Lundi 30 novembre 2020 **Rendu**

Note d'avancement :

Projet d'Algorithmique Programmation

JIANG Jacques

LLAGONNE Nathan

I. Prise en main du sujet et des premiers outils

Dès la prise de connaissance du sujet, notre objectif a été d'appréhender l'outil numérique git avec l'utilisation de Github et Git Bash. C'est donc à travers des tutoriels que nous avons appris à l'utiliser. Nous nous sommes alors entraîné sur un *repository* factice que nous avions créé. Il est à relever que cette prise en main n'a pas été très facile. Et même lorsque nous avons travaillé sur le *repository* du projet, il nous est arrivé au début de faire plusieurs coquilles. Mais au final, notre utilisation s'est avérée être plutôt efficace et productive.
Voici le lien url de notre *repository* : https://github.com/LLAGONNE/Algo-Info-projet

D'un autre côté, il a fallu séparer les différentes tâches de travail. Nathan s'est alors orienté vers la conception des différents programmes tandis que Jacques s'est dirigé vers les programmes qui consistaient à afficher les courbes. C'est également lui qui s'est occupé de la dernière partie qui consiste à rendre accessible notre travail dans la ligne de commande.

II. Découverte du module Pandas et des attentes du sujet

Tout d'abord, ce module qui nous permet de manipuler un fichier CSV nous était inconnu à tous les deux. Il a fallu apprendre à s'en servir et c'est ce que nous avons fait dans un premier temps.

```
import pandas as pd
```

Il s'est alors posé un premier problème. En effet, nous n'avions pas indiqué préalablement dans la lecture du ficher les séparateurs. La lecture affichait donc une unique colonne. Nous avons donc ajouté la fonctionnalité *sep*. On a ensuite tenu à trier le fichier par capteur comme ci-dessous.

```
KM = pd.read_csv ('EIVP_KMbis.csv' ,sep = ';')
KMb = KM.sort_values (by = 'id')
```

Par la suite, nous avons tenu à utiliser le moins de fonctions d'un module possible. Et dans un souci de rendre notre travail le plus modulable possible et capable de traiter la plus grande diversité de fichiers. Les arguments des fonctions sont certes long et pénible à rentrer. Mais cela est compensé par une optimisation dans la console.

III. Programme principal

Dans un premier temps, la création de deux fonctions nous a paru nécessaire : c'est-à-dire *count_time* et *Temps*. Ces deux fonctions ont pour objectifs respectivement de donner les indices de changement de capteur et de créer une liste des temps pour les différents affichages graphiques. Cependant, cette idée de créer deux fonctions séparément du reste ne nous est pas venue dans un premier temps, mais plus tard lorsqu'on s'est aperçu qu'elles

allaient nous être nécessaire plusieurs fois. Il est important de préciser que dans la suite la fonction *count_time* correspondra à *f1* et *Temps* à *f2*.
Par exemple si l'on exécute la première fonction il sort deux listes.

```
In [55]: count_time (KMb, 'id', 'sent_at')
Out[55]: ([0, 1336, 2681, 4026, 5370, 6535], [1335, 2680, 4025, 5369, 6534, 7879])
```

La première donne le premier indice d'un nouveau capteur, et la deuxième donne le dernier indice où apparaît un capteur
La fonction *Temps*, quant à elle, renvoie la liste des temps découpée par capteur.
Ces deux fonctions ont une complexité linéaire (O(n)).

Avec ceci réalisé, on a alors pu commencer à répondre au premier point demandé : qui consiste à représenter les courbes de valeurs en fonction du temps. On s'est alors servi des deux fonctions précédentes. Ce point ne nous a pas posé de problèmes. Et par précision, *f3* situé en argument de la fonction courbe1 est *point1*. Sur ce point, aucune difficulté n'a été relevée et une réponse a été formule naturellement, l'utilisation de *matplotlib.pyplot* étant connue de nous deux.

