux`, *et retournant la valeur approchée de l'intégrale de η sur une période de 20 minutes. On demande d'utiliser la méthode*

des trapèzes (on supposera que l'échantillonnage est constant). En déduire une fonction `moyenne_precise` *prenant en argument une liste non vide* `liste_niveaux` *et retournant une estimation de la moyenne de* η *sur une période de 20 minutes.*

Correction

```
def integrale_precise(liste_niveaux):
    integrale = 0
    for i in range(20 * 60* 2  - 1):
        trapeze = (liste_niveaux[i] + liste_niveaux[i+1] ) / 2 * 0.5
        integrale += trapeze
    return integrale

def moyenne_precise(liste_niveaux):
    return integrale_precise(liste_niveaux) / (20 * 60)
```

Question 13 *Proposer une fonction* `ind_premier_pzd(liste_niveaux:list) -> int` *retournant, s'il existe, l'indice du premier élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner -1 si aucun élément vérifiant cette condition n'existe.*

Correction

```
def ind_premier_pzd(liste_niveaux):
    i = 0
    moy = moyenne(liste_niveaux)
    while not(liste_niveaux[i] > moy and \
              liste_niveaux[i+1] < moy) and \
              i < len(liste_niveaux):
        i += 1
    if liste_niveaux[i] > moy and liste_niveaux[i+1]:
        return i
    else:
        return -1
```

Question 14 *Proposer une fonction* `ind_dernier_pzd(liste_niveaux:list) -> int` *retournant l'indice i du dernier élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner -2 si aucun élément vérifiant cette condition n'existe. On cherchera à proposer une fonction de complexité* $O(1)$ *dans le meilleur des cas.*

Correction

```
def ind_dernier_pzd(liste_niveaux):
    i = len(liste_niveaux) - 2
    moy = moyenne(liste_niveaux)
    while not(liste_niveaux[i] > moy and liste_niveaux[i+1] < moy) and i >0:
        i -= 1
    if liste_niveaux[i] > moy and liste_niveaux[i+1]:
        return i
    else:
        return -2
```

Question 15 *On propose la fonction* `construction_successeurs`. *Elle retourne la liste* `successeurs`. *Compléter (sur la copie) les lignes à compléter.*

Correction

```
def construction_successeurs(liste_niveaux):
    n = len(liste_niveaux)
    successeurs = []
    m = moyenne(liste_niveaux) # à la place de moyenne précise pour pouvoit
       traiter de plus petites listes
    for i in range(n-1):
        if i+1 > ind_premier_pzd(liste_niveaux): # A completer
```

```
            successeurs.append(i+1) # A completer
    return successeurs
```

Question 16 *Donner la complexité algorithmique de la fonction* `construction_successeurs` *dans le pire des cas.*

Question 17 *Proposer une fonction* `decompose_vagues(liste_niveaux)` *qui permet de décomposer une liste de niveaux en liste de vagues. On omettra les données précédant le premier PND et celles succédant au dernier PND. Ainsi* `decompose_vagues([1,-1,-2,2,-2,-1,6,4,-2,-5])` *(noter que cette liste est de moyenne nulle) retournera* `[[-1,-2,2], [-2,-1,6,4]]`.

Correction

```
def decompose_vagues(liste_niveaux):
    m = moyenne(liste_niveaux)
    i_debut = ind_premier_pzd(liste_niveaux)
    i_fin = ind_dernier_pzd(liste_niveaux)
    vagues = []
    i = i_debut + 1
    while i < i_fin:
        i_vague = ind_premier_pzd(liste_niveaux[i:])
        vagues.append(liste_niveaux[i: i + i_vague + 1])
        i += i_vague + 1
    return vagues
```

On désire maintenant caractériser les vagues. Ainsi, on cherche à concevoir une fonction `proprietes(liste_niveaux)` retournant une liste de listes à deux éléments `Hi` , `Ti` permettant de caractériser chacune des vagues `i` par ses attributs :

- `Hi`, sa hauteur en mètres (m) (voir Figure précédente),
- `Ti`, sa période en secondes (s).

Question 18 *Proposer une fonction* `proprietes(liste_niveaux)` *réalisant cet objectif. On pourra utiliser les fonctions de Python* `max(L)` *et* `min(L)` *qui retournent le maximum et le minimum d'une liste* `L`*, respectivement.*

Correction

```
def proprietes(liste_niveaux):
    vagues = decompose_vagues(liste_niveaux)
    liste = []
    for i in range(len(vagues)-1):
        Hi=max(vagues[i])-min(vagues[i+1])
        Ti=len(vagues[i]) * 0.5 # fréquence de 2 Hz entre chaque mesure
        liste.append([Hi, Ti])
    return liste
```