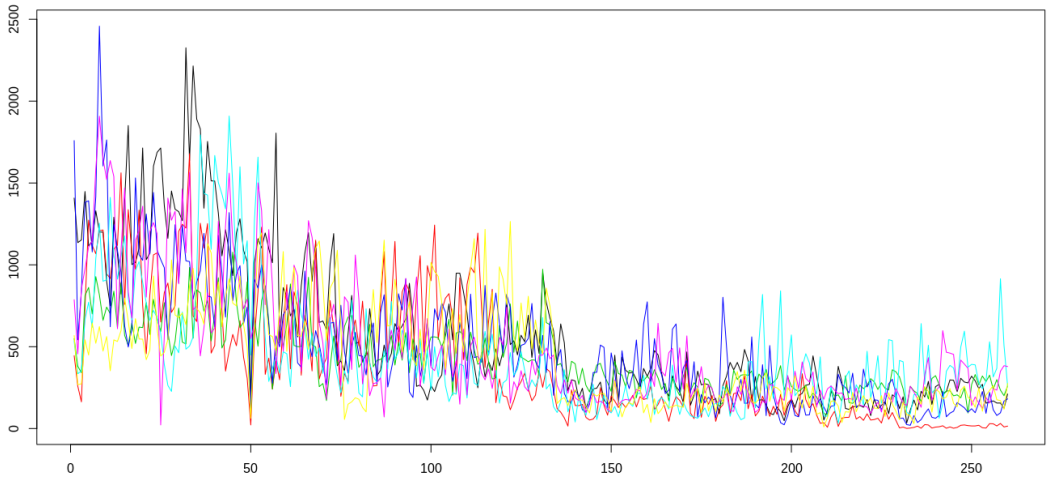
SORBONNE UNIVERSITÉ

POLYTECH SORBONNE

MAIN 3

Rapport de Projet d'Initiation

Approche Bayésienne pour l'étude des capteurs



Achille Baucher
Homer Durand
Marco Naguib

Encadrant : M. Zied BEN OTHMANE

22 Février 2019 — 5 Juin 2019

Nous tenons à remercier M. Zied BEN OTHMANE *pour sa participation et son aide.*

Table des matières

1 Introduction

Dans le cadre de notre formation en mathématique et informatique à l'école d'ingénieurs Polytech Sorbonne, nous sommes amenés à réaliser des projets d'initiation qui combinent les connaissances mathématiques et algorithmiques à des application en informatique et robotique. Nous, Achille Baucher, Homer Durand et Marco Naguib, avons choisi ce sujet sous l'encadrement de M. Zied Ben Othmane, qui s'intéresse à l'étude des capteurs, en particulier le degré de fiabilité des capteurs.

Imaginons qu'on dispose d'un grand nombre de capteurs qui réalisent des mesures sur une période de temps. Les valeurs mesurées par ceux-ci ne peuvent jamais être exactement égales aux valeurs réelles. En effet, ces capteurs subissent constamment des bruits externes et des déficiences internes, on ne peut donc pas faire aveuglément confiance à ces mesures. Dans ce projet, on se propose d'étudier et de programmer des outils qui permettent de comparer les valeurs mesurées par les capteurs entre elles, afin d'avoir une idée sur la fiabilité de chacun des capteurs.

1.1 Git

Le code est disponible à cette adresse :
`https://github.com/acSpinoza/capteurs.git`
Il faut exécuter le fichier ACTION.R pour obtenir les résultats. Un README.MD est disponible pour plus d'informations sur le code.

2 Présentation du problème

Nous avons à notre disposition N capteurs notés c_i pour $1 \leq i \leq N$, qui effectuent chacun M mesures au cours d'une période de temps. Nous avons pu récupérer l'ensemble des séries temporelles mesurées par les capteurs. La série temporelle mesurée par le i-ième capteur est une suite à valeurs réelles $(c_{i,t})_{1 \leq t \leq M}$. Notre mission est alors de créer un outil informatique qui sera capable d'identifier les capteurs les moins fiables, ceux qui ont une grande chance d'être déficients.

Dans notre cas, on dispose de $N = 630$ capteurs qui sont des robots qui pointent vers des sites web de magazines différentes. Chaque capteur mesure trois variables concernant ces sites web, (par exemple, nombre de visiteurs, nombre de bannières, etc.). On effectue la mesure de ces 3 variables M=350 fois sur une année. Or, ces valeurs mesurées par les capteurs ne sont pas exactes : Une bannière peut ne pas être prise en compte, un visiteur peut être compté deux fois...

Le but de ce projet sera d'étudier, pour chacune de ces trois variables, $(c_{i,t})_{1 \leq i \leq N, 1 \leq t \leq M}$, c'est à dire toute les mesures effectuées par tous les capteurs, afin d'attribuer à chacun des capteurs un degré de fiabilité, autrement dit, une probabilité que ses mesures soient vraies. Pour cela, il faudra procéder par étapes :

— **Classifier les capteurs (section 5) :**
Bien que les capteurs mesurent tous le même type de données, ceux-ci peuvent provenir de sources appartenant à des catégories λ différentes. La première étape consiste donc à identifier les différentes catégories auxquelles chaque capteur appartient.

Dans notre cas, les données proviennent de sites web de magazines variés, qui peuvent appartenir aux catégories sport, information en direct, voyages, etc. Nous pourront identifier les différentes catégories existantes à partir des variations et tendance des données récupérées : hausse pendant les évènements sportifs, en vacances, etc.

— **Définir une variabilité (section 6) :**
Pour identifier la fiabilité d'un capteur, nous avons décidé de nous appuyer principalement sur sa variabilité v par rapport à celles des autres capteurs de sa catégorie. La deuxième étape consiste donc à déterminer une mesure pertinente de la variabilité des séries renvoyées par chaque capteur.

— **Déterminer la fiabilité pour chaque capteur (section 7) :**
Nous pouvons alors analyser chaque catégorie, et y identifier les capteurs dont la variabilité est trop différente des autres. Ils seront considérés comme les moins fiables. *Nous avons remarqué que la plupart de nos séries avaient une variabilité très grande, avec des écarts grands et fréquents : les capteurs les plus constants seront donc interprétés comme potentiellement déficients.*

3 Observation des données

Les données sur lesquelles nous travaillons peuvent prendre des formes diverses que nous avons observé pour mieux adapter nos outils futurs. Ce sont généralement des séries très variables, comme le montre le graphique suivant.

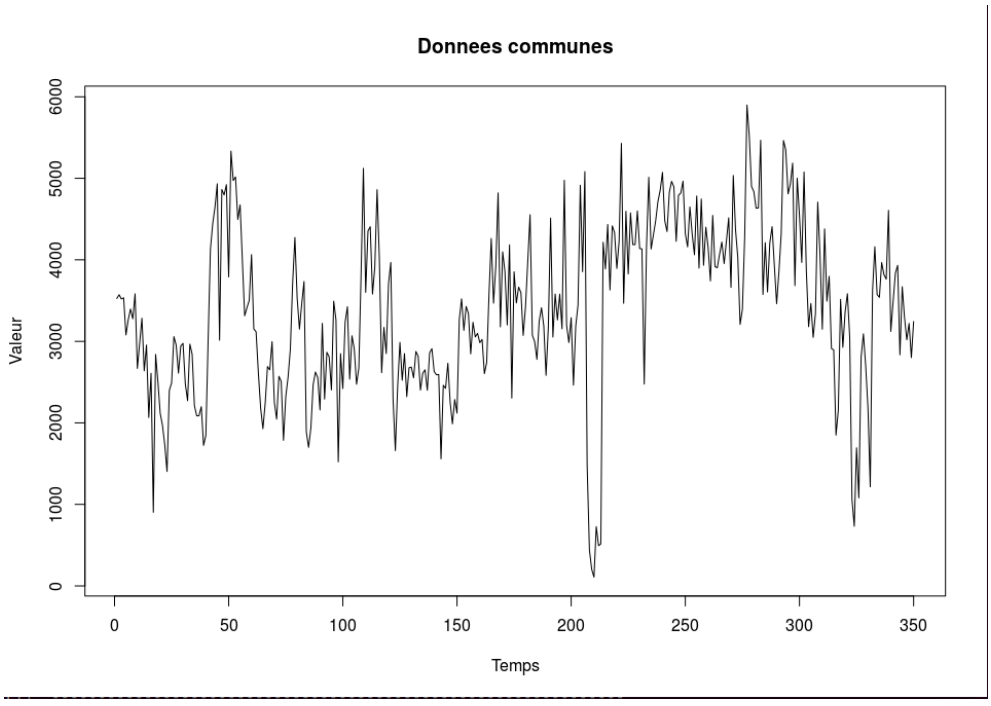


FIGURE 1 – Une série très variable

Il est fréquent d'observer des changements de tendance très brusques, comme on peut le voir sur ce graphique :