Dans un second temps, il nous était demandé de calculer les minimum, maximum, moyennes… Pour répondre aux attentes nous avons décidé de construire une unique fonction qui calcule de manière identique les minimums, maximum, écart-types, variances, médianes de chaque colonne (bruit, température…). Et qui suivant la colonne, détermine la moyenne adaptée. En effet, l'intensité sonore étant en dBA, la moyenne logarithmique est la plus adaptée, de même que pour l'humidité qui s'exprime en pourcentages, on utilise la moyenne géométrique. Pour le reste, on a choisi de se servir de la moyenne arithmétique.
Un problème est alors survenu. Pour déterminer la médiane, il a fallu trier la liste que l'on avait. Or, il n'est pas possible de trier comme nous le souhaitions, sûrement à cause du format DataFrame de la bibliothèque Pandas. Pour remédier au problème, il a été décidé de transférer les données dans une nouvelle liste, et seulement après, nous avons trié grâce au tri par insertion. Ce choix a été pris par commodité (le tri fusion aurait été le plus adéquat), il n'était effectivement pas aisé de joindre le tri au découpage par capteur. Le tri fusion étant de complexité quadratique ($O(n^2)$) et le tri fusion de complexité quasi-linéaire (O(nlog(n)).

```
    #calcul de la médiane
    #on va d'abord trier cette liste avec le tri par insertion par exemple
    L = []
    med = []

    for j in range (len (f1[0])) :
        L.append ([])
        for k in range (f1[0][j], f1[1][j]) :
            L[j].append (KMb[colonne][k])
        for i in range (1, len (L[j])) :
            x = L[j][i]
            m = i
            while m > 0 and x < L[j][m - 1] :
                L[j][m] = L[j][m - 1]
                m = m - 1
            L[j][m] = x
        if len (L[j]) % 2 == 0 :
            med.append ( (L[j][len (L[j]) // 2] + L[j][len (L[j]) // 2 + 1]) / 2)
        else :
            med.append (L[j][len (L[j]) // 2 + 1])
```

Pour cette fonction *point2,* chaque liste demandée (liste des min, des max…) est découpée par capteur sur le type de valeur demandé (luminosité, température…). De plus, il ne nous vient pas de manière plus efficace de traiter ce point-ci sans utiliser directement les fonctions directement implémentées dans les différents modules. En sortie de fonction, notre choix s'est orienté vers une liste de liste des différentes valeurs demandées.

```
        return min_, max_, moygeo, ect, V, med, ['min_', 'max_', 'moygeo', 'ect', 'V', 'médiane']
```

La plupart des « sous-fonctions » de cette fonction sont de complexité quadratique.

Ensuite, il était demandé de donner l'indice humidex. Nous nous sommes alors renseigné, et nous avons décidé d'utiliser le tableau présent sur la page wikipédia. Notre démarche a alors été de mettre ce tableau sous format CSV, le lire puis déterminer à partir de celui-ci la valeur de l'indice. Pour cela, il a été nécessaire d'arrondir chaque valeur à une valeur présente dans le tableau. On a alors décidé de choisir d'arrondir à la valeur présente la plus proche (on aurait également pu choisir la valeur inférieure qui convient). *Hr1* (humidité relative donnée par les capteurs) devient *Hr2* pour pouvoir utiliser le tableau. De même pour la température T2.
L'étape suivante consiste à trouver la valeur de l'indice associée à une humidité relative et à une température donnée.

```
    for k in range (len (T2)) : #On rappele que T2 et Hr2 sont de même longueur
        i = str (int (T2[k])) #on veut ici le nom de la colonne que l'on va chercher dans le tableau humidex
                              #certaines valeurs sont également des flottants
        j = 0 #on va chercher l'indice de la ligne associée
        while THx.loc[j][0] != int (Hr2[k]) : #car les valeurs de cette ligne sont des flottants
            j += 1
        Hx.append (int (THx[i][j]))
    print (Hx)
```

Il y a dans ces quelques lignes des subtilités auxquelles il a fallu faire face. Pour trouver la bonne colonne, il fallait trouver le nom de la colonne tandis que pour déterminer la bonne ligne, il fallait l'indice de la ligne. Mais aussi, il fallait bien faire attention que les valeurs voulues étaient des entiers ou des flottants pour qu'elles soient reconnues. Nous avons décidé, en fonction des consignes de renvoyer un unique liste (qui regroupe tous les capteurs). L'absence de nécessité d'affichage de courbe, nous permet de ne pas séparer cette liste et de la laisser « brute ».

Le dernier exercice demandé n'a pas particulièrement posé de problème. Nous avons utilisé la formule pour calculer un coefficient de corrélation et nous l'avons retranscrite. Nous avions besoin des moyennes, on a alors utilisé le fonction *point2* qui nous les données. Nous sommes conscients que d'agir ainsi ralenti considérablement la vitesse d'exécution car on calcule d'autres valeurs qui sont déjà lourdes à déterminer. Nous avons tout de même choisi d'agir ainsi. C'est une fonction de complexité quadratique.

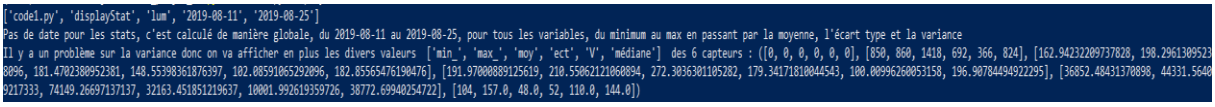
IV. Sujet 1 – Groupe C

Dans cette partie, nous devions nous préoccuper de trouver les différentes anomalies qui pouvaient être présentes. C'est dans cette partie que nous avons rencontré le plus de problèmes. Il a d'abord fallu se décider sur la manière de repérer les possibles anomalies. Nous avons donc fait le choix de détecter les anomalies grâce à un intervalle de confiance. Cela

consiste à calculer une médiane locale et un écart-type local. Puis de garder toute valeur comprise dans cet intervalle et de ressortir l'indice d'une anomalie détecté.
Dans deux fonctions annexes, on calcule la médiane et l'écart-type en local. Puis dans la fonction principale on vérifie que chaque valeur appartienne à cet intervalle.
Cependant, nous n'avons pas réussi à faire une algorithmique faible en complexité. Nous n'avons pas réussi à faire un programme concluant, le temps de calcul est beaucoup trop long (supérieur à 10min). Nous n'avons donc pas les résultats. Nous avons cependant testé sur les premières valeurs et cela marchait. On peut noter une complexité supérieure ou égale à $O(n^4)$.
Nous avons essayé de conjecturer ce que l'affichage des courbes d'anomalie donnerait.

V. Powershell

Pour rendre le code utilisable sur PowerShell, nous avons utilisé la fonction *argv* de la bibliothèque *sys*. Cette fonction est adaptée à l'utilisation de PowerShell. Quand on lance le terminal, on peut rentrer des arguments qui seront interprétés par Python sous forme de liste avec comme premier élément le nom du fichier. Comme on peut le voir sur la capture d'écran ci-dessous.



Pour utiliser *argv*, nous avons décidé d'établir une nouvelle fonction, appelée *execution*, qui a pour but de vérifier les arguments entrés dans la liste *sys.argv*, par des tests d'égalités. Qui se présentent sous la forme :

```
    if a[1]== "help" or a[1]=="man":
        return "Regardez le PDF du sujet "

    elif a[1]=="display":

        if a[2]=="humidex":

            return point3(KMb, 'id', 'sent_at', count_time(KMb,'id','sent_at'))
```

Avec *a* qui correspond à la liste générée par la fonction *sys.argv*.

Cette fonction permet également de prendre en compte par ces mêmes tests d'égalités, les éventuelles erreurs que les différentes personnes pourraient effectuer. Ainsi, on peut exécuter le programme sans même avoir à lancer un interpréteur Python (comme Pyzo, Jupyter, Spyder…), tout peut se faire depuis le terminal ou bien PowerShell. Par conséquent, les courbes obtenues sont celles de fonctions sans modification.

VI. Conclusion

Pour conclure, ce projet nous a permis de nous rendre compte de l'utilité et des capacités que pouvait fournir l'utilisation de Git dans un travail sur projet. Il nous a permis de nous familiariser avec cet outil. En ce qui concerne la programmation, cela a été l'occasion de

pousser nos compétences acquises dans notre cas en CPGE. Mais surtout, il nous a permis de découvrir la fonctionnalité en rapport avec PowerShell.
Cela constitue une bonne ouverture au projet du semestre prochain.

En espérant avoir correctement répondu à vos attentes.

JIANG Jacques
LLAGONNE Nathan