CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

CENTRE RÉGIONAL ASSOCIÉ DE NOUVELLE-AQUITAINE

_____

MÉMOIRE
présenté en vue d'obtenir
le DIPLÔME d'INGÉNIEUR·E CNAM

SPÉCIALITÉ : Informatique
Parcours Big Data et Intelligence Artificielle
EN PARTENARIAT AVEC l'Université de Poitiers

par

AUVRAY Marc-Antoine

_____

Prédictions de temps d'exécution de programmes d'un autopilote de drone

Soutenu le 08/07/2022

_____

JURY

PRÉSIDENT·E :
MEMBRES :

Remerciements

Le travail de recherche présenté dans ce mémoire s'est déroulé au sein de l'équipe-projet Kopernic de l'Institut National de Recherche en Informatique et Automatique, sous la responsabilité de Madame Liliana Cucu-Grosjean.

Tout d'abord, je désire vivement remercier Madame Liliana Cucu-Grosjean tutrice encadrante de mon projet. Déjà lors de mon précédent stage, elle m'a accueilli dans son équipe et procuré tous les moyens nécessaires au bon déroulement de mon travail. Je la remercie sincèrement de l'attention particulière qu'elle a portée sur mes travaux et de la confiance qu'elle m'a renouvelée à la suite de mon stage.

Je voudrais aussi remercier Monsieur Slim Ben Amor pour l'intérêt qu'il a porté à mon travail : il m'a notamment conseillé des ressources et présenté ses enseignements sur la programmation et le *machine learning*.

J'adresse également mes remerciements à Monsieur Kevin Zagalo pour son suivi et le temps qu'il a consacré à mon travail : il m'a notamment proposé une approche sur la planification des travaux et m'a enseigné le fonctionnement de plusieurs algorithmes.

Mes remerciements vont aussi à tous les membres de l'équipe pour leur accueil sympathique. Leur intérêt, leurs compétences respectives et leurs précieux conseils ont grandement contribué à l'enrichissement de mes connaissances et à la diversité de mes travaux réalisés.

Liste des sigles

- IA : Intelligence Artificielle.
- INRIA : Institut National de Recherche en Informatique et Automatique.
- IRIA : Institut de Recherche en Informatique et Automatique.
- PX4 : *Pixhawk* 4.
- SCP : Système Cyber-Physique.
- EPST : Etablissement Public à caractère Scientifique et Technologique.
- WCET : *Worst Case Execution Time* (pire cas de temps d'exécution).
- RITS *: Robotics & Intelligent Transportation Systems*.
- SED : Service d'Exploitation et de Développement.
- HIL : Hardware-In-the-Loop.
- HDR : Habilitation à Diriger des Recherches.
- RSU : Rapport Social Unique.
- COP : Contrat d'Objectifs de Performance.
- BIC : *Bayesian Information Criterion*
- GMM : *Gaussian Mixture Modele*

Glossaire

- Temps-réel : comportement d'un système informatique dont le fonctionnement est assujetti à l'évolution dynamique d'un procédé industriel connecté à lui.
- Système temps-réel : système capable de contrôler (ou piloter) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé.
- Ordonnancement : organisation des traitements, leurs enchainements entre eux, leur planification, la synchronisation des travaux et leur exécution.
- Ordonnanceur : composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur les processeurs d'un ordinateur.
- Autopilote : pilote automatique ou en partie automatique.
- Microcontrôleur : ordinateur présent dans un seul circuit intégré qui est dédié à effectuer une tâche et exécuter une application spécifique.

Sommaire

Introduction

Ce mémoire de première année d'ingénierie en informatique parcours Big Data & Intelligence Artificielle en alternance à l'Institut National de Recherche en Informatique et Automatique est un rapport du projet ayant pour thème le traitement de données issues de l'autopilote d'un drone. Il est divisé en trois parties.

Sa première partie concerne l'environnement de travail du projet : l'organisme d'accueil et l'équipe-projet. L'organisme d'accueil, l'INRIA sera alors introduit à travers ses travaux, ses domaines d'activité, sa structure, etc. Ensuite l'équipe-projet Kopernic sera présentée notamment à travers ses travaux et ses membres.

La deuxième partie est axée autour de la présentation du projet sur la prédiction de programmes de temps d'exécution d'un autopilote de drone. Elle est divisée en trois sous-parties.

La première sur le projet de l'équipe, c'est-à-dire un système d'exploitation temps-réel d'un autopilote de drone avec un état des lieux, les objectifs et les suites attendues.

La deuxième sous-partie concerne un sous-projet au projet global de l'équipe et elle met en place une problématique : comment optimiser l'ordonnancement de programmes de l'autopilote d'un drone selon leurs temps d'exécution et leur consommation d'énergie à l'aide de méthodes d'apprentissage automatique.

La troisième sous-partie est sur l'organisation de ce projet, c'est-à-dire la méthodologie de travail et les ressources mises à disposition.

La dernière partie concerne les travaux faits avec la présentation des algorithmes utilisés, des résultats obtenus et des difficultés et limites. Elle est décrite dans l'ordre chronologique de réalisation.

1. L'INRIA et Kopernic

1.1. L'INRIA

1.1.1. Introduction

Créée en 1967 à Rocquencourt dans les Yvelines en tant qu'IRIA, l'institut devient INRIA (Institut national de recherche en sciences et technologies du numérique) en 1979.

L'institut soutient la diversité des voies de l'innovation à travers notamment la création de startups technologiques et l'édition open source de logiciels. Elle est également fortement engagée dans le transfert de technologie par les partenariats noués avec des entreprises industrielles et aussi par l'intermédiaire de ses sociétés de technologie.

Moteur pour la recherche et l'innovation numérique, l'institut se voit confier différentes missions à travers la France et l'Europe tels que :

- Entreprendre des recherches fondamentales et appliquées ;
- Réaliser des systèmes expérimentaux ;
- Organiser des échanges scientifiques internationaux ;
- Assurer le transfert et la diffusion des connaissances et du savoir-faire ;
- Contribuer à la valorisation des résultats de recherches ;
- Contribuer, notamment par la formation, à des programmes de coopération avec des pays en voie de développement ;
- Effectuer des expertises scientifiques.

L'objectif principal est d'effectuer une recherche de haut niveau et d'en transmettre les résultats aux étudiants, au monde économique et aux partenaires scientifiques et industriels.

L'INRIA intervient dans cinq domaines de recherche, chaque domaine étant subdivisé en plusieurs thèmes :

- Santé, biologie et planète numériques ;
- Mathématiques appliquées, calcul et simulation ;
- Perception, cognition, interaction ;
- Réseaux, Systèmes et services, calcul distribué ;
- Algorithmique, programmation, logiciels et architectures.

1.1.2. Structure

Afin de répondre à de nombreux défis scientifiques et technologiques, près de 4000 chercheurs et ingénieurs explorent à l'Inria des voies nouvelles, que ce soit dans l'interdisciplinarité et en collaboration avec des partenaires industriels. Ce qui fait le cœur de

l'INRIA est son modèle unique d'équipe-projet, en effet 200 équipes-projets dont les équipes RITS, EVA et Kopernic constituent la cellule de base de la recherche au sein de l'institut. Constituée d'une vingtaine de personnes rassemblées autour d'un responsable scientifique, une équipe-projet agrège les talents nécessaires à la conduite d'un projet de recherche et d'innovation évalué tous les quatre ans. Généralement les équipes-projet sont composées de doctorants, post-doctorants et ingénieurs qui travaillent aux côtés de chercheurs confirmés.

L'INRIA s'est développé à travers cinq régions :

- En Ile-de-France, deux unités de recherche une à Paris et une à Saclay, ainsi que le siège qui se trouve à Rocquencourt ;
- En Bretagne, une unité de recherche à Rennes ; en Provence-Alpes-Côte d'Azur, une unité de recherche à Sophia Antipolis ;
- En Lorraine, une unité de recherche à Nancy ;
- En Rhône-Alpes, une unité de recherche près de Grenoble ;
- En Hauts de France, une unité de recherche à Lille.