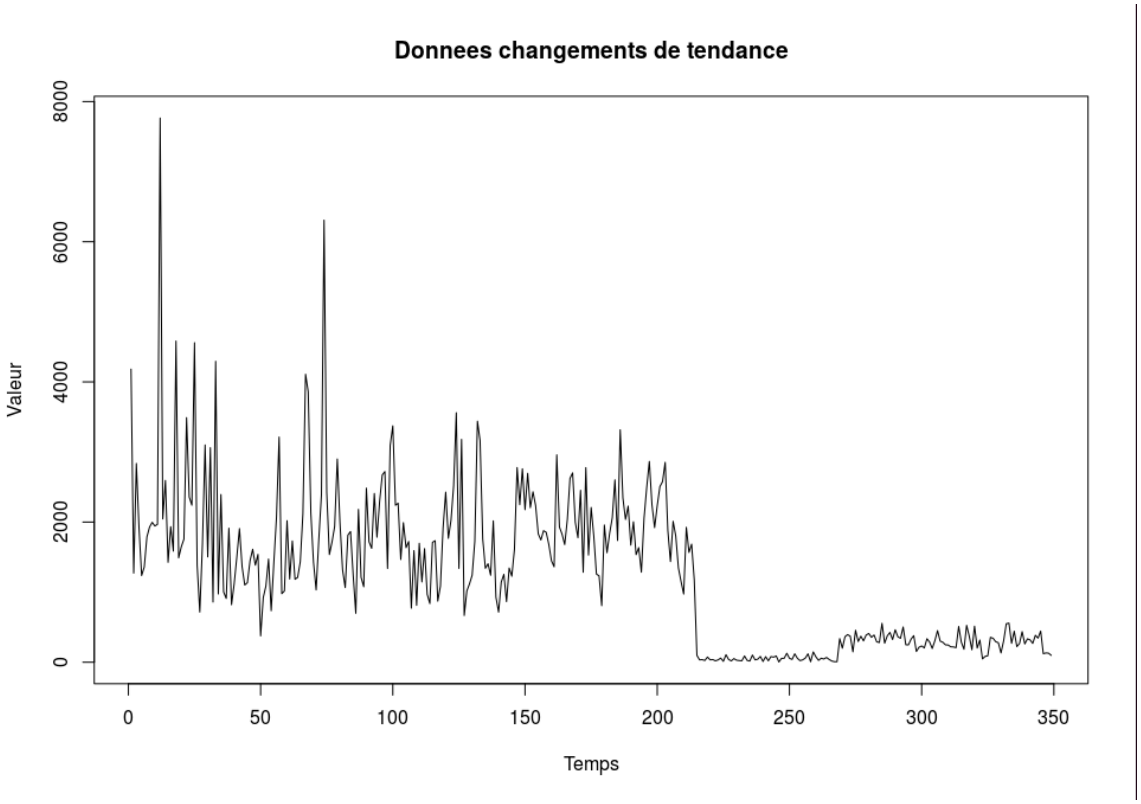


FIGURE 2 – De brusques changements de tendance

Parfois, les données présentent un plafond, qui est très souvent atteint. Il est assez probable que ce type de capteur soit déficient :

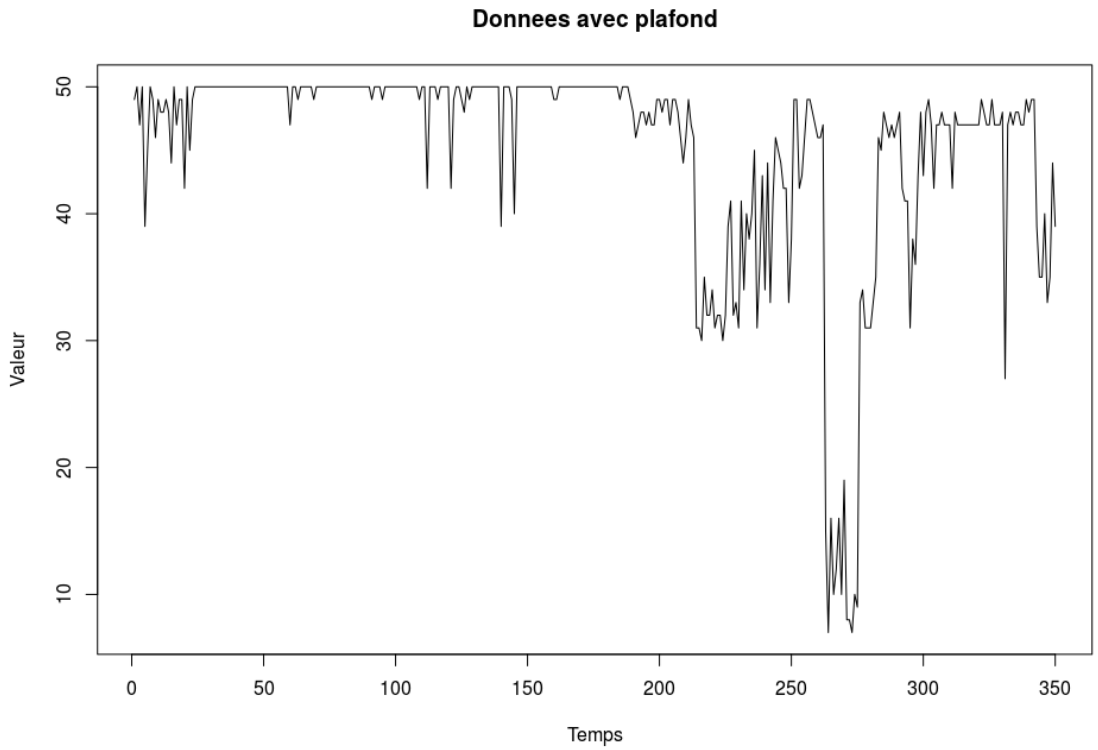


FIGURE 3 – Un capteur plafonné

Sur un même site, les données des trois robots sont parfois très corrélées, parfois moins.

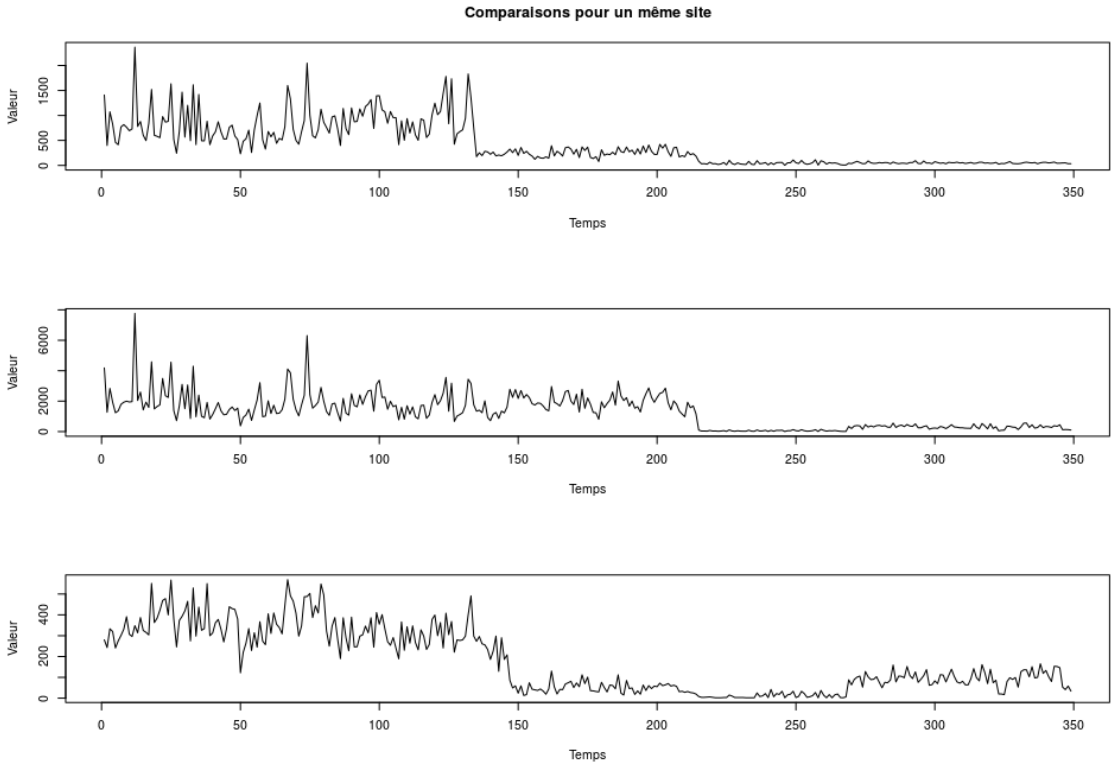


FIGURE 4 – Corrélations sur un même robot

Entre les différents sites, on observe plus rarement des corrélations.

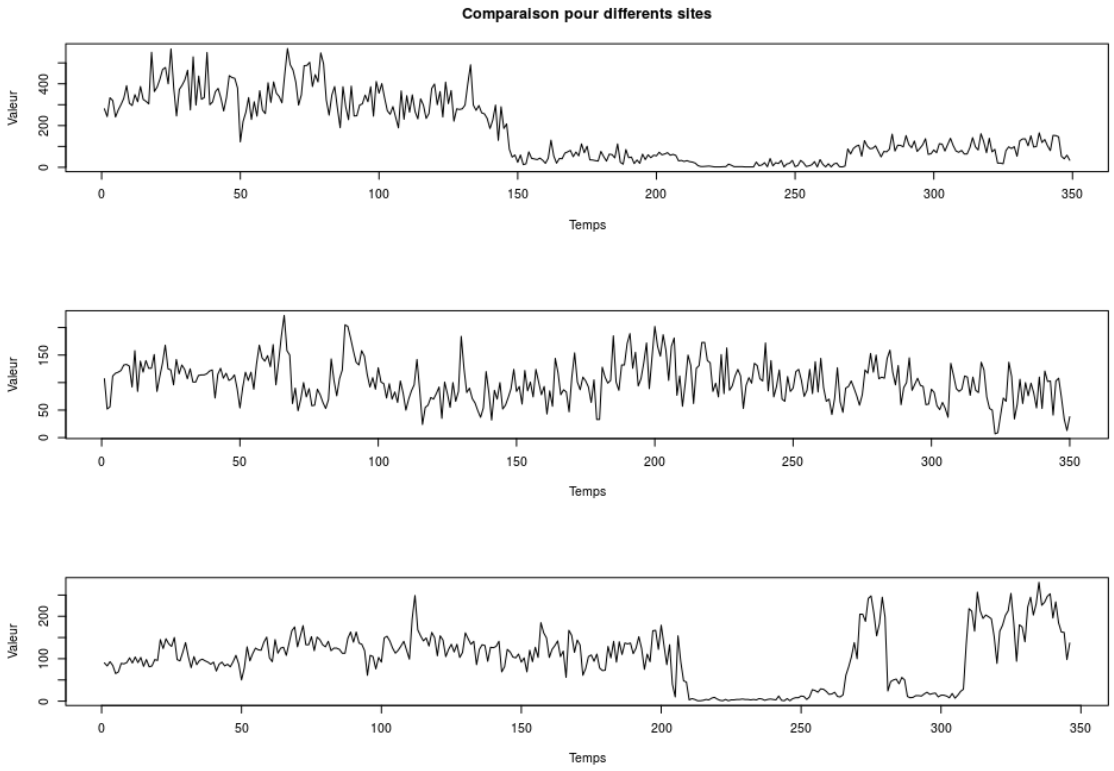


FIGURE 5 – Corrélation entre différents robots

4 Fonctions clés pour identifier les séries

Dans l'idée de proposer une mesure de variabilité des données pertinentes, nous avons créé plusieurs fonction pouvant nous permettre d'identifier les propriétés des séries temporelles qui sont analysées.

