Sujet E3A PC Physique Modélisation : Troisième partie

F / BASE DE DONNEES

F1

```
'''PRIMARY KEY ('id')'''
'''PRIMARY KEY ('idStation', 'date', 'heure')'''
```

F2

```
n_lignes_station = 63
n_lignes_hauteurs = 63 * 365 * 24 * 6
```

La seconde table comporte donc 3 311 280 lignes, ce qui commence à faire beaucoup pour une variable stockée dans la mémoire vive d'un ordinateur.

F3

```
'''SELECT latitude, longitude FROM station WHERE nom = 'SAINT-MALO' ;'''
'''SELECT nom FROM station WHERE longitude < 0 ;'''
'''SELECT hauteur FROM hauteurs WHERE idStation in (SELECT id FROM station
WHERE nom = 'BREST') AND date = '06/04/2013' AND heure = '14 :00 :00' ;'''
```

G / REPRESENTATIONS GRAPHIQUES

G1

`data[2][1]` renvoie `'00 :20 :00'`.

G2

```
height = [ligne[2] for ligne in data]
```

G3

```
h_jan13 = height[12 * 24 * 6  : 13 * 24 * 6]
h_fev = height[31 * 24 * 6  : (31 + 28) * 24 * 6]
```

Un code amélioré de ce type était aussi possible :

```
h_jan13bis = []
for mesure in mar.data  :
    if mesure [0] == '13/01/2013'  :
        h_jan13bis.append(mesure[2])

h_fevbis = []
for mesure in mar.data  :
    if mesure [0][3  : 5] == '02'  :
        h_fevbis.append(mesure[2])
```

G4

```
dates1 = [float(k / 6) for k in range(len(h_jan13))]
dates2 = [float( k / (24 * 6)) for k in range(len(h_fev))]
```

Le code minimum pour obtenir les deux tracés demandés est le suivant.

```
plt.plot(dates1, h_jan13)
plt.show()

plt.plot(dates2, h_fev)
plt.show()
```

Pour obtenir le tracé de la Figure 6, c'est le suivant.

```
plt.figure(6, figsize = (12, 6))

plt.subplot(1, 2, 1)
plt.plot(dates1, h_jan13)
plt.title('hauteurs de marée à Brest le 13 janvier 2013')
x_pas = 5  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates1[k] for k in range(0, len(dates1), 6 * x_pas)]
x_labels = list(range(0, int(len(dates1) / 6), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates1[0], dates1[-1]])
plt.xlabel('heure')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.subplot(1, 2, 2)
plt.plot(dates2, h_fev)
plt.title('hauteurs de marée à Brest durant le mois de février 2013')
x_pas = 5  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates2[k] for k in range(0, len(dates2), 24 * 6 * x_pas)]
x_labels = list(range(0, int(len(dates2) / (24 * 6)), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates2[0], dates2[-1]])
plt.xlabel('jour dans le mois')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.savefig('figure_6.png')
plt.show()
```

Voici alors la figure obtenue.

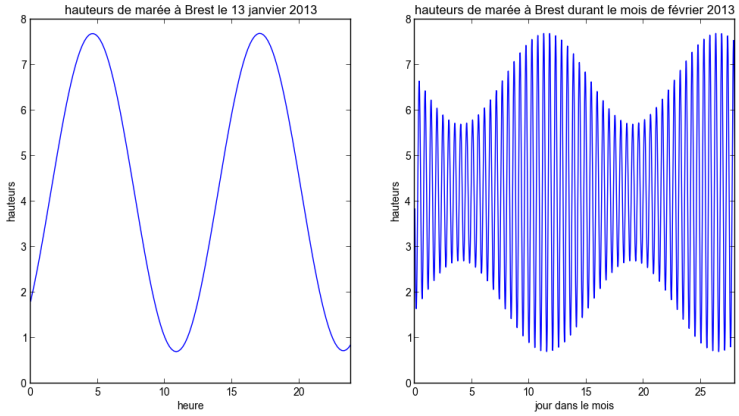

FIGURE 1 – Figure 6 obtenue

Remarque 1. *Pour obtenir ces courbes, j'ai simulé par un produit de sinusoïdes les hauteurs d'eau de marée. C'est pourquoi les courbes obtenues sont très régulières, mais cela permet de valider les codes proposés.*

H / PERIODE ET AMPLITUDE DES MAREES

H1

```
def minimum(L) :
    '''renvoie la plus petite valeur contenue dans la liste L, qui est une
liste de nombres non vide.'''
    res = L[0]
    for k in range(1, len(L)) :
        if res > L[k]  :
            res = L[k]
    return res
```

La variable intermédiaire `res` de la fonction `minimum([5, 6, 3, 4, 3, 8])` prendra d'abord la valeur 5 puis la boucle `for` va opérer 5 fois. À la deuxième itération, `res` prendra la valeur 3 et la gardera jusqu'à la fin de la boucle. La fonction renverra donc 3.