1.1.3. Présentation économique et juridique

L'INRIA est l'un des 6 EPSTs, établissement public à caractère scientifique et technologique. L'INRIA est à la fois sous la tutelle du ministère de l'éducation nationale, de la recherche et de la technologie et sous la tutelle du ministère de l'économie, des finances et de l'industrie.

Depuis 2020, l'INRIA publie à la place du bilan social annuel, un Rapport Social Unique.

Les informations marquantes qu'on peut tirer du RSU de 2020 [1] :

- Effectifs globaux :
  - 42% des agents à l'INRIA sont rémunérés par d'autres organismes partenaires collaborateurs dont 2/3 sont issus d'universités et écoles françaises ;
  - 4851 agents participent aux missions d'INRIA ;
  - 28% sont des ressortissants étrangers.
- Effectifs rémunérés par l'INRIA :
  - 1/3 sont des femmes ;
  - la moyenne d'âge est de 36 ans ;
  - il y a 600 doctorants dont 269 recrutés cette année, un record à l'INRIA.
- Concours chercheurs :
  - maintenant l'INRA recrute des chercheurs sur contrat à durée indéterminée associé à un service d'enseignement dans un établissement supérieur partenaire d'INRIA ;
  - il y a eu 48 recrutements cette année.
- Le handicap :

-l'INRIA a travaillé cette année sur un projet de convention avec le FIPHFP (Fonds pour l'Insertion des Personnes Handicapées dans la Fonction Publique) ;

-le taux d'emploi d'agents en situation de handicap était de 2,82% cette année, et devrait continuer sa hausse

- La parité :

  -les femmes représentent 22% des scientifiques et 11% des responsables d'équipe de recherche à l'INRIA ;

  -à travers son plan Egalité Professionnelle femmes/hommes, l'INRIA s'est fixé l'objectif d'au moins 30% de femmes parmi les responsables d'équipes de recherche.

1.1.4. Production

A travers le COP (Contrat d'Objectifs de Performance) de 2019-2023 entre l'Etat et l'INRIA [2], l'ambition stratégique de l'INRIA est d'accélérer la construction d'un leadership scientifique, technologique et industriel, dans et par le numérique, de la France dans une dynamique européenne, en s'engageant pleinement dans le développement d'universités de recherche de rang mondial, au cœur d'écosystèmes entrepreneuriaux et industriels dynamisés par le numérique.

1.2. Kopernic

1.2.1. Introduction

KOPERNIC (*Keeping worst case raising appropriate for different cricality*) est une équipe-projet lancée en 2018 par l'Inria dont la responsable (scientifique) est Liliana Cucu-Grosjean.

L'équipe aborde plusieurs sujets de recherches :

- Proposition d'une classification des facteurs de variabilité des temps d'exécution d'un programme par rapport aux fonctionnalités du processeur.
- Définition d'une règle de composition de modèles statistiques basée sur des approches bayésiennes pour des bornes sur les temps d'exécution des programmes.
- Construire des algorithmes d'ordonnancement prenant en compte l'interaction entre différents facteurs de variabilité.
- Prouver des analyses d'ordonnancement pour les algorithmes d'ordonnancement proposés.
- Décider de la planification des programmes communiquant via des réseaux prévisibles et non prévisibles.

L'équipe projet collabore avec des partenaires académiques et industriels en France et à l'étranger. Leurs partenaires industriels appartiennent généralement à l'industrie avionique, l'automobile ou rails.

1.2.2. Membres

- Chercheurs : :
  Prof. Avner Bar-Hen (professeur au CNAM)
  Liliana Cucu-Grosjean (cheffe d'équipe, HDR)
- Membres associés :
  Yasmina Abdeddaïm (professeure à l'ESIEE)
  Slim Ben-Amor (StatInf)
  Adriana Gogonel (Statinf)
  Yves Sorel
- Etudiant post-doctorant :
  Evariste Ntaryamira (Université de Nanterre)
- Etudiants doctorants :
  Chiara Daini
  Ismail Hawaila
  Mohamed Amine Khelassi (Université Gustave Eiffel)
  Kevin Zagalo
  Marwan Wehaiba El Khazen (CIFRE)
- Ingénieurs chercheurs :
  Rihab Bennour
  Hadrien Clarke
  Kossivi Kougblenou (Statinf)
- Apprentis et stagiaires :
  Marc-Antoine Auvray (CNAM)
  Marharyta Tomina
  Olena Verbytska (Université Paris-Cité)

1.2.3. Travaux

L'équipe étudie les propriétés temporelles (temps d'exécution d'un programme ou ordonnançabilité d'un ensemble de programmes communiquant) des composants cyber des SCP.

Un SCP (Système Cyber-Physique) est formé de :

- Composants cyber/informatiques ;
- Composants physiques (qui communiquent entre eux).

Les études de Kopernic sont généralement faites sur des voitures, avions et drones. Un composant cyber est généralement formé de fonctions possédant des niveaux différents de criticité relativement aux propriétés temporelles.

Une solution est appropriée à un niveau de criticité si toutes les fonctions correspondantes remplissent les exigences de ce niveau.

En fonction de leurs fondements mathématiques, les solutions sont :

- Soit non probabilistes quand toutes les propriétés temporelles sont estimées ou bornées par des valeurs numériques
- Soit probabilistes quand au moins une propriété temporelle est estimée ou bornée par une fonction de distribution.

Dans son rapport d'activité de 2021 [3], l'équipe se fixe 3 objectifs :

- L'estimation des temps d'exécution pire cas d'un programme ;
- La décision d'un ordonnancement de tous les programmes s'exécutant dans le même composant cyber, compte tenu d'un budget énergétique ;
- La décision de l'ordonnancement de tous les programmes communiquant par des réseaux prédictibles et non prédictibles, compte tenu d'un budget énergétique

Le programme de recherche de l'équipe est alors organisé autour de 3 grands axes de recherche afin de répondre à ces trois objectifs :

- L'estimation des temps d'exécution pire cas ;
- La construction des repères basés sur des mesures ;
- L'ordonnancement des tâches sur différentes ressources.

2. Prédiction de temps d'exécution de programmes d'un autopilote de drone

2.1. Système d'exploitation temps-réel d'un autopilote de drone

2.1.1. Etat des lieux

Dans le cadre d'un précédent projet, l'équipe Kopernic a étudié l'ensemble des programmes en logiciel libre qui constitue l'autopilote de drones PX4 sur un processeur à un seul cœur. Cet ensemble de programmes a ensuite été transformé afin qu'il respecte des contraintes temps réel.

L'autopilote PX4 est un logiciel libre conçu pour contrôler un large panel de véhicules aériens, terrestres, marins et sous-marins. Il peut s'exécuter sur plusieurs types de matériel, en particulier sur une série de cartes (*Pixhawk*) dont le processeur (*ARM Cortex-M4*) a une architecture interne peu complexe.

L'autopilote gère un ensemble de programmes appelés modules, qui s'exécutent périodiquement et donc produisent périodiquement des données, à partir de données de capteurs sensoriels. Ces modules s'exécutent, aussi, périodiquement en héritant des périodes des modules gérant les capteurs et en se synchronisant avec eux via les accès aux données qu'ils s'échangent. Les modules qui ne gèrent pas les capteurs se synchronisent entre eux de la même manière.

Pour que l'autopilote puisse fonctionner en temps-réel, ces modules ont été transformés en un ensemble de tâches temps-réel dépendantes, possédant des caractéristiques temporelles (périodes d'activation et WCETs) et des contraintes temps-réel (échéances) conformément à la théorie de l'ordonnancement temps-réel.