4.1 Repérer les changements de tendance

Nous avons remarqué de très forts changements d'échelle selon les périodes d'une même série temporelle (voir figure Tendances). Il est donc nécessaire de distinguer ces différentes périodes pour pouvoir les analyser indépendemment : c'est le rôle de la fonction `SEP_TENDANCES`, qui renvoie les différents instants de changement de tendance, sous forme de liste.

Fonctionnement :
On commence par séparer la série en plusieurs intervalles (de taille fixée par le paramètre `MINCAT`), et à calculer des moyennes pour chacun d'entre eux. Ensuite, on examine chaque intervalle pour déterminer si sa moyenne est proche de celle de son voisin : c'est à dire si leur différence est plus petite qu'un pourcentage `PCTMOY` de leur valeur. Si c'est le cas, on fusionne les deux intervalles et on en déduit une nouvelle moyenne. Sinon, on passe à la suite. Les différents intervalles et leurs moyennes correspondantes sont représentés sur ce graphique :

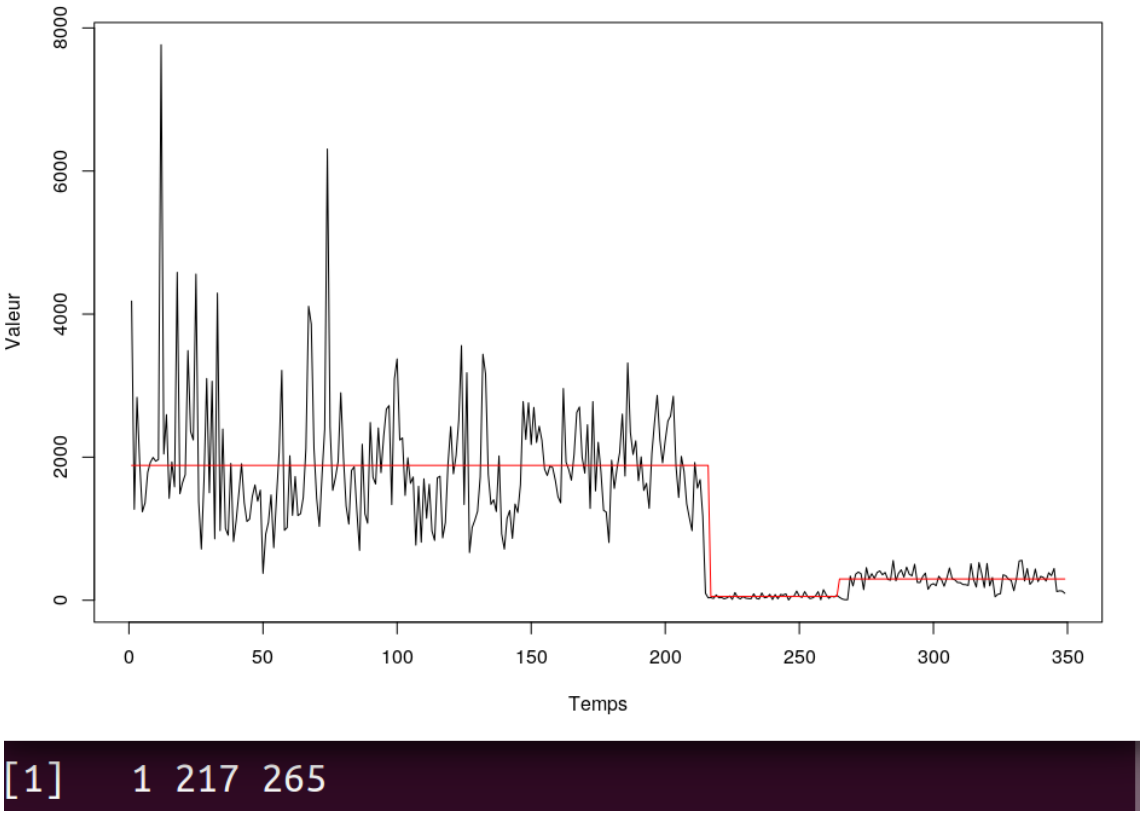


FIGURE 6 – Résultats de la fonction `SEPCAT`

Nous pouvons voir ci dessus que les principaux changements de tendance sont en 217 et en

265. Nous avons remarqué après différents tests que de bonnes valeurs pour les paramètres étaient :
`MINCAT` = 24
`PCTMOY` = 0.2

4.2 Repérer les données plafonnées

La fonction `pctEgal` indique quel pourcentage la valeur maximale occupe dans les données. Si elle occupe un trop gros pourcentage, on en déduit que les données sont plafonnées

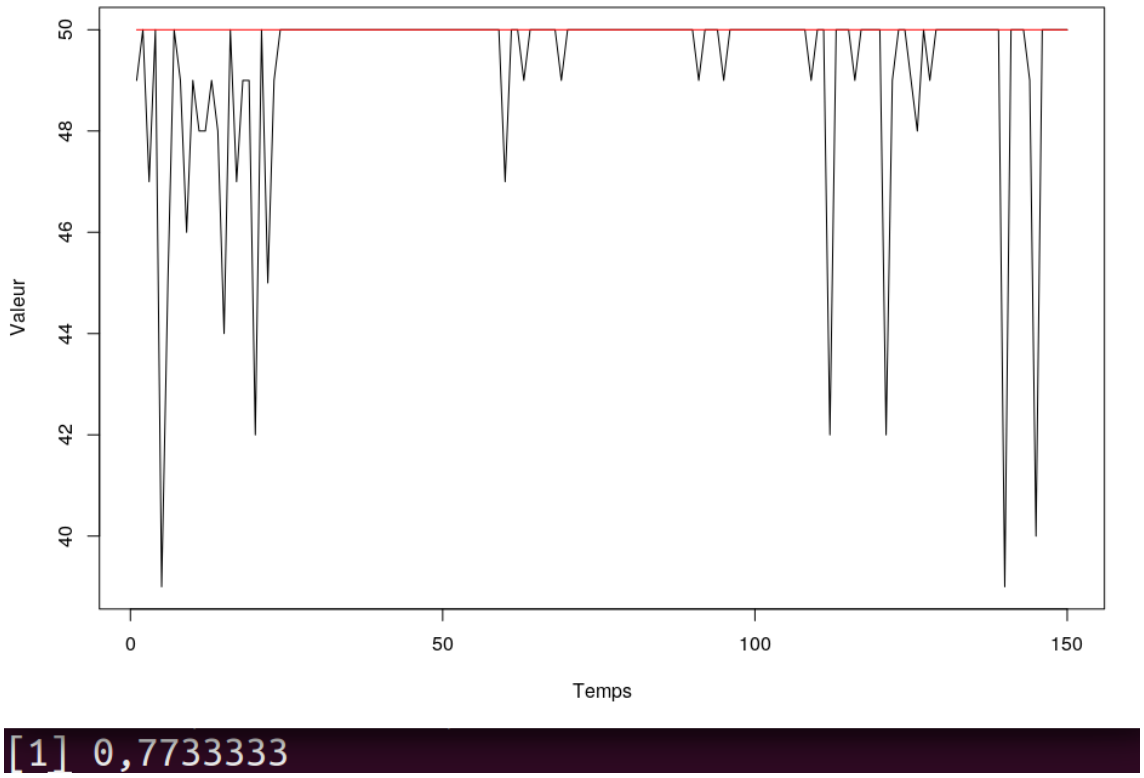


FIGURE 7 – Un plafond repéré par la fonction `pctEgal`

On peut voir sur cette figure que le plafond est atteint 77% du temps

5 Classifier les capteurs

Comme l'on a vu précedemment, il est utile de classer les capteurs en catégories (clusters) en fonction de leurs valeurs et de la forme de leurs séries temporelles. Dans notre exemple, chaque capteur pointe vers un site web d'une magazine. Il peut être intéressant de séparer les magazines pointées en fonction de leurs catégories (sport, voyage, etc) pour ensuite pouvoir comparer chaque capteur à ses "voisins" de la même catégorie et non à tous les capteurs. En effet, les magazines d'une même catégorie ayant des tendances similaires (les magazines de voyage ayant un maximum de visiteurs en avril/mai par exemple), il est plus intéressant de

comparer les capteurs correspondants entre eux, qu'à un capteur d'une autre catégorie, (magazine de sport par exemple).

5.1 Classification Ascendante Hiérarchique

Afin de classer les capteurs en catégories cohérentes et pouvoir ensuite étudier leurs variabilités au sein de celles-ci, nous avons utilisé un algorithme de "Classification Ascendante Hiérarchique" dit CAH. Nous avions besoin, pour pouvoir l'exécuter, d'une mesure de similarité ou de dissimilarité entre deux séries temporelles s_1 et s_2 de longueur N.

5.1.1 Derivative Dynamic Time Warping

La première et plus simple méthode pour mesurer la similarité entre deux séries temporelles est la distance euclidienne "Euclidian Distance" (ED). Nous noterons cette distance d_{ED} et la définissons comme suit :

$$d_{ED}(s_1, s_2) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_{1_i} - s_{2_i})^2} \tag{1}$$

On obtient ainsi une première idée de similarité entre deux série temporelle mais qui ne tient pas compte des dépendances temporelles. Il nous a donc paru intéressant de rechercher des méthodes de calcul de similarité plus adaptées à nos besoins.
Nous voulons, en effet, classer les séries en différentes catégories en fonction de leur forme.