H2

```
def min2(L) :
    '''renvoie l'indice de la plus petite valeur contenue dans la liste L,
qui est une liste de nombres non vide.
La fonction renverra l'indice de la première occurrence de cette valeur.'''
    ind = 0
    for k in range(1, len(L)) :
        if L[ind] > L[k]  :
            ind = k
    return ind
```

Pour obtenir le jour et l'heure de la plus petite hauteur d'eau à Brest lors de l'année 2013, il suffit d'écrire `jour, heure = tuple(data[min2(height)][ :2])`.

H3

```
def moyenne(L) :
    '''renvoie la moyenne des éléments de la liste L qui est une liste de
nombres non vide.'''
    somme, n = 0, len(L)
    for i in range(n) :
        somme += L[i]
    return ( somme / n)
```

H4 Voici son code commenté.

```
def fonction(liste)  :

    m = moyenne(liste)  # m est la moyenne des hauteurs d'eau
    e = (m-minimum(liste))/10
# [m - e, m + e] est donc un intervalle de fluctuation centré
# sur la hauteur d'eau moyenne

    val = []  # initialisation d'une liste vide
    i = 0
    while i < len(liste)  :
# Cette boucle while aurait pu être remplaçée par une boucle for
# pour simplement parcourir la liste.
        if liste[i] > m-e and liste[i] < m+e  :
# m - e et m + e définit une bande autour de la moyenne
            a = i
# a mémorise donc l'indice d'entrée dans la bande moyenne
            while liste[i] > m-e and liste[i] < m+e  :
                i += 1
```

```
# la liste est parcourue tant que la hauteur est dans la
# bande moyenne
            b = i-1
# b mémorise donc l'indice de sortie de la bande moyenne
            val.append((a+b)/2)
# On ajoute l'indice moyen entre deux entrée-sortie dans la
# bande moyenne (ce qui est une estimation de l'indice où la
# hauteur d'eau est égale à la hauteur moyenne).
        i += 1

    s = 0
    n = len(val)
# n correspond donc au nombre de fois que la hauteur d'eau passe par
# la hauteur d'eau moyenne
    for i in range(n-1)  :
        s += val[i+1]-val[i]
# s est la somme télescopique des écarts d'indice entre deux passages
# successifs de la marée à la hauteur d'eau moyenne, donc une image en
# indice de la demi-période de la marée
# C'est donc directement val[-1] - val[0], ce qui ne nécessite pas de
# boucle

    return 2*s/(n-1)
# renvoie donc une estimation d'une image en indice de la période moyenne
# des marées
```

H5 L'utilisation des fonctions précédentes pour construire cette dernière fonction n'est pas aussi limpide qu'espéré. L'estimation de la période moyenne des marées en indice qui exige donc de l'arrondir pour être utilisé crée un biais qui se cumule et rend l'exploitation de cette fonction caduque.

Voici un code que je propose comme solution.

```
def marnages(Liste) :
    '''renvoie la liste des marnages qui sont la différence de hauteurs
entre une pleine mer et une basse mer successives à partir d'une liste des
hauteurs de marées.'''
    iT = int(fonction(Liste))   # la fonction de la question H4 qui renvoie
                                # la période moyenne des marées en indices
    ihm = min2(Liste[ : iT]) # indice de la première marée basse
    M = []                  # initialisation de la liste des marnages

    if ihm < iT // 2 :       # Si Liste démarre par une marée basse
        ihm = iT + min2(Liste[iT  : 2 * iT])  # recherche la seconde

    iacc = ihm            # indice accumulé

    while iacc + iT < len(Liste) :   # parcours de toute la liste

        ihM = ihm - iT // 2     # estimation de l'indice de la pleine mer
                                # précédent la basse mer trouvée par ihm
        M.append(Liste[ihM] - Liste[ihm])   # le marnage est ajouté
        ihm = iacc + 5 * iT // 6 + min2(Liste[ihm + 5 * iT // 6  : ihm + iT ])
                            # recherche de l'indice de la marée basse suivante
        iacc = ihm

    return(M)

h_1trim = height[ : 90 * 24 * 6]
Ampl = marnages(height)
dates3 = [float( k / (24 * 6)) for k in range(len(h_1trim))]
dates4 = list(range(len(Ampl)))     # ne correspondront pas aux jours vraiment
```

```
plt.figure(7, figsize = (12, 6))

plt.subplot(1, 2, 1)
plt.plot(dates4, Ampl)
plt.title('amplitudes des marées à Brest durant le premier trimestre 2013')
x_pas = 10  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates4[k] for k in range(0, len(dates4), 6 * x_pas)]
x_labels = list(range(0, int(len(dates4) / 6), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates4[0], dates4[-1]])
plt.xlabel('nombre de marnages')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.subplot(1, 2, 2)
plt.plot(dates3, h_1trim)
plt.title('hauteurs de marée à Brest durant le premier trimestre 2013')
x_pas = 10  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates3[k] for k in range(0, len(dates3), 24 * 6 * x_pas)]
x_labels = list(range(0, int(len(dates3) / (24 * 6)), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates3[0], dates3[-1]])
plt.xlabel('jour dans l\'année')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.savefig('figure_7.png')
plt.show()
```

Mais voici alors la figure obtenue.

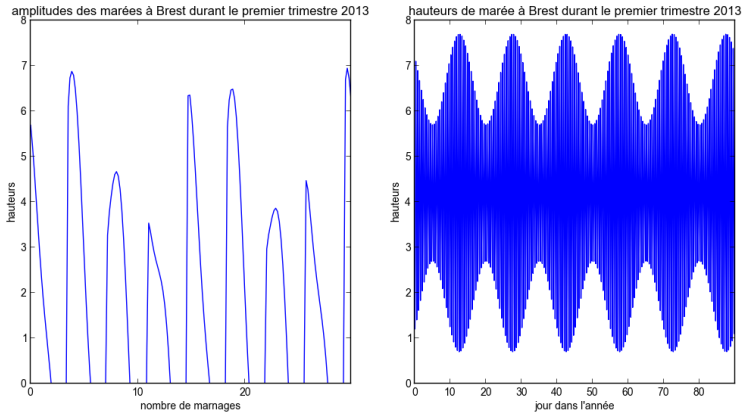

FIGURE 2 – Figure 7 obtenue

Voici un code que je propose comme solution qui utilise une fonction `extrema_locaux` qui recherche les extrema locaux d'un nuage de points.

```
def extrema_locaux(L, m = 5) :
    '''renvoie une liste d'indices correspondant à des extrema locaux pour L,
vus sur m valeurs locales. L est une liste de nombres non vide et m un entier'''
    extrema = []
    n = len(L)
    pente = (moyenne(L[1  : 1 + m]) - moyenne(L[ : m])) > 0
    for i in range(1, n - m) :
        var = (moyenne(L[i  : i + m]) - moyenne(L[i - 1  : i - 1 + m])) > 0
        if var  != pente  :
            extrema.append(i + (m - 1) // 2 - 1)
            pente = var
    return( extrema )

def amplitudes(L, Extr) :
    '''renvoie une liste d'amplitudes calculées entre chaque valeur prise par
L aux indices indiqués dans Extr.'''
    Amplit = []
    for i in range(len(Extr) - 1) :
        Amplit.append(abs(L[Extr[i + 1]] - L[Extr[i]]))
    return( Amplit )

Extrema = extrema_locaux(h_1trim) # Rappel  : h_1trim = height[ : 90 * 24 * 6]
Ampl = amplitudes(h_1trim, Extrema)
dates5 = [float( k / (24 * 6)) for k in range(len(h_1trim))]
dates6 = [float( k / (24 * 6)) for k in Extrema]

plt.figure(7, figsize = (12, 6))

plt.subplot(1, 2, 1)
plt.plot(dates6, [None] + Ampl)
plt.title('amplitudes des marées à Brest durant le premier trimestre 2013')
x_pas = 10  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates6[k] for k in range(0, len(dates6), 6 * x_pas)]
x_labels = list(range(0, int(len(dates6) / 6), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates6[0], dates6[-1]])
plt.xlabel('jour dans l\'année')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.subplot(1, 2, 2)
plt.plot(dates5, h_1trim)
plt.title('hauteurs de marée à Brest durant le premier trimestre 2013')
x_pas = 10  # Pas des valeurs indiquées en abscisse
x_valeurs = [dates5[k] for k in range(0, len(dates5), 24 * 6 * x_pas)]
x_labels = list(range(0, int(len(dates5) / (24 * 6)), x_pas))
plt.xticks(x_valeurs, x_labels)
plt.xlim([dates5[0], dates5[-1]])
plt.xlabel('jour dans l\'année')
y_valeurs = list(range(9))
plt.yticks(y_valeurs, y_valeurs)
plt.ylim([y_valeurs[0], y_valeurs[-1]])
plt.ylabel('hauteurs', rotation ='vertical')

plt.savefig('figure_7bis.png')
plt.show()
```

Voici alors la figure obtenue.

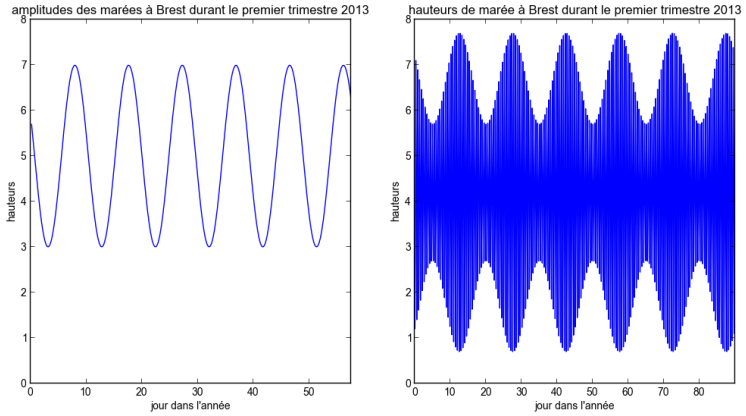

FIGURE 3 – Nouvelle Figure 7 obtenue

C'est déjà plus satisfaisant.