Les données de temps d'exécution des programmes de l'autopilote sont générées à partir de simulations dites *Hardware in the loop* (HIL). Le principe est de créer un Jumeau numérique du drone, et de simuler un environnement de vol à l'aide d'un logiciel (Gazebo). L'environnement étant simulé, le microcontrôleur *Pixhawk* exécute l'autopilote PX4 comme lors d'un vol réel. Donc pas besoin de faire voler un vrai drone pour avoir des données. Ces simulations temps-réel permettent de tester un code embarqué sans drone.

« *Hardware-in-the-loop* » est traduisible par « matériel dans la boucle ». Cela permet de simuler un environnement à l'aide de l'association de véritables composants connectés à une partie temps-réel simulée. Les tests HIL sont alors réalistes, rentables, flexibles et contrôles par l'utilisateur.

2.1.2. Objectifs

- Le premier objectif est de rendre possible l'exécution des programmes sur plusieurs cœurs.
- Le deuxième objectif est la réalisation de benchmarks, c'est-à-dire un ensemble de programmes, des ensembles de temps d'exécution mesurés pour des processeurs mono-cœurs et multicœurs, et les protocoles de mesures servant à obtenir les temps d'exécution. Les travaux seront mis à disposition en open-source et réutilisables.
- Le troisième objectif est d'optimiser la consommation d'énergie pour les drones.
- Le quatrième objectif est la prédiction des temps d'exécution des programmes temps-réel, qui respectent des contraintes de temps.

2.1.3. Suites attendues

- La validation par d'autres membres de la communauté temps-réel des analyses proposées par l'équipe. Un ensemble de benchmarks open source sera proposé.
- Le développement d'un ordonnancement probabiliste à l'aide de méthodes d'apprentissage statistique.

2.2. Optimisation d'ordonnancement de programmes selon leur temps d'exécution et leur consommation énergétique à l'aide de méthodes d'apprentissage statistique

2.2.1. Etat des lieux

Dans ce projet, l'une des questions à laquelle il faut répondre est l'optimisation de l'ordonnancement des programmes du drone. C'est-à-dire, l'ordre optimal dans lequel les programmes doivent être ordonnancé afin que tous les programmes respectent leurs contraintes de temps.

Les données des capteurs utilisées par les programmes, et leur temps d'exécution doivent être associés.

Avec des données prétraitées, on pourrait chercher les corrélations entre temps d'exécution et autres variables, estimer les temps d'exécutions, ce qui permettrait de développer un algorithme d'ordonnancement utilisant ces estimations.

2.2.2. Pistes de réponses

Nous allons avoir besoin d'un tableau de données horodatées avec des temps d'exécution et des données sensorielles associées. Nous avons 2 tableaux alors que nous avons besoin d'en avoir qu'un seul. Il va falloir être sûr de comment appliquer l'association.

Pour chercher les corrélations entre temps d'exécution et autres données sensorielles, la première étape (qui concerne les travaux de première année) concernera l'application de régressions linéaires sur les données. L'avantage de la régression linéaire est la simplicité de l'algorithme. Quand elle est appliquée, il n'y aura certes pas d'importantes corrélations linéaires mises en avant, mais nous pourrons remarquer les tendances qui en partie permettront de donner des idées quant aux algorithmes plus complexes et adaptés à appliquer par la suite. Nous pourrons également nous apercevoir si les données ont bien été associées.

Enfin, pour faire les meilleures estimations, il faudrait utiliser des algorithmes de *machine learning*.

2.3. Organisation du projet

2.3.1. Méthodologie de travail

Tous les travaux de programmation ont été faits en *Python*. Leur rendu s'est fait grâce à *Jupyter Notebook*.

Kevin, doctorant est celui qui encadre mes travaux.
La méthode de travail est la suivante : j'envoie mes comptes rendus *Jupyter Notebook* à Kevin avec commentaires accompagnant le code et avec un plan expliquant les étapes accomplies, ensuite Kévin me fait un retour sur ce qu'il y a à changer, sur la suite et me réexplique certaines informations du projet de l'équipe, et enfin je fais des retours à ma tutrice encadrante Liliana.

2.3.2. Ressources

Une partie des connaissances nécessaires à la compréhension et l'utilisation des différents travaux appliqués dans ce projet m'a été instruite par les membres de l'équipe notamment Kevin Zagalo et Slim Ben Amor.

Ils m'ont par ailleurs aiguillé vers différents contenus sur Internet. Ainsi, le site sickit-learn.org m'a permis d'apprendre les algorithmes utilisés dans les programmes.

Pour l'autopilote PX4, un ensemble de fichiers (privé) comportant les explications ont été fournies par l'équipe. En plus de cette aide, Kevin me recontextualise le projet de l'équipe et ma part dans le projet avec les informations nécessaires à la compréhension de ceux-ci.

Pour le code Python, plusieurs librairies ont du être installées et leurs fonctions comprises pour être utilisées, et cela à l'aide de documentation :

- *Pandas* [4] : elle propose la manipulation de données et l'analyse des données, notamment à travers des structures de données et des opérations de manipulation sur des tableaux numériques ou séries temporelles. Elle permet en partie de créer des *dataframe*, d'en concaténer.
- *Numpy* [5]: elle propose plus efficacement d'effectuer des calculs logiques et mathématiques sur des tableaux, des listes et des matrices. Elle permet notamment de trouver les valeurs uniques d'une liste, de transformer un objet en tableau (*array*), de trouver les valeurs manquantes d'un tableau ou encore de trier une liste dans un certain ordre.
- *Matplotlib* [6] : elle permet de créer des graphiques plus élaborés, en précisant par exemple la taille, une légende, un label d'abscisses, un label d'ordonnées, un titre, ou encore de créer des figures avec plusieurs graphiques juxtaposés, d'afficher ces graphiques, de créer des histogrammes, de créer des nuages de points, d'ajouter plusieurs éléments dans un graphique, etc.
- *Seaborn* [7] : elle est basée sur *Matplotlib* mais permet l'utilisation d'objets de type Pandas. Elle est plus simple et parfois plus complète que *Matplotlib*. Elle permet notamment de créer des matrices de corrélation.
- *Scipy* [8]: elle utilise des objets *Numpy* et y ajoute des fonctionnalités supplémentaires et peut permettre par exemple de créer une variable distribuée normalement.

- *Statistics* [9] : comme son nom l'indique elle permet de calculer des statistiques assez triviales de manière simple (comme un écart-type).
- *Itertools* [10] : elle permet de faire des boucles plus élaborées, comme faire une boucle sur 2 listes de même taille.
- *SKLearn* [11] : elle propose de l'utilisation d'apprentissage automatique sous des formes simples à complexes. Les agents de l'INRIA font partie des contributeurs à son développement. Elle peut permettre de normaliser des données, d'utiliser des mélanges gaussiens, d'appliquer des modèles de régression linéaire, d'utiliser l'algorithme des *KMeans*.

2.3.3. Algorithmes utilisés

- *MinMaxScaler* :

  Cet algorithme permet de redimensionner une variable distribuée dans un certain intervalle dans un nouvel intervalle, souvent entre 0 et 1. Il normalise donc les données et il peut alors être plus facile de visualiser, traiter et comparer plusieurs variables quand elles ont été normalisées.

  Sa formule est assez simple :

  (Pour une normalisation entre 0 et 1, avec x la variable d'origine et x' de nouvelle dimension)

$$x_i' = \frac{\mathrm{x_i} - \min(\mathrm{x})}{\max(\mathrm{x}) - \min(\mathrm{x})}$$

- Régression linéaire :

  La régression linéaire multiple permet de prédire une variable en fonction d'autres variables. Chaque variable est alors multipliée par un coefficient qu'on retrouve dans une fonction linéaire :

  (Avec y' la prédiction de la variable à prédire, les x les variables prédicatrices et les β les coefficients)

$$y' = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

  Si on fait une régression linéaire simple (non-multiple), soit la prédiction d'une variable grâce à une seule autre variable, pour chaque couple de variables d'un tableau, on obtient les coefficients de corrélation. Plus le coefficient est absolument élevé, plus la corrélation entre les 2 variables est forte.

- Mélange gaussien :

  Un mélange gaussien est une variable composée de plusieurs variables gaussiennes Pour partitionner cette variable, l'algorithme espérance-maximisation. Par exemple, si

une variable a l'air d'être divisible en 2 variables gaussiennes, on commence par initialiser 2 lois normales avec des valeurs aléatoires, ensuite pour chaque x, on calcule sa probabilité qu'il suive la première loi puis la deuxième, on calcule les poids pour chaque probabilité et enfin on calcule des nouveaux paramètres de lois normales en ajustant à partir des poids, puis on relance les étapes. Le Critère d'information bayésien (BIC) permet d'approximer le calcul de la vraisemblance de données conditionnellement à un modèle donné. Entre plusieurs modèles, celui sélectionné sera celui avec le plus petit BIC.

Sa formule est la suivante :

(avec L la vraisemblance du modèle, k le nombre de paramètres et N le nombre d'observations dans l'échantillon)

$$-2ln(L) + k * ln(N)$$

- *K-Means* :

  L'algorithme des *K-Means* (ou partitionnement en K-moyennes) est un algorithme de classification non supervisé. Il consiste à regrouper des données en fonction des distances entre les données et des centroïdes donnés.

3. Travaux réalisés

3.1. Pré-traitement et description des données

Cette partie concerne mes premiers travaux dont le code est en première partie des annexes.

Les premiers travaux étaient centrés sur le traitement de données issues de deux fichiers créés lors d'une simulation de vol de drone. Le premier fichier (« ulog_scheduler_0.csv ») a une variable de temps d'exécutions et une variable d'horodatage, ainsi qu'une autre variable « event » qui indique à quel module correspond le temps d'exécution associé et dont on verra l'utilité plus tard. Le deuxième fichier (« ulog_sensor_combined_0.csv ») a une variable d'horodatage, 6 variables de données issues de capteurs sensoriels du drone (accéléromètre et gyromètre), et d'autres variables qui vont être supprimées car inutiles. Lors de la dernière simulation, le fichier de temps d'exécution avait environ 70 000 données et celui des variables sensorielles 10 000.

Pour associer les données et créer un tableau unique, 5 étapes ont été opérées :

- Nettoyage respectif des 2 fichiers (avec tri des variables) ;
- Concaténation des 2 fichiers (en laissant des données manquantes) avec tri du tableau dans l'ordre croissant des horodatages ;

- Remplacement de chaque donnée manquante de ligne issue du fichier de temps d'exécution par la donnée non manquante suivante ;
- Suppression des lignes avec données manquantes ;
- Suppression de la variable d'horodatage.

Les données des 6 variables sensorielles ont ensuite été normalisées (entre 0 et 1) à l'aide de l'algorithme *MinMaxScaler*.

Pour se donner une idée de la distribution de chacune des 6 variables, voici un histogramme par variable :

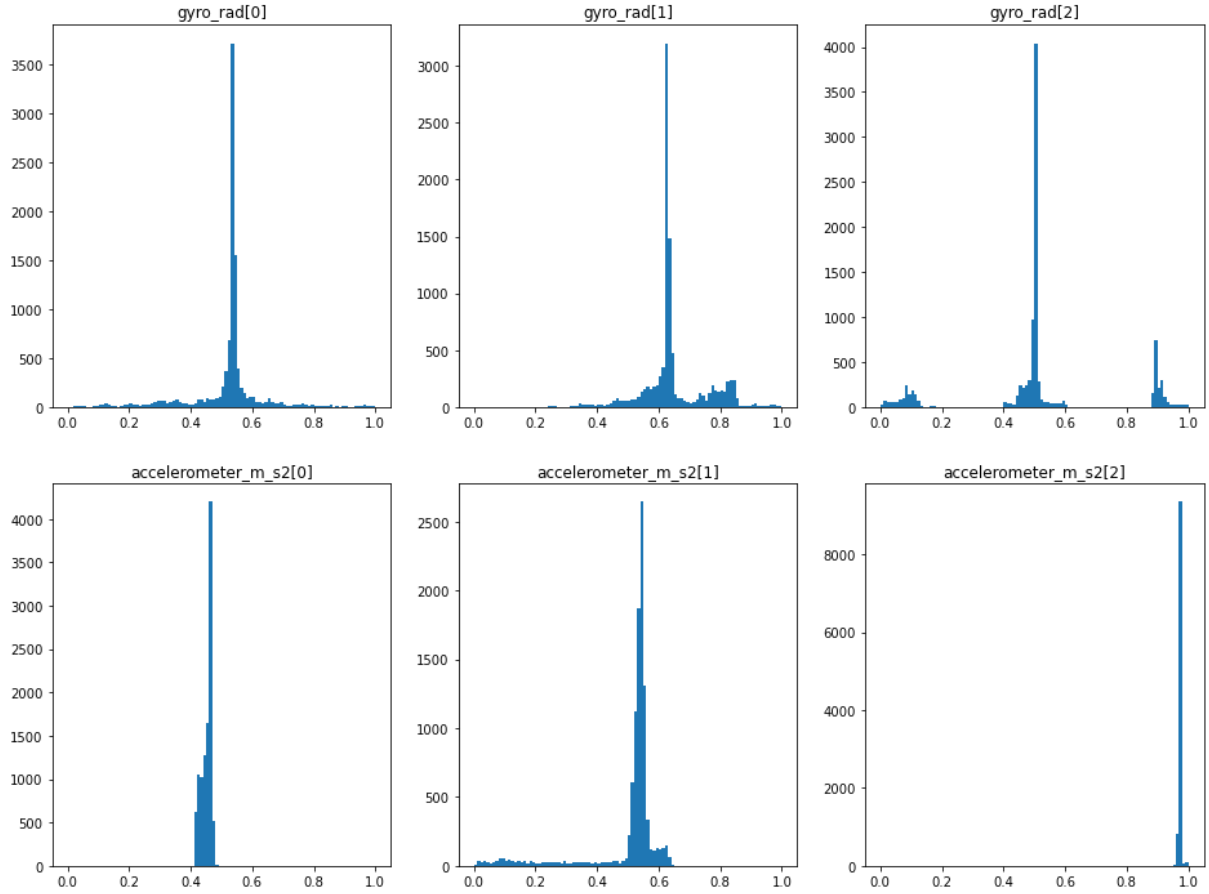


Figure 1 Histogramme de chaque variable normalisée du fichier ulog_sensor_combined_0.csv

On voit que les valeurs normalisées de gyro_rad[0], gyro_rad[2], accelerometer_m_s2[0] et accelerometer_m_s2[0] sont distribuées autour d'environ 0,5, celles de gyro_rad[1] autour de 0,6 et celles de accelerometer_m_s2[2] autour de 0,95.

Enfin, nous faisons aussi un histogramme des temps d'exécution afin de mettre en évidence les tendances.

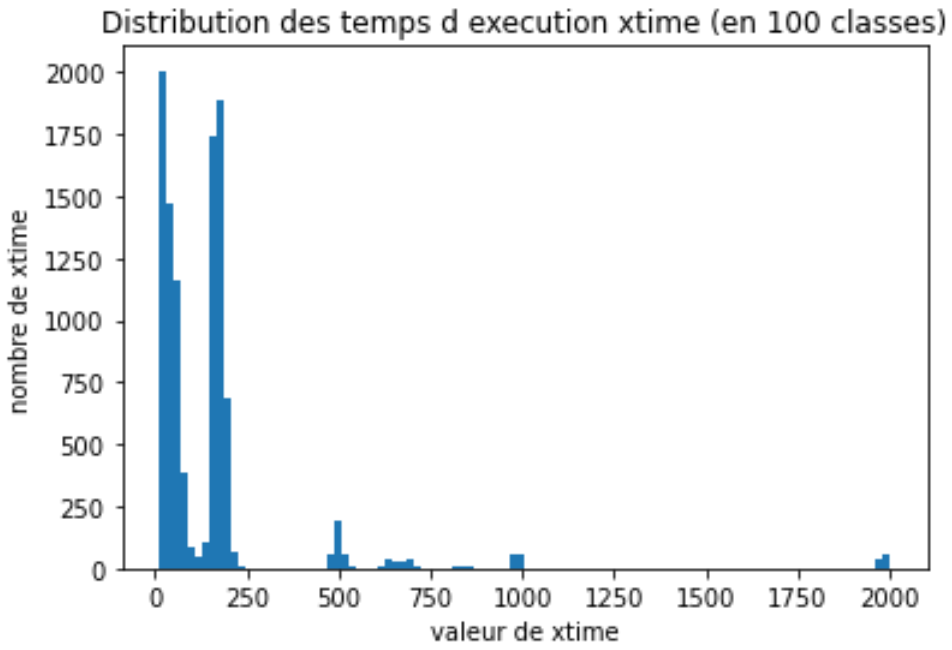


Figure 2 Histogramme des temps d'exécution (en microsecondes)

Les temps d'exécution vont de 10 à 2002 microsecondes.

Nous allons faire des régressions linéaires avec plusieurs variables pour essayer de voir si un modèle linéaire est possible avec ces données.

3.2. Premières régressions linéaires

Cette partie concerne la suite des premiers travaux dont le code est en deuxième partie des annexes.

Nous avons fait une régression linéaire des sur les temps d'exécution en fonction des 6 variables sensorielles issues du fichier ulog_sensor_combined_0.csv .

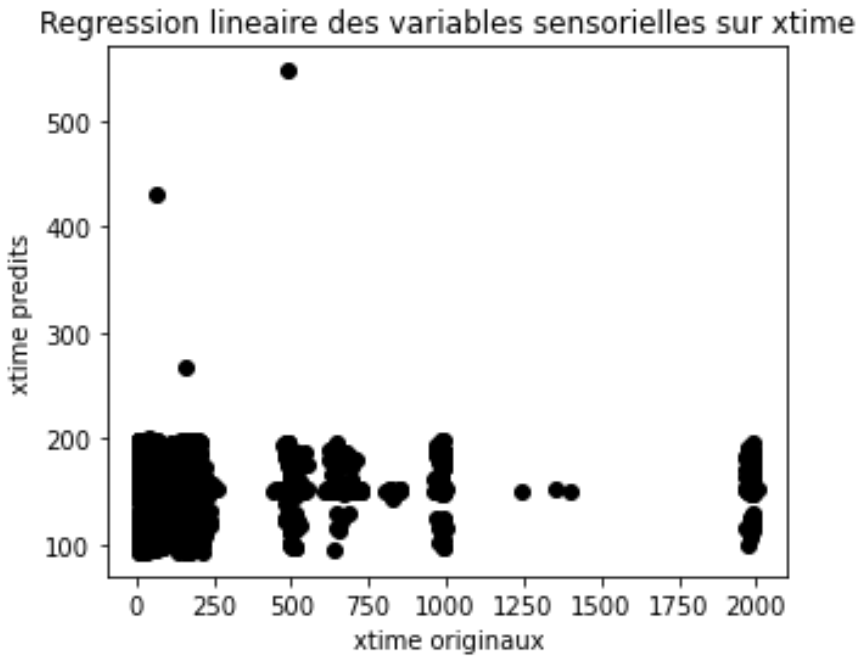


Figure 3 Nuage de points de régression linéaire des temps d'exécution selon les variables de ulog_sensor_combined_0.csv

Pour la lecture du graphique, on voit que par exemple les vrais temps d'exécutions entre 0 et 250 ont pour la plupart été estimés entre 100 et 200, puis un à environ 275 et un autre à 425. Ensuite, tous les temps d'exécution au-dessus de 250 ont été estimés en-dessous de 200. Les prédictions ne dépassent jamais 600 alors que les données vont jusqu'à plus de 2000.

Pour essayer de trouver si certaines variables sont plus ou moins influentes (linéairement) sur les temps d'exécution, nous allons calculer les corrélations linéaires pour chaque couple.

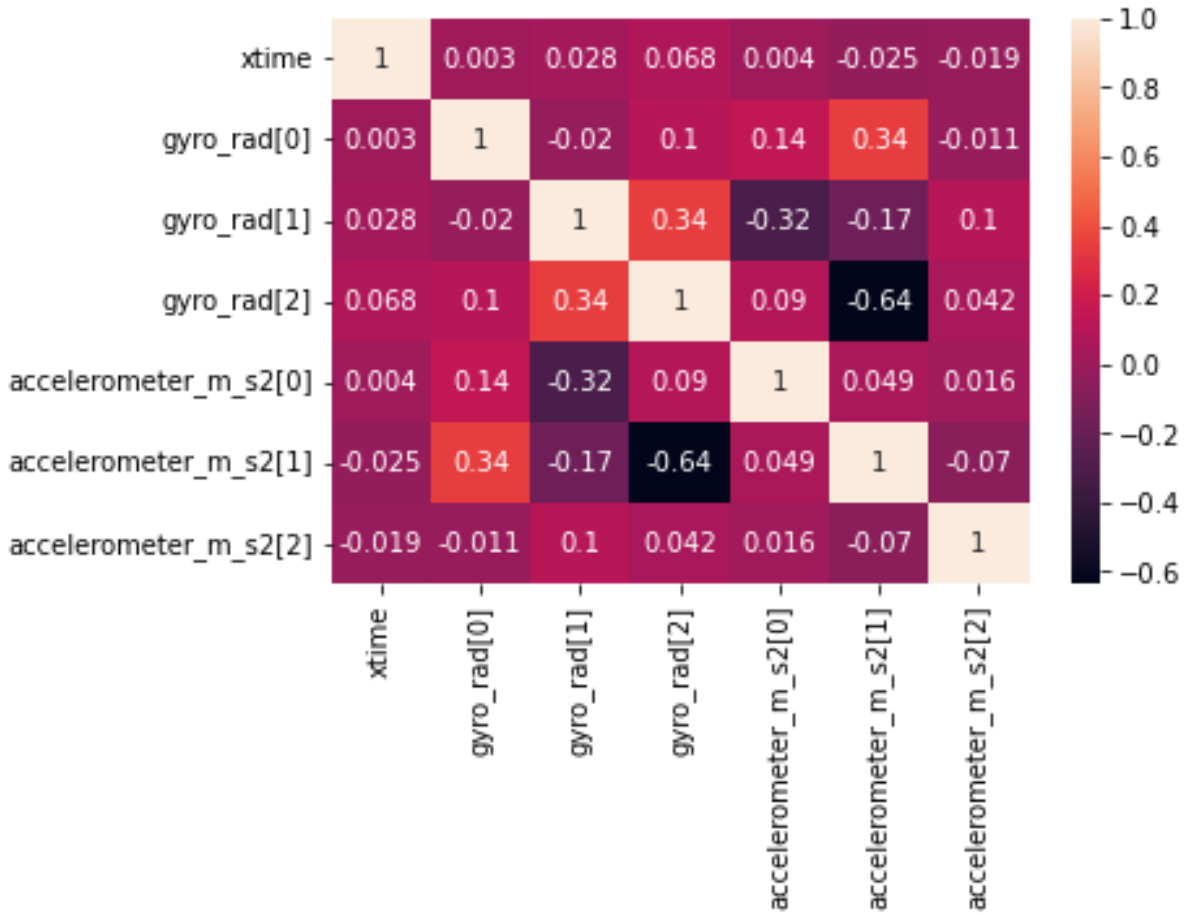


Figure 4 Matrice de corrélations de toutes les variables

Nous allons réappliquer une régression linéaire en fonction des 4 variables dont les coefficients de corrélation avec temps d'exécution sont les plus éloignés de 0.