On introduit alors une nouvelle méthode de calcul, Le "Dynamic Time Warping" (DTW) [3] qui permet de tenir compte des déformations temporelles. Cet algorithme consiste à faire correspondre les sous séquences qui se ressemblent, même si elles sont déphasées. Cette distance DTW est alors définie par :

$$d_{DTW}(s_1, s_2) = min\sqrt{\sum_{k=1}^{K} d_{ED}(k)} \tag{2}$$

avec $n \leq K \leq 2n$ la taille du chemin parcouru. L'idée étant de calculer une matrice des chemins reliants les premiers et derniers points en minimisant la distance cumulée, (voir figure 10).

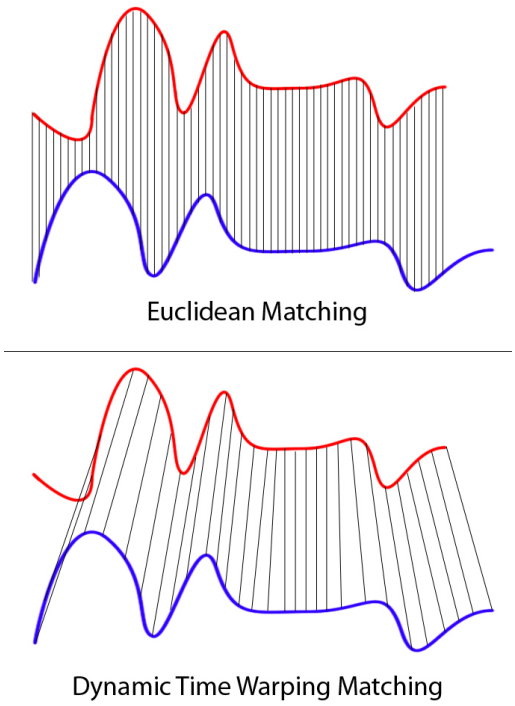


FIGURE 8 – Comparaison DTW et DDTW

On remarque donc que les sous séquences se ressemblant sont reliés avec la distance DTW quand la distance euclidienne se contente de relier les points ayant une même abscisse.

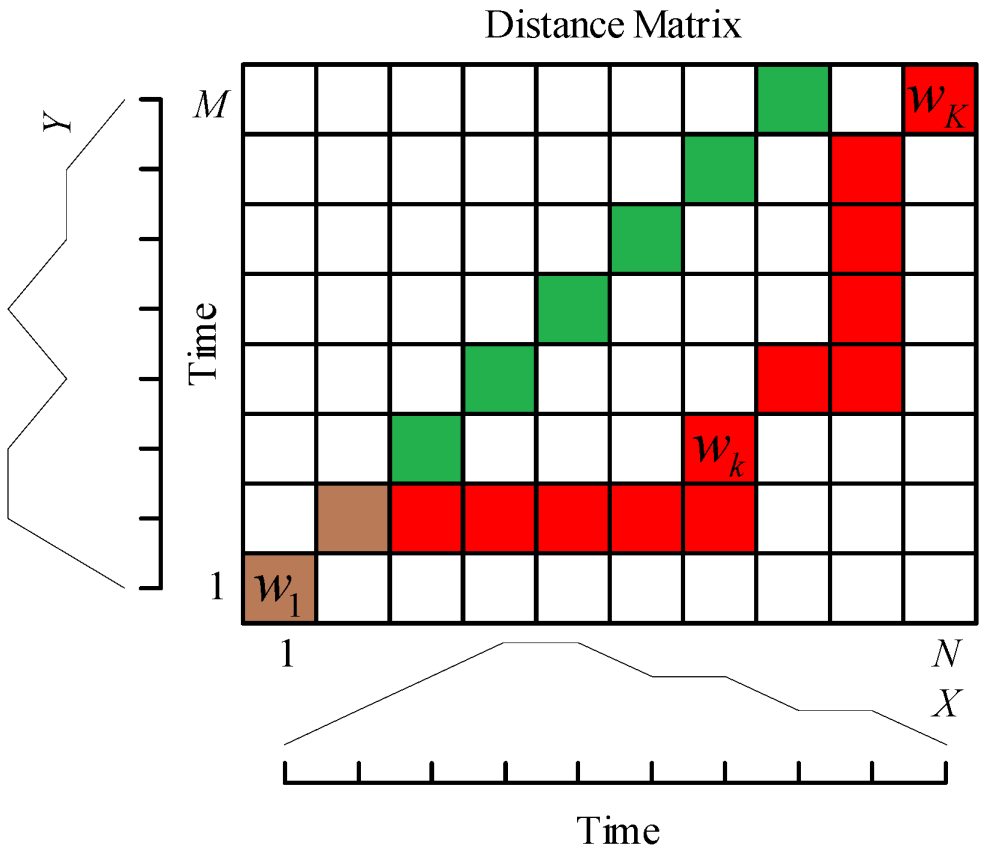


FIGURE 9 – Comparaison des matrices avec chemin euclidien et DTW

On a ici en vert le chemin choisit par la distance euclidienne d_{ED} et en rouge celui choisit par une distance DTW d_{DTW} afin de minimiser la distance.
Il existe plusieurs façon de définir ce chemin, nous avons choisit le celui décrit par le schéma A :

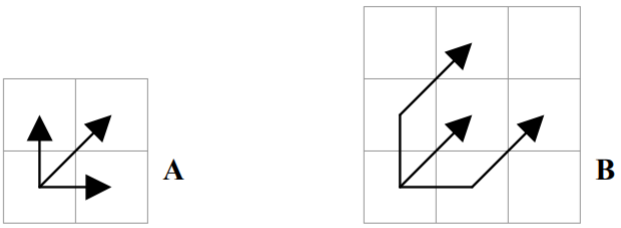


FIGURE 10 – Chemin pour DTW

On calcule donc une matrice M_{DTW} des distances de taille N × N comme suit :
On initialise la matrice avec $M_{DTW}(1,\ 1) = |s_1(1) - s_2(1)|$, puis les premières lignes et colonnes :
$M_{DTW}(1,\ \text{j}) = |s_1(j) - s_2(1)| + M_{DTW}(1, j-1)$ avec j > 1
$M_{DTW}(\text{i},\ 1) = |s_1(1) - s_2(i)| + M_{DTW}(i-1, j)$ avec i > 1
Enfin, on calcule les valeurs des autres cases pour i + j > 2 :
$M_{DTW}(\text{i},\ \text{j}) = |s_1(j) - s_2(i)| + min(M_{DTW}(i-1, j-1), M_{DTW}(i, j-1), M_{DTW}(i-1, j))$

Mais cette algorithme peut amener à des alignement de séquence non désirables dans notre cas car nous cherchons à reconnaître des patterns qui apparaîtrait sur des périodes similaires. Dans l'article "Derivative Dynamic Time Warping" [4] E.J.Keogh et M.J.Pazzani proposent une méthode qui répond à cette problématique.
On cherche désormais non plus à minimiser les distances cumulés entre les séries mais à minimiser leurs dérivées. On remplit donc une matrice M_{DDTW} avec maintenant les distances entre les dérivées respectives des séries étudiées puis on lui applique les même algorithmes que pour la distance DTW. Il existe plusieurs méthodes plus ou moins complexes pour calculer la dérivées d'une série discrète, nous avons choisit celle proposeée par Keogh et Pazzani pour sa simplicité de calcul :

$$d_x(s_{1_i}) = \frac{(s_{1_i} - s_{1_{i-1}}) + (s_{1_{i+1}} - s_{1_{i-1}})/2}{2} \tag{3}$$

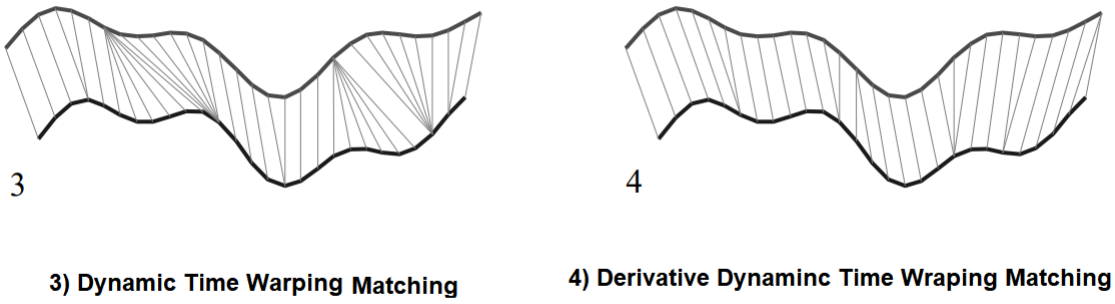


FIGURE 11 – Comparaison des distances DDTW et DTW

Cette méthode, comme on peut le voir, fournit de meilleurs performances en minimisant le nombre de points "dupliqués" et rend ainsi mieux compte de la correspondance entre les formes des séries.

Toutefois la mesure de similarité par la distance DDTW peut s'avérer coûteuse en temps de calcul car elle possède une complexité en $o(n^2)$ contrairement à la mesure de la distance euclidienne qui a une complexité en o(n).

Nous avons désormais un outil intéressant nous permettant de mesurer la similarité entre deux séries temporelles et pouvons donc chercher à les classer par catégories.

5.1.2 Classification Ascendante Hiérarchique

Cette méthode de classification consiste à répartir un ensemble d'individu dans un certain nombre de classes. Cette méthode est dite ascendante car elle part d'une situation où tous les individus forment leurs propres classes puis sont rassemblés dans des classe de plus en plus grandes à chaque itération jusqu'à obtenir une unique classe. On peut alors choisir le nombre de classe que l'on souhaite garder en remontant l'arborescence de la classification.
Le principe de cet algorithme est simple. Initialement, chaque série est seule dans une classe, nous avons donc n classes pour n séries. A chaque itération on fusionne deux classes réduisant ainsi le nombre de classe. Les classes fusionnant sont celles qui sont les plus proches, c'est à dire celles qui sont les plus similaires.

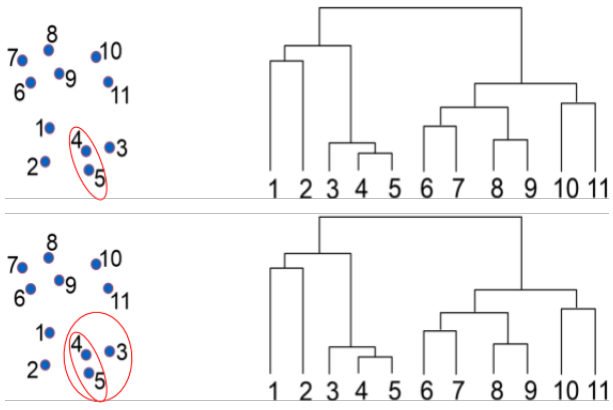


FIGURE 12 – dendogramme CAH

On voit ici qu'on commence par regrouper dans une classe les séries 4 et 5 qui sont les plus similaires. On voit ensuite que les séries les plus similaires sont les 3 et 4, on ajoute donc la 3 à la classe créée précédemment.

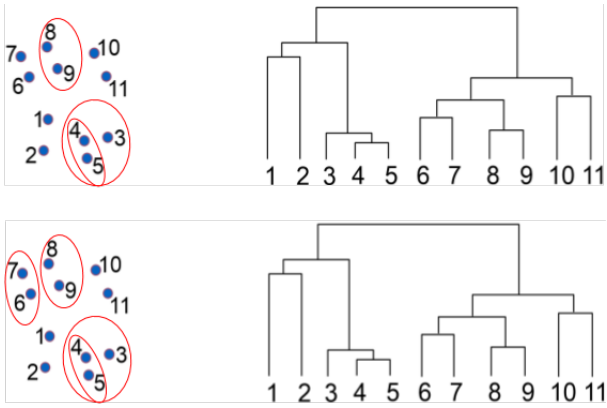


FIGURE 13 – dendogramme CAH

On regroupe ensuite les séries 8 et 9 et les séries 7 et 6 dans deux classes.

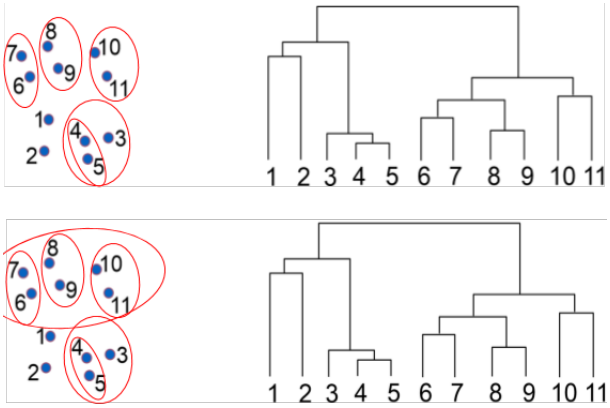


FIGURE 14 – dendogramme CAH

On continue de la sorte jusqu'à obtenir une unique classe dans laquelle se trouve toute les séries classées.

Il nous reste alors à choisir à quel niveau de l'arborescence on souhaite revenir pour avoir le nombre de classe voulu.

5.1.3 Problématique rencontrée avec CAH

Nous avons donc testé notre algorithme pour plusieurs capteurs. Mais nous avons remarqué qu'il créé très peu de classes et que celles-ci disparaissaient très vite. Comme on le voit sur la figure ci-après, tous les capteurs sont d'abord dans leurs propres classes et sont donc notées `NA`. Les capteurs 5 et 6 sont ensuite regroupés dans la classe 1. Puis une nouvelle classe 2 est créée avec les capteurs 3 et 4. Mais cette classe fusionne à l'itération suivante avec la classe 1. Et finalement, tous les autres capteurs s'y ajoutent et aucune nouvelle classe n'est créée.

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
[1,]	NA	NA	NA	NA	NA	NA	NA
[2,]	NA	NA	NA	NA	1	1	NA
[3,]	NA	NA	2	2	1	1	NA
[4,]	NA	NA	1	1	1	1	NA
[5,]	NA	NA	1	1	1	1	NA
[6,]	NA	NA	1	1	1	1	NA
[7,]	NA	NA	1	1	1	1	1
[8,]	NA	NA	1	1	1	1	1
[9,]	1	NA	1	1	1	1	1
[10,]	1	NA	1	1	1	1	1
[11,]	1	NA	1	1	1	1	1
[12,]	1	NA	1	1	1	1	1
[13,]	1	NA	1	1	1	1	1
[14,]	1	NA	1	1	1	1	1
[15,]	1	NA	1	1	1	1	1
[16,]	1	1	1	1	1	1	1
[17,]	1	1	1	1	1	1	1
[18,]	1	1	1	1	1	1	1
[19,]	1	1	1	1	1	1	1
[20,]	1	1	1	1	1	1	1
[21,]	1	1	1	1	1	1	1

FIGURE 15 – En abscisse les capteurs (de 1 à 7) et en ordonnée les itérations (de 1 à 21)

Nous avons donc cherché des méthodes plus efficaces et répondant mieux à nos attentes pour classer nos capteurs.

5.2 Algorithme de K-means

Dans un premier temps nous allons introduire une des méthodes de clustering (classification) les plus utilisées, l'algorithme des k-means [2], développé par James McQueen en 1967. Étant donné un grand nombre de points en dimension $n \geq 1$, on veut les classer dans des catégories, de façon à ce que les points de chaque catégorie soient les plus proches [1] possibles, et que les centres des catégories soient les plus éloignés possibles. Il s'agit d'un problème de complexité non polynomiale ; ce qui n'est pas pratique du moment qu'il s'agit d'un grand nombre de points, comme ici. L'algorithme des k-means donne une solution polynomiale à ce problème.

Posons $n = 2$, et supposons que l'on traite de 500 points, et que, en représentant les points sur un graphe (figure 16), on obtient la figure de gauche. On veut que les catégories obtenues correspondent, par exemple, à la figure de droite.

On précise à l'algorithme qu'on veut une classification en $k = 3$ catégories L'algorithme part de 3 points au hasard, et crée 3 classes de points de la manière suivante : un point appartient à la classe i si, entre les 3 points considérés, le point i lui est le plus proche. Une fois tous les points classés dans les 3 catégories, on calcule le centre de chaque catégorie (on prend la moyenne sur chaque coordonnée), on a maintenant les 3 centres des 3 catégories.

1. au sens euclidien

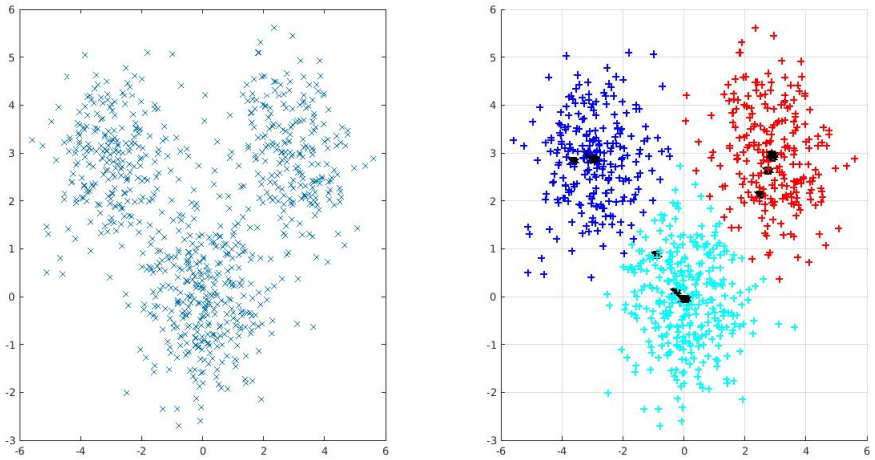


FIGURE 16 – Points classés en 3 catégories

Maintenant, on re-classe tous les points en fonction de leurs proximités respectives des centres. Puis on recalcule les nouveaux centres et on réitère le processus. L'algorithme s'arrête seulement quand il n'y a plus de changement entre l'itération n et l'itération $n+1$. Les classes ainsi obtenues à l'itération n minimisent (au moins localement) la somme des distances entre les points de chaque catégorie entre eux.

Il est possible de prouver que cet algorithme converge toujours, il existe pourtant de rares cas où la solution donnée par celui-ci ne correspond pas à la meilleure solution au niveau pratique. Ça reste quand même un algorithme assez optimisé pour résoudre un problème qui est à priori de complexité non polynomiale.

5.3 Classification des capteurs

Les données qu'on a récupérées sont dans un fichier `csv`. Chaque ligne indique, pour un instant donné, et pour un capteur donné, les 3 variables mesurées.

On va s'intéresser à une seule des trois données, $var1$ par exemple. On veut avoir une matrice D de 610 lignes, et de 350 colonnes, telle que $d_{i,j}$ représente la j-ième mesure de la variable $var1$ réalisée par le i-ème capteur. Pour cela, on a utilisé la programmation python pour gérer le chaînes de caractères et rendre cette matrice. Maintenant, chacun des 610 capteurs est représenté par un vecteur de 350 composantes, qui sont sur la ligne correspondante dans la matrice D. On peut donc, au moins numériquement, associer à chacun de ces capteurs un point (de dimension 350 !) et ensuite appliquer à l'ensemble des 610 points l'algorithme des k-means comme précédemment. Deux capteurs seront donc classés dans la même catégorie si les vecteurs représentants les valeurs mesurés par chacun d'eux sont proches au sens euclidien. Ainsi, chaque catégorie devrait contenir de capteurs qui ont mesuré des valeurs relativement proches pour toutes les mesures. Il sera donc pertinent de comparer ces deux capteurs entre eux plutôt qu'un troisième capteur qui n'appartient pas à la même catégorie.

5.4 Résultats de la classification

On a classifié les capteurs en 60 catégories. On choisit ensuite de représenter sur un graphe les mesures par les capteurs appartenant à une catégorie donnée, voici ce les mesures réalisées par des capteurs appartenant à la même catégorie.

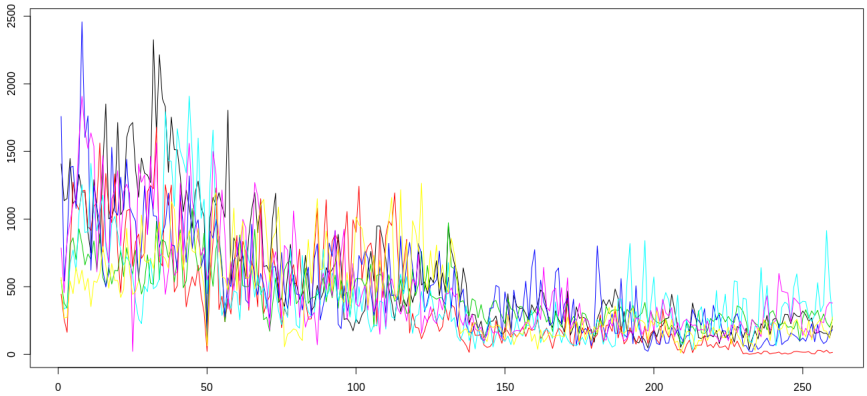


FIGURE 17 – catégorie 5

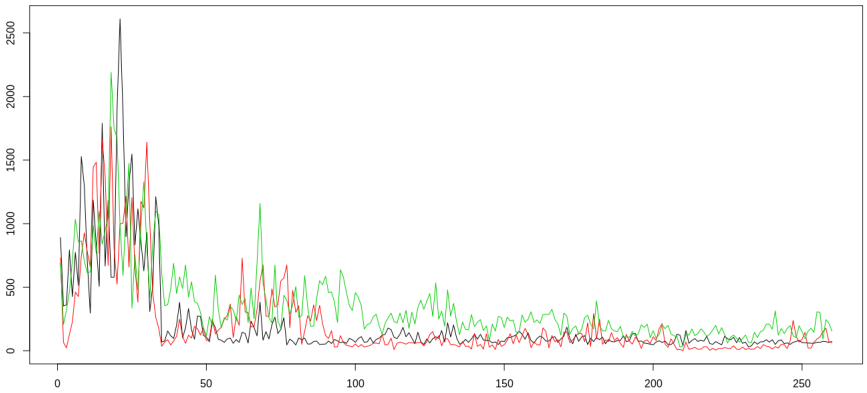


FIGURE 18 – catégorie 14

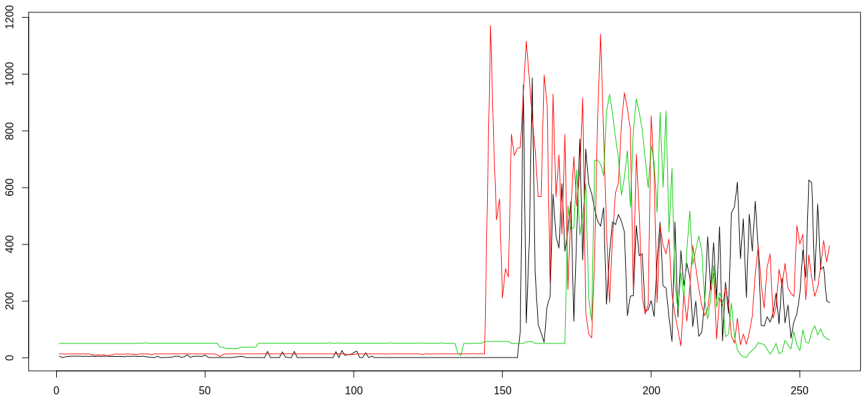


FIGURE 19 – catégorie 18

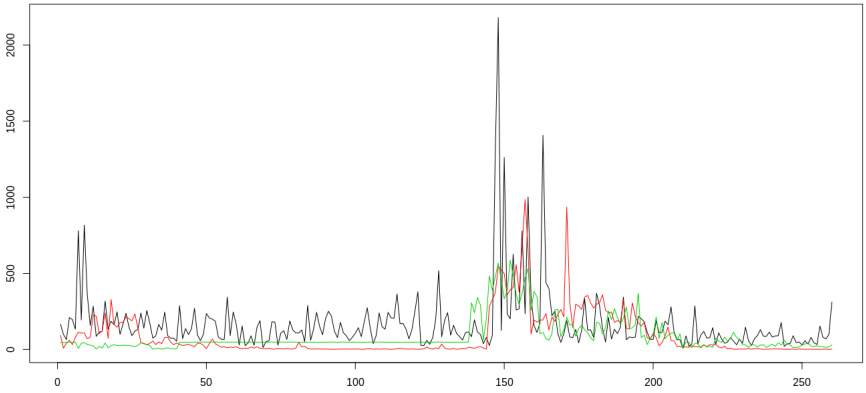


FIGURE 20 – catégorie 40

6 Déterminer la variabilité des séries temporelles

6.1 Variabilité ETP

Une première mesure de la variabilité de séries temporelles que nous avons utilisé est un écart-type pondéré par la moyenne (ETP) de valeurs de la série X : :

$$\nu(X) = \frac{\sigma(X)}{\overline{X}} \tag{4}$$

Ce choix est justifié par l'observation que les variances des séries étaient généralement proportionnelles à leur moyenne locale. De plus, comme nous l'avons expliqué dans la section 4.1, nous avons remarqué que chaque série temporelle était soumise à des changements de tendances. Il nous a parût dès lors plus intéressant de calculer les variabilités ν_i sur chacune des sous séries X_i puis de calculer la moyenne de ces variabilités afin d'obtenir une variabilité globale ν pour l'ensemble de la série.

$$\nu(X) = \frac{1}{n}\sum_{i=1}^{n}\nu(X_i) \tag{5}$$

Avec n le nombre de sous-séries de X.
Nous avons remarqué au cours de nos essais que cette mesure de la variabilité convenait tout à fait à notre analyse et c'est donc elle que nous avons utilisé.

7 Déterminer les fiabilités des capteurs d'une catégories

Les deux étapes précédentes nous fournissent l'ensemble des variabilités $v_j, j \in \{1, ..., N\}$ des capteurs de chaque catégorie. On se concentre à présent sur une catégorie λ, et on essaie de déterminer la fiabilité de chacun de ses capteurs en comparant leur variabilité à celles de l'ensemble de leur catégorie. La fiabilité $f(c_a)$ du capteur c_a dans une catégorie $\lambda(c_a)$ est la probabilité que sa variabilité v_a soit correcte sachant toutes les variabilités de sa catégorie : Soit V_a l'évènement : la variabilité v_a est correcte. La fiabilité est alors :

$$f(c_a) = P(V_a|\{v_i, i \in \{i/\lambda(c_i) = \lambda(c_a)\}\}) \tag{6}$$

Notre objectif est donc de trouver des fonctions pertinentes qui calculent cette probabilité.

7.1 Fiabilité par intervalles

Idée :
L'idée de la fonction `f_intervalles` est de prendre un intervalle I_{c_a} centré en la variabilité v_{c_a} du capteur c_a en question et de retourner la proportion des capteurs de sa catégorie qui appartiennent à cet intervalle. Ainsi, les intervalles centré en les valeurs les plus éloignées des autres donneront une très faible fiabilité.

FIGURE 21 – Une bonne (vert) et mauvaise (violet) valeur de variabilité parmi les autres de la catégorie

Une difficulté réside dans le choix de la taille T de l'intervalle. On peut la déterminer comme une proportion (dépendante d'un paramètre σ) de la distance qui sépare les bornes de l'ensemble des variabilités de la catégorie.

$$T_\lambda = \sigma(max(v_\lambda) - min(v_\lambda)) \tag{7}$$

Premiers résultats : Nous avons donc essayé cette fonction pour plusieurs valeurs de σ et deux variables d'un capteur :

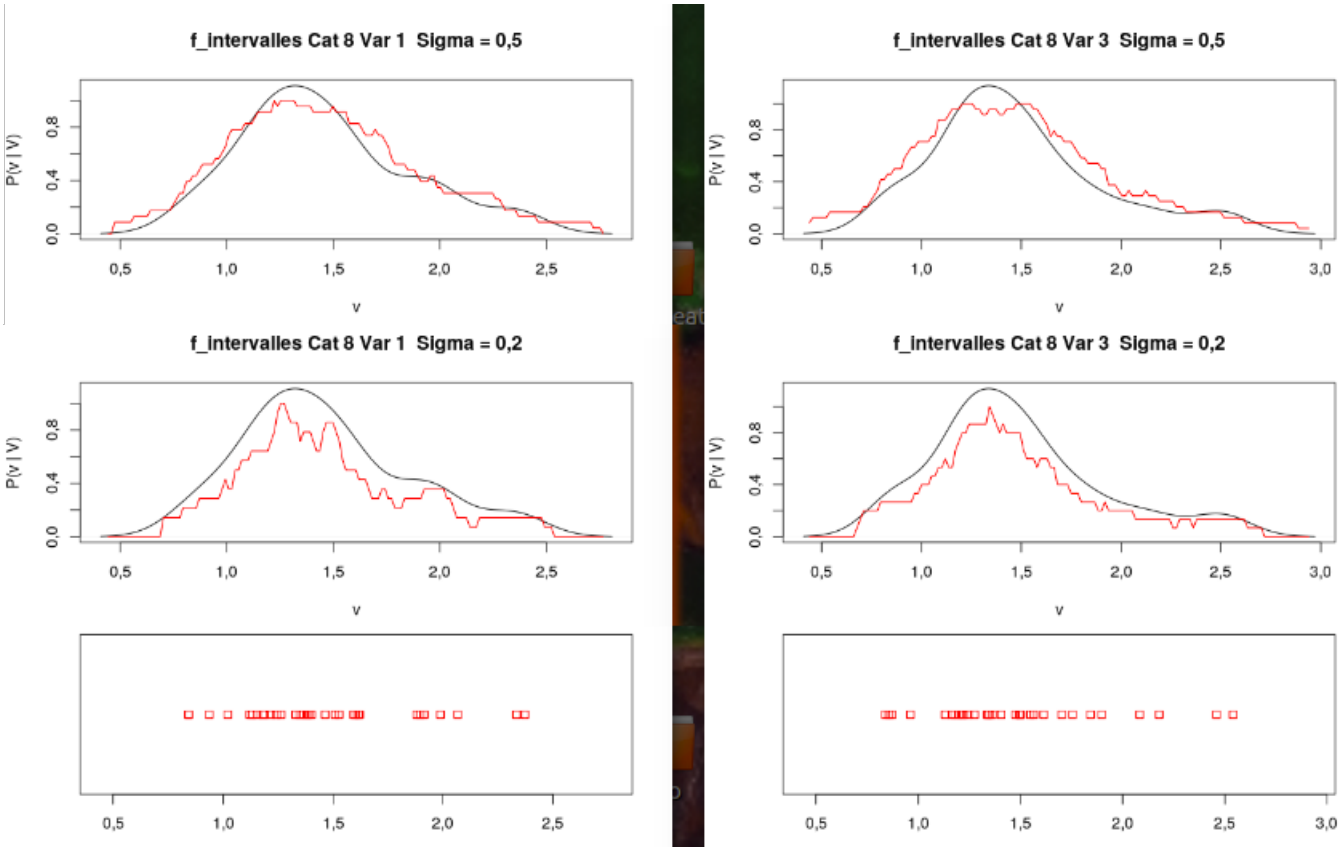


FIGURE 22 – En rouge les courbes des fonctions f_intervalles et en noir les densités

Les densités ont été calculées par la fonction `density` (en noir) de matlab pour vérifier la pertinence de notre fonction `f_intervalle` (en rouge). En bas du graphiques, ce sont la répartition des variabilités des capteurs qui est représentée, pour nous aider aussi à estimer la précision de notre fonction.
On remarque que pour $\sigma = 0.2$ les courbes ont l'air plus justes et paraissent même plus pertinentes que les densités. Mais nous avons réalisé plus tard que plus les capteurs que l'on considéraient étaient nombreux, plus une valeur faible de σ est pertinente, Nous avons donc choisi dans la suite un $\sigma = 0.1$.
C'est donc cette fonction et cette valeur de σ qui nous servira dans la suite pour déterminer la fiabilité de chaque capteur.

7.2 Fiabilité par distance de ses voisins

L'idée de la fonction `f_dist_voisins` est de considérer l'ensemble V_{c_a} des voisins les plus proches de la variabilité v_{c_a} du capteur c_a en question et de retourner l'inverse de la somme des distances avec v_{ca} au carré . On décide au début le nombre de voisins qu'on va prendre, il parait optimal de le fixer autour de 10.

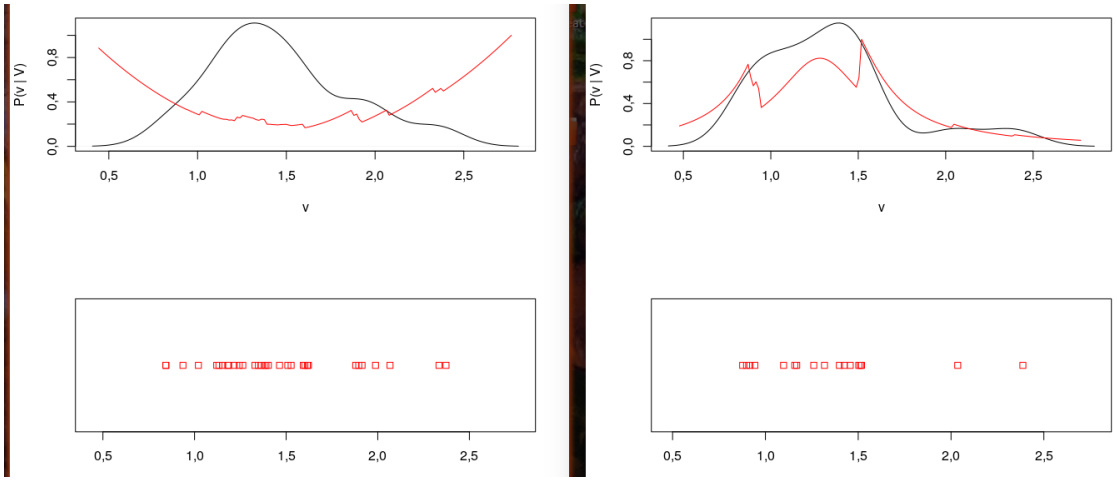


FIGURE 23 – Deux courbes associées à deux catégories différentes

Nous avons abandonné cette façon de mesurer les fiabilités car elle nous paraît beaucoup moins précise que la précédente.

8 Résultat final

Nous avons donc pu :

- Classifier nos capteurs dans plusieurs catégories différentes, grâce à l'algorithme des k-means.
- Déterminer pour chacun une valeur de variabilité, avec la fonction `variabilite_ETP`.
- En déduire une fiabilité pour chacun des capteurs, avec la fonction `f_intervalles`.

Il ne nous reste à présent qu'à parcourir chaque catégorie et à repérer les capteurs avec les valeurs de fiabilité les plus faibles, pour constater si effectivement ils diffèrent des autres.

8.1 Demonstration pour une catégorie

Clustering :
Nous choisissons arbitrairement de former 20 catégories et l'algorithme des k-means nous renvoie la répartition suivante :

```
 [1] 13  3 12 13 17 17 19  2 13  2 17 19 12  5 17 17 19  8 13 17  5  7  5 17 17
[26]  5  5 17 11 13 13 14  1  1  5 17 19 13  8  5  5 12 15 17 17 19 13 20  2 17
[51] 19 19 17 11 17 17 12 13 13  5 12 18 12 13 15 13 12 13  8 19 16 13 16  2  5
[76]  5  1  5 17  2  2 12 12 13 17 14 14 16 17 12 12 17  7 12 19  2 13  2 12 13
[101] 17 19 17 16  3  1 12  2 12  5 17  2 13  5 12 13 12 12 17  2 17 17 12 17 12
[126] 19 19 14  7 19 17  1 12 17 17  5 17  2  1  8 17 17 12 17 12 12 13  5 12  5
[151] 10  5 17  5 11  5 19  1 13 15  8  4  2  5 13 17  2 17 17 19  9 19 13 13 13
[176] 17  3 13 19 14 10 13  5 17 17  2 17 12  2 19 12  2 17 19 17 17 15 12
[201]  5 19 15  2 17 12  3  8 12  1  2  7 13  1 17 19  7  1  5 12 17  5 11  8 16
[226]  4 12 13 12  3 13 13  5  2  5 13  2 17 17 13 17 19  5 14  2 12 19  5 17 15
[251]  2 16 13  3  5 17 17 19  3 17 13 19 12 13  8 17 17 17 14 19 12 12 17 13 13
[276] 10 12 13 12  8  5 17 17 16 17 17 17  5 12 12  8 12 19 19 12  5 17 17 17 17
[301] 14 12  8 12 13  5 13  2 13 15 17 17 12 19 17 14  7 17 17  3 12 17 19 15 17
[326] 17 17 17 17 12  1 12 19 17 12  5 14 17  5 15 17 12  3 17 17 13 12 19 19  7
[351]  2 17 14 15 13  8  8 13  8 12 13 12 12  5  5  5 13 19 12 17 17 17 19  2 19
[376] 12 17 17  8 12  8 12 12 12 19  2 17 10 15 13 13 19  7 17  5 13 16 17 17 15
[401]  2 17 19 13 13 13 12  1 16 17 19  1 17 13 12 13  5 16  5 12 19  3 17  5 14
[426]  6 19 12 15  5 13  1 13 17 13 12 17 12 17 13  5 17  1  3 14 19 13  8 12 17
[451] 17 13 17 17 12 17 12 17 13 12 12  5 17 19  5 17 12 16 17 12 17 13  8 17 17
[476] 12  5  5 17  1 13  7  5  5 13  5  5  5  2 11  6  1 14 13 12  5 12 12 12  5
[501] 13 19  2 17  2 17  5 13 13 13 12 12 14 13 13 17 19 12 12  8 12  3 11  6 17
[526] 12  5 17  2 17 17 15 12 17  2 12  8  5  5  4 19 19 12  5 17 13 19 14 19  2
[551] 17 17 13 17 12 19 12 13 12
```

FIGURE 24 – Le tableau contenant la catégorie de chaque capteur

Nous remarquons une forte présence de la catégorie 17, à laquelle nous pouvons nous intéresser. On affiche alors toutes les les courbes de la catégorie sur un même graphique :

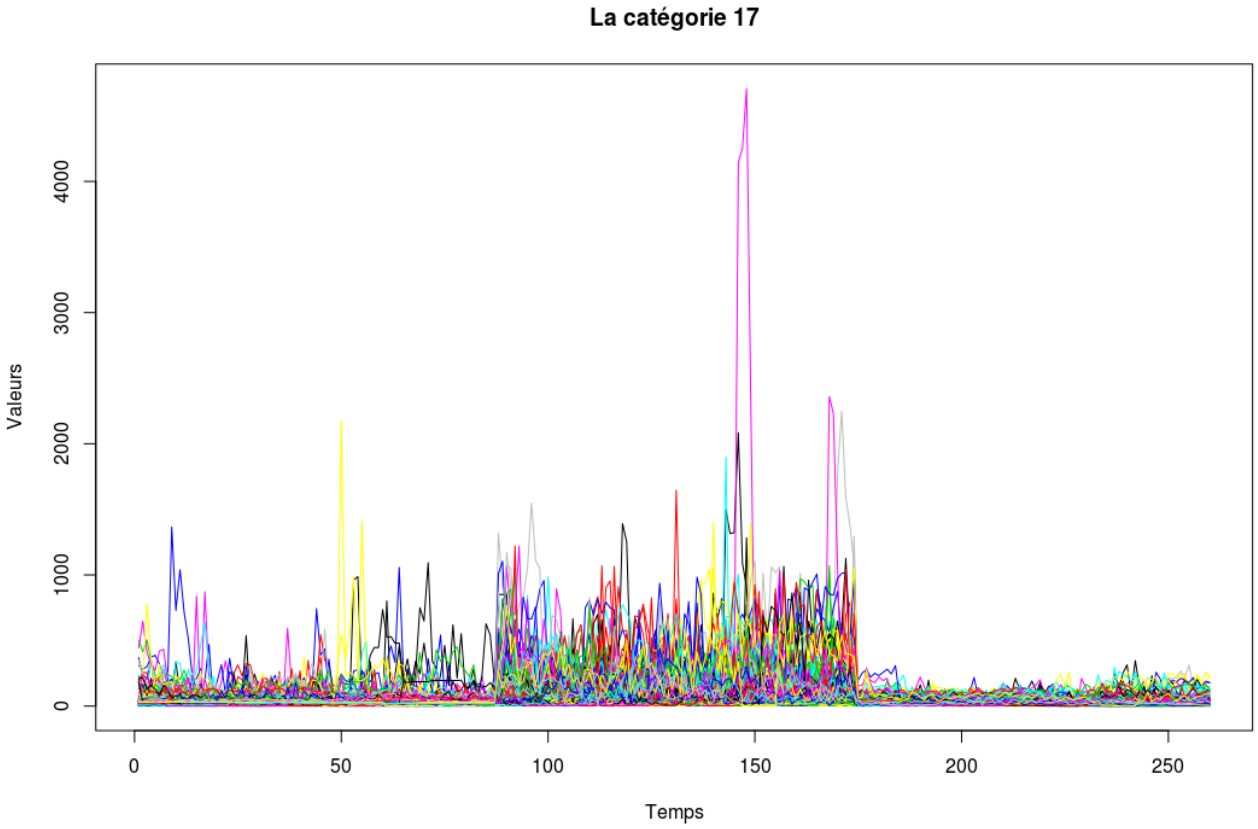


FIGURE 25 – Les courbes de toutes les capteurs de la catégorie 17

Variabilités et fiabilités

Les variabilités sont calculées et on obtient la répartition suivante :

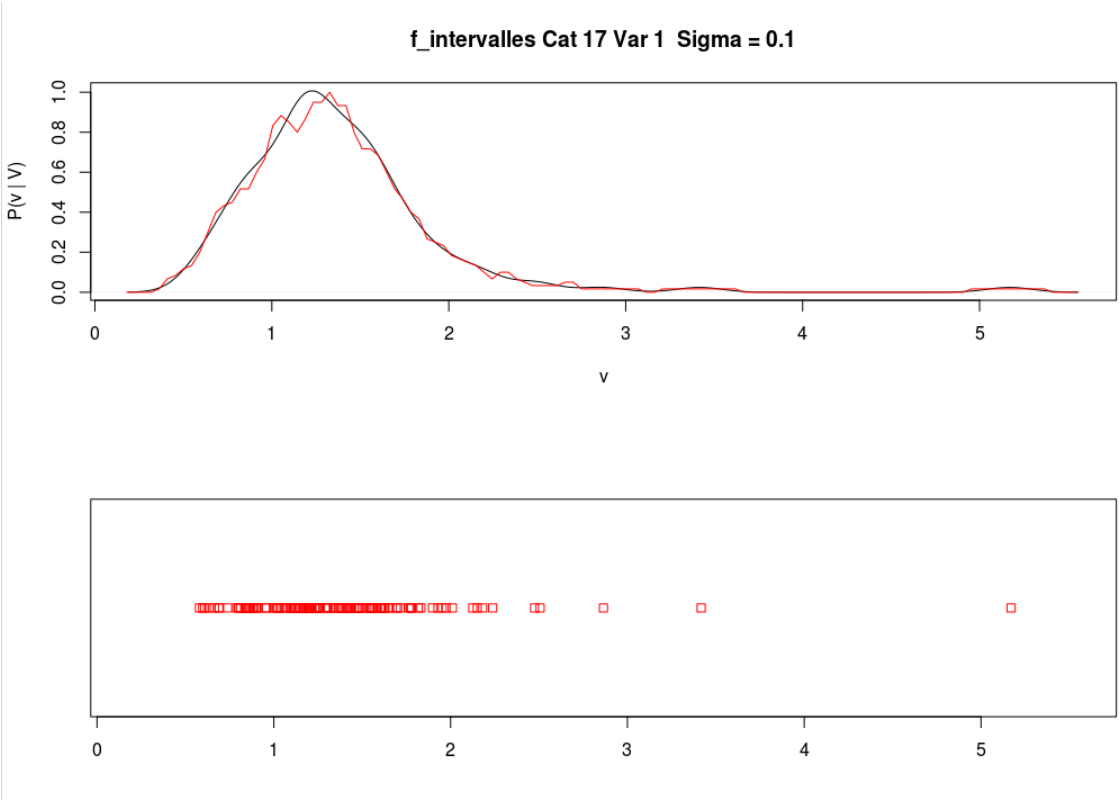


FIGURE 26 – Répartition des variabilités de la catégorie 17 (en bas), et les résultats de l'étude des fiabilités de cette catégorie (en haut, rouge)

On remarque que certains points, sont très différents des autres, la catégorie comporte donc sûrement des capteurs déficients. Nous demandons donc au programme de renvoyer les 2 capteurs les moins fiables de cette catégorie, ainsi que deux capteurs très fiables pour avoir un élément de comparaison. En affichant les courbes de ces 4 capteurs :

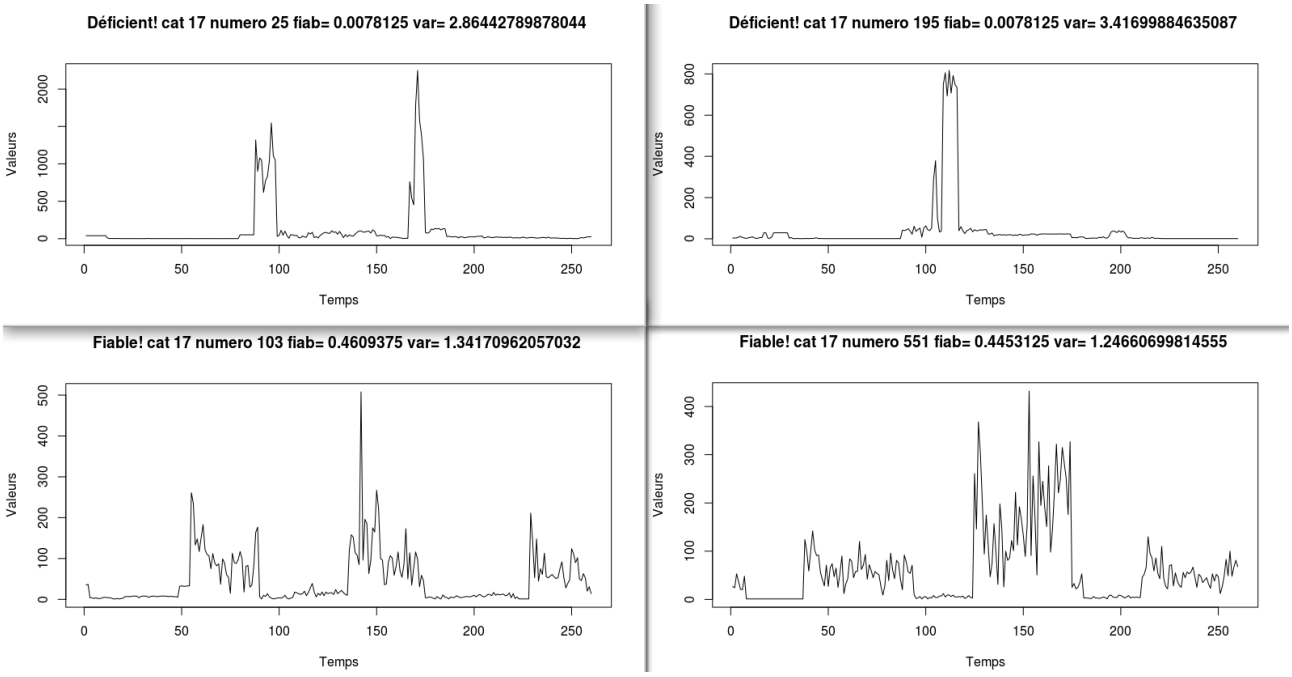


FIGURE 27 – Comparaison de deux capteurs déficients (en haut) et de capteurs fiables (en bas)

On s'aperçoit effectivement que les capteurs les repérés par notre algorithme comme déficients ont une forme très étrange par rapport aux autres.

9 Conclusion

Finalement ce projet aura été pour nous une première approche de l'analyse statistique. Dans un premier temps, il nous aura permis de nous familiariser avec certains outils très utilisés dans ce domaine qui nous seront très certainement utiles dans la suite de nos études et de notre parcours professionnel ou universitaire. C'est le cas de l'algorithme des K-means, des distances Dynamic Time Warping et Derivated Dynamic Times Warping et de la Classification Ascendante Hiérarchique.
Dans un second temps, nous avons pu développer nos propres outils et algorithmes afin de répondre à notre problématique. Certain ayant de très bons résultats comme la mesure de variabilité ETP (Ecart-type pondéré), la fiabilité par intervalles ou le repérage des changements de tendances.

Bien que notre système aboutissent à des résultats concrets, nous avons remarqué durant notre analyse plusieurs pistes d'amélioration de l'algorithme. Premièrement, le nombre de catégories est pour l'instant choisi arbitrairement, mais nous avons vu lors de notre documentation des méthodes qui permettraient de l'estimer et qui pourraient donc être intéressantes de développer. Ensuite, nous avons remarqué que les capteurs pouvaient être fiables durant une période et aberrants durant une autre : il serait donc judicieux de préciser notre algorithme en renvoyant les périodes durant chaque capteur est fiable ou non.

Références

[1] Classification ascendante hiérarchique.

[2] A. H. Djeffal. Clustering. `www.abdelhamid-djeffal.net/web_documents/diaposclustering.pdf`, Septembre 2017.

[3] J. C. Donald J. Berndt. Using dynamic time wrapping to find patterns in time series. Avril 1994.

[4] M. J. P. Eamonn J. Keogh. Derivated dynamic time wrapping.

[5] A. J. P. M. M. S. EHala Najmeddine, Frédéric Suard. Mesures de similarité pour l'aide à l'analyse des données énergétiques de bâtiments. Janvier 2012.