DS4 d'informatique – corrigé

Partie I. Stockage interne des données

1. Chaque ligne est composée de 8 caractères de 8 bits chacun, soit 64 bits de données par ligne du fichier. Pour un enregistrement de 20 minute, soit 1200 secondes, on aura besoin de 153600 bits de données soit environ 153 kilobits ou environ 20 kilooctets.

   La campagne de mesures dure 15 jours avec 20 minutes d'enregistrement toutes les demi-heures, soit 48 enregistrements de 20 minutes par jour, soit 720 enregistrements de 20 minutes. Ce qui correspond à environ 14 400 ko, soit environ 15 Mo. La carte mémoire de 1 Go est très largement suffisante !

   Si on ôte un chiffre significatif, on supprime 1/8ème des bits, soit 12,5 % des données. On aura donc un gain relatif de 12,5 % d'espace mémoire. Vu la question précédente, c'est parfaitement inutile.

2. On peut utiliser la suite d'instructions suivante :

```
fich = open("donnees.txt")
fich.readline()
liste_niveaux = []
for l in fich:
  liste_niveaux.append(float(l))
```

Partie II. Analyse « vague par vague »

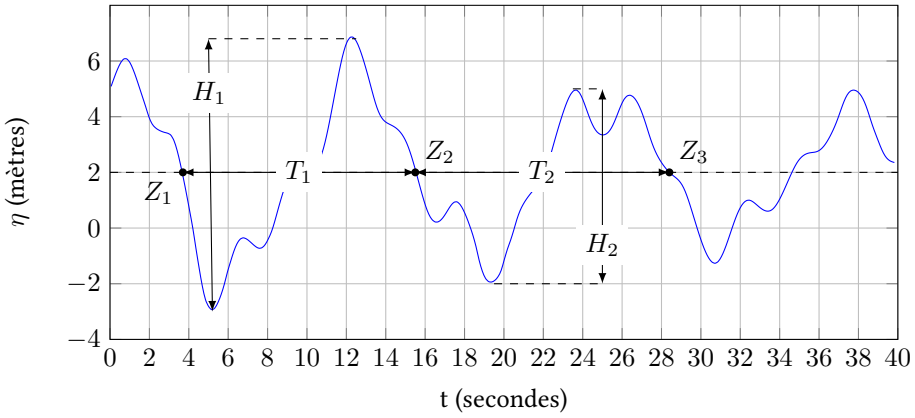

FIGURE 1 – Passage par le Niveau moyen en Descente (PND). Ici la moyenne m vaut 2 m.

3. Sur le signal représenté, on a $H_1 \approx 10\,\text{m}$, $H_2 \approx 7\,\text{m}$, $T_1 \approx 12\,\text{s}$ et $T_2 \approx 12\,\text{s}$.

4.

```
def moyenne(liste_niveaux):
  S=0
  for n in liste_niveaux:
    S+=n
  return S/len(liste_niveaux)
```

5.

```
def integrale(liste_niveaux):
  S=0    # Pour stocker la somme
  h=0.5  # temps qui sépare deux points
  for i in range(len(liste_niveaux)):
    S += h*(liste_niveaux[i]+liste_niveaux[i+1])/2
  return S
```

6.

```
def moyenne_precise(liste_niveaux):
  return integrale(liste_niveaux)/1200 #20 min = 1200 sec
```

7.

```
def ind_premier_pnd(liste_niveaux):
  m = moyenne_precise(liste_niveaux)
  for i in range(0, len(liste_niveaux)-1):
    if liste_niveaux[i]>m and liste_niveaux[i+1]<m:
      return i
  return -1
```

8.

```
def ind_dernier_pnd(liste_niveaux):
  m = moyenne_precise(liste_niveaux)
  ind = -2
  for i in range(0, len(liste_niveaux)-1):
    if liste_niveaux[i]>m and liste_niveaux[i+1]<m:
      ind = i
  return ind
```

9. Dans le meilleur des cas, cette fonction est de complexité $O(n)$ car il faut parcourir toute la liste. Pour écrire une fonction avec une complexité $O(1)$ dans le meilleur des cas, on pourrait parcourir la liste en partant de la fin, par exemple :

```
def ind_dernier_pnd(liste_niveaux):
  m = moyenne_precise(liste_niveaux)
  for i in range(len(liste_niveaux-1), 0, -1):
    if liste_niveaux[i]<m and liste_niveaux[i-1]>m:
      return i-1
  return -2
```

10.

```
def construction_successeurs(liste_niveaux):
    n = len(liste_niveaux)
    successeurs = []
    m = moyenne(liste_niveaux)
    for i in range(n-1):
        if liste_niveaux[i]>m and liste_niveaux[i+1]<m:
          successeurs.append(i+1)
    return successeurs
```

11.

```
def decompose_vagues(liste_niveaux):
  successeurs = construction_successeurs(liste_niveaux)
  vagues = []
  for i in range(len(successeurs)-1):
     vagues.append(liste_niveaux[successeurs[i]:successeurs[i+1])
  return vagues
```

12.

```
def plusGrand(liste):
  pg = liste[0]
  for e in liste:
    if e > pg:
      pg = e
  return pg

def plusPetit(liste):
  pp = liste[0]
  for e in liste:
    if e < pp:
      pp = e
```

```
  return pp
```

13.

```
def proprietes(liste_niveaux):
  vagues = decompose_vagues(liste_niveaux)
  prop = []
  for v in vagues:
    H = plusGrand(v) - plusPetit(v)
    T = len(v)*0.5                  #0.5 s par point
    prop.append([H,T])
  return prop
```

Partie III. Contrôle des données

14.

```
def hauteur_max(liste_niveaux):
  prop = proprietes(liste_niveaux)
  Hmax = prop[0][0]  # Hauteur de la première vague.
  for p in prop:
    if p[0] > Hmax:
      Hmax = p[0]
  return Hmax
```

15.

```
def hauteur_13(liste_niveaux):
  # liste des propriétés triées par hauteur de vague
  prop = tri(proprietes(liste_niveaux))
  # Le tiers supérieur des plus grandes vagues observées.
  prop13 = prop[0:len(prop)//3]
  # Il ne reste plus qu'à calculer la moyenne des hauteurs des vagues de la liste prop13
  S = 0
  for p in prop13:
    S += p[0]
  return S/len(prop13)
```

16. On reprend en grande partie la fonction précédente :

```
def periode_13(liste_niveaux):
  # liste des propriétés triées par hauteur de vague
  prop = tri(proprietes(liste_niveaux))
  # Le tiers supérieur des plus grandes vagues observées.
  prop13 = prop[0:len(prop)//3]
  # Il ne reste plus qu'à calculer la moyenne des périodes des vagues de la liste prop13
  S = 0
  for p in prop13:
    S += p[1]
  return S/len(prop13)
```

17. Le problème est que la moyenne est recalculée à chaque itération de la boucle `for`. On modifie la fonction pour calculer la moyenne une seule fois avant la boucle.

```
def skewness(liste_hauteurs):
    n = len(liste_hauteurs)
    et3 = (ecartType(liste_hauteurs))**3
    m = moyenne(liste_hauteurs)
    S = 0
    for i in range(n):
        S += (liste_hauteurs[i] - m)**3
    S = n/(n-1)/(n-2) * S/et3
    return S
```

18. Les complexités des deux fonctions devraient être du même type car pour les deux grandeurs il faut calculer une somme de n éléments. On s'attend donc à une complexité en $O(n)$ dans les deux cas.