Les coefficients de corrélation avec temps d'exécution de gyro_rad[1], gyro_rad[2], accelerometer_m_s2[0], accelerometer_m_s2[1] sont respectivement de : (arrondis) 0.028, 0.068, -0.019 et -0.025.

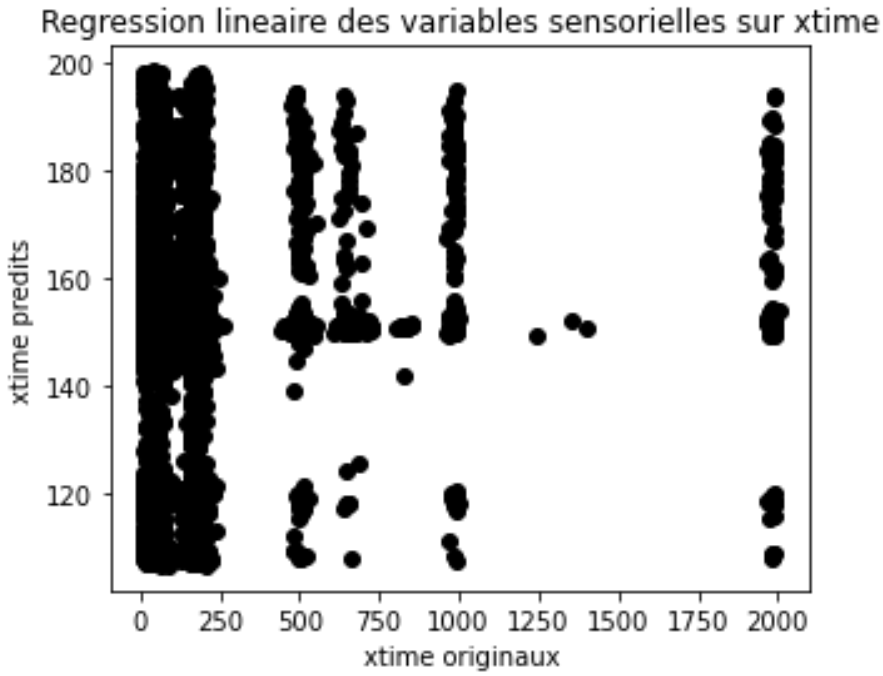


Figure 5 Nuage de points de régression linéaire des temps d'exécution selon les 4 variables aux plus forts coefficients de corrélation avec les temps d'exécution

Nous faisons ensuite une régression linéaire avec les 2 variables aux plus gros coefficients absolus de corrélation.

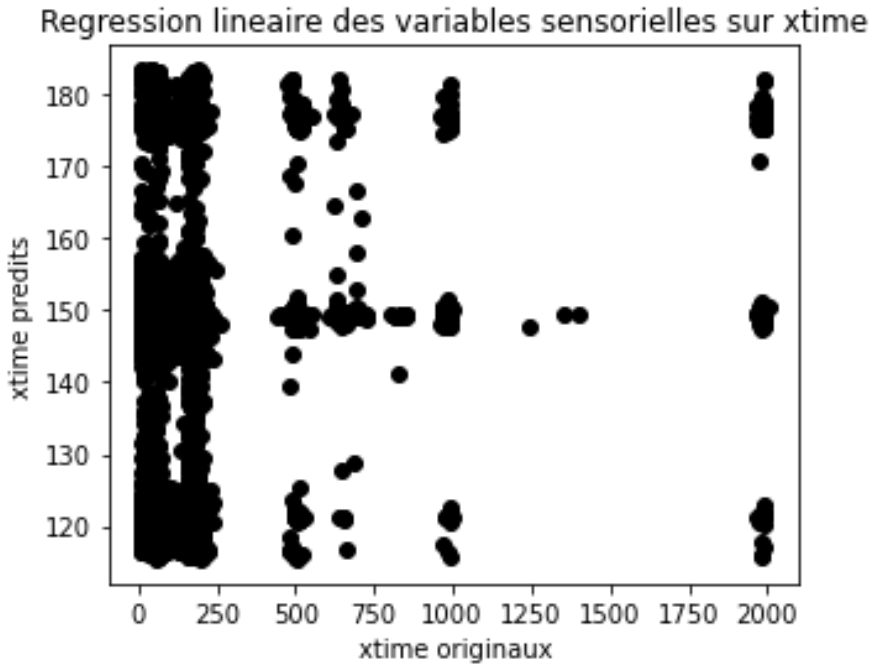


Figure 6 Nuage de points de régression linéaire des temps d'exécution selon les 2 variables aux plus forts coefficients de corrélation avec les temps d'exécution

Dans les 2 cas, les estimations sont toujours trop peu correctes, elles vont jusqu'à 180 ou 200 microsecondes alors que les données originelles vont jusqu'à 2002.

Nous pensons qu'il y a des classes à l'intérieur de la variable de temps d'exécution alors nous allons par la suite essayer des classifications non-supervisées, puis appliquer une régression linéaire par classe.

3.3. Régressions linéaires optimisées avec classification non-supervisée

Cette partie concerne la suite des travaux dont le code est en troisième partie des annexes.

Maintenant, nous allons essayer de trouver plusieurs distributions au sein des données et appliquer une régression linéaire par distribution. Nous pensons grâce aux résultats précédents, notamment l'histogramme des temps d'exécution, que le mélange est gaussien c'est-à-dire qu'on peut découper la distribution en plusieurs distributions chacune suivant une loi normale. Pour mettre en évidence un mélange gaussien on peut utiliser l'algorithme espérance-maximisation. Cet algorithme va chercher les paramètres $N(\mu,\sigma)$ des différentes distributions au sein de la distribution des temps d'exécution.

Nous avons fait 6 graphiques à courbes. Chaque graphique représente les courbes des distributions normales de mélanges gaussiens. La distribution totale a été découpée en n (de 2 à 7) sous-distributions, chacune suivant une loi normale représentée par une courbe. Les paramètres (moyenne, écart-type) de chaque sous-distribution sont indiqués en légende.

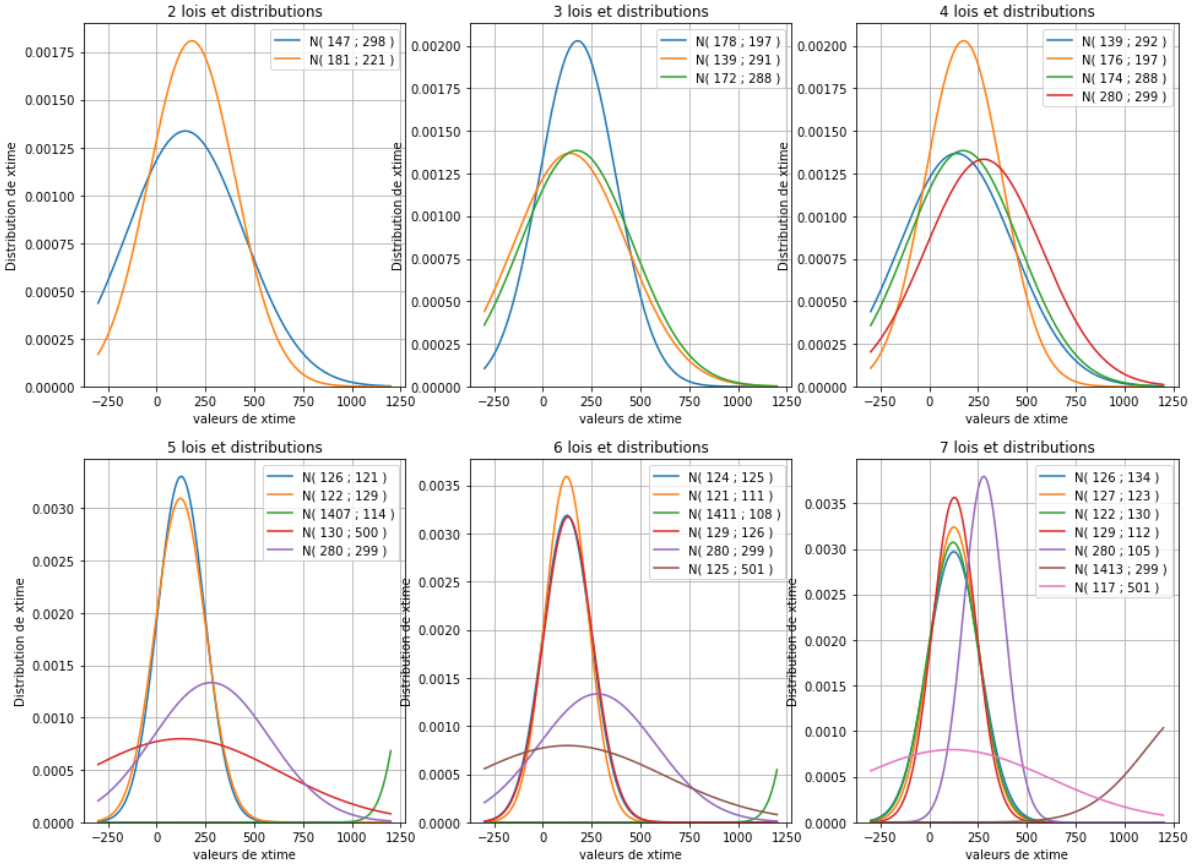


Figure 7 Représentations graphiques de modèles de mélange gaussien pour chaque nombre n (de 2 à 7) de classes choisi

Pour la lecture : par exemple, pour le premier graphique, la variable a été divisée en 2 parties, la première suit une loi normale N(μ=147,σ=298) et la deuxième la loi N(μ=181, σ=221).

Ici nous avons choisi les nombres de clusters (de 2 à 7) mais on ne sait pas encore lequel est idéal. Pour le déterminer, il y a l'indicateur BIC (*Bayesian Information Criterion*) calculé à l'aide d'une formule. Plus il est bas plus le nombre de clusters est bon. Nous allons regarder les valeurs du BIC mais on ne va pas dépasser 20 distributions.

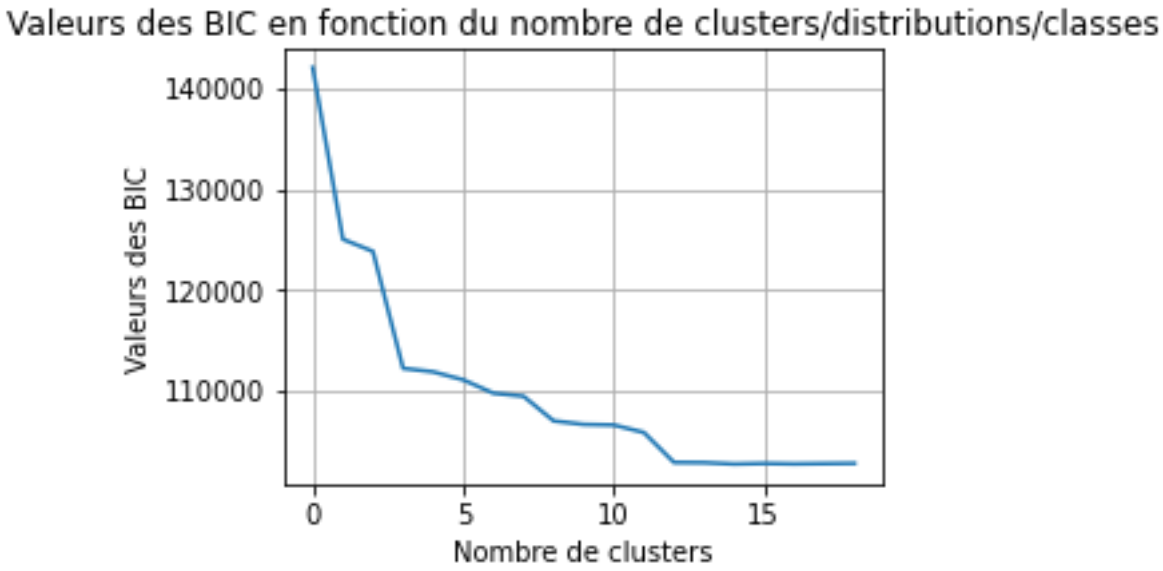


Figure 8 Courbe sur l'évolution de la valeur du BIC en fonction du nombre de clusters choisi

Le BIC continue de diminuer quand on augmente le nombre de clusters. Néanmoins, graphiquement il se stabilise vers 6.

Nous avons donc choisi le nombre de 6 clusters grâce au BIC. Pour chacune des sous-distributions issues de la distribution, l'algorithme GMM va leur trouver une loi normale suivie.

Graphiquement, cela correspond aux courbes suivantes.

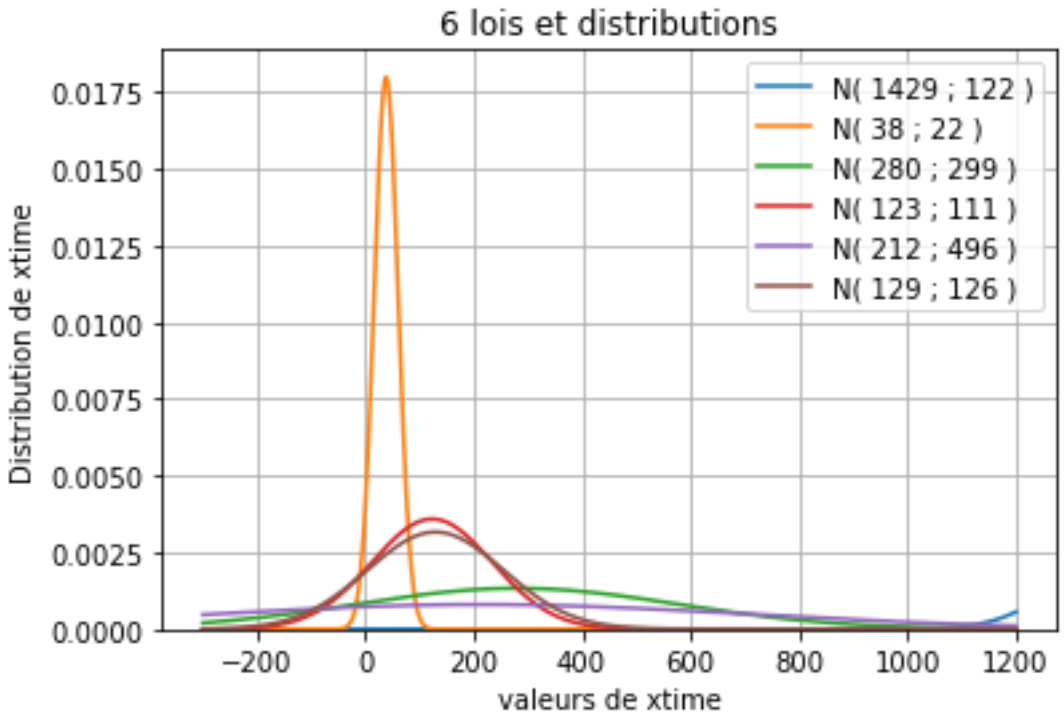


Figure 9 Représentations graphiques de modèles de mélange gaussien pour 6 classes

Les 6 classes ont l'air de pouvoir être regroupées en 2 classes. Nous allons alors faire ce regroupement et appliquer une régression linéaire sur chacune de ces 2 nouvelles classes, nous allons aussi en faire pour le découpage en 6 classes.

Régression pour chacune des 2 classes :

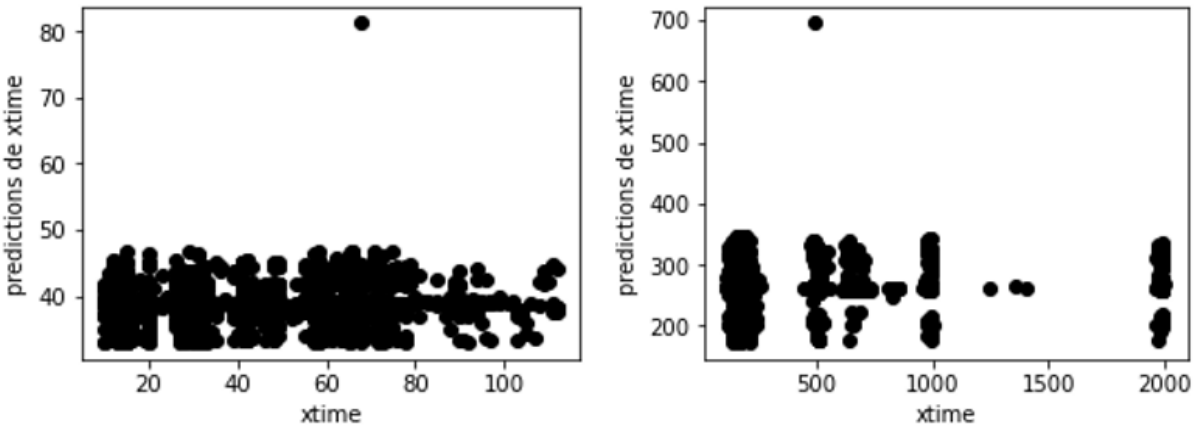


Figure 10 Régression linéaire des temps d'exécution selon les autres variables pour chacune des 2 classes

Dans la première classe, les temps d’exécutions originaux allaient de 10 à environ 150 et ont été prédites jusqu’à 80, pour la plupart entre 10 et 50. Dans la deuxième classe, les valeurs

étaient comprises entre environ 150 et 2002 et ont été prédites entre 150 et 700. C'est mieux que pour la première régression linéaire mais les prédictions sont toujours trop erronées.

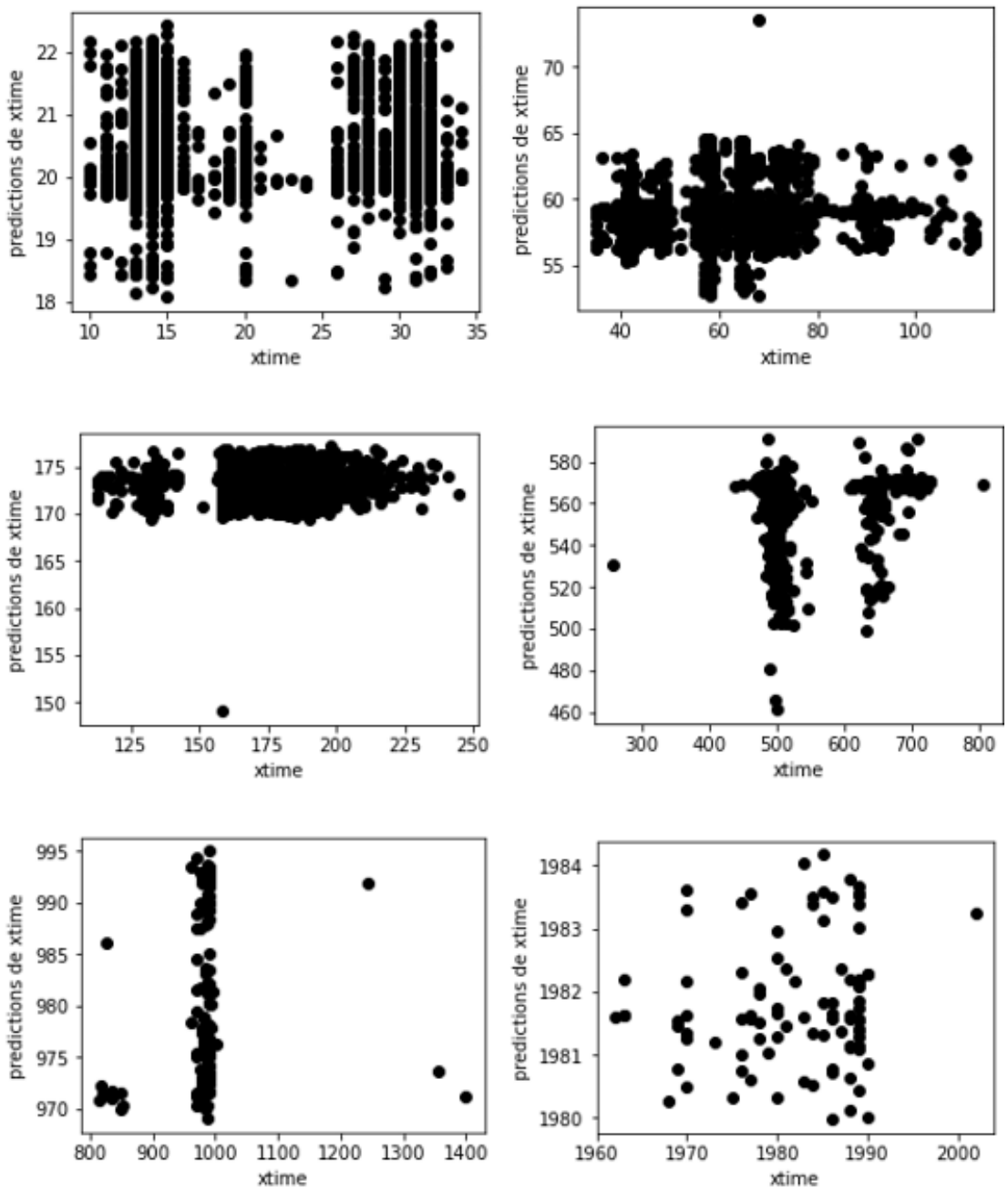


Figure 11 Régression linéaire des temps d'exécution selon les autres variables pour chacune des 6 classes

Les prédictions ont l'air plus correctes. La différence entre chaque centre d'intervalle des temps d'exécution prédits et le centre de l'intervalle de ses temps d'exécution originaux entre d'environ 40 maximum sauf pour les données du 5e graphique qui vont jusqu'à une différence de 300 microsecondes. Les prédictions se sont améliorées mais ne sont pas parfaites. Surtout, lorsque l'on applique des régressions linéaires, nous sommes censés pouvoir observer une

tendance à travers un regroupement des points ayant l'air de suivre une courbe. Ce n'est pas le cas ici, les données sont parfois regroupées très horizontalement, ou très verticalement ou de manière très dispersée. Nous allons voir si d'autres données peuvent apporter plus d'information.

3.4. Généralisation du code

Cette partie concerne les derniers travaux à l'heure actuelle dont le code n'est pas en annexe car trop long.

Nous avons maintenant reçu les données issues de 3 nouveaux fichiers qui ont eux aussi des données sensorielles :

- ulog_vehicle_magnetometer_0 :
  Il a 3 variables de capteurs sensorielles de type magnétomètre (magnetometer_ga[0] , magnetometer_ga[1] , magnetometer_ga[2]) et une variable d'horodatage (timestamp). Les autres variables vont être supprimées car inutiles (par exemple 1 seule valeur dans la variable).
- ulog_vehicle_gps_position_0 :
  Il a 10 variables de capteurs sensorielles qui concernent des données de position (lat, lon, alt, alt_ellipsoid, vel_m_s, vel_n_m_s, vel_e_m_s, vel_d_m_s, cog_rad). Il a également des horodatages, et d'autres variables qui vont être supprimées car inutiles.
- ulog_vehicle_air_data_0 :
  Il a 4 variables sensorielles (baro_alt_meter , baro_temp_celcius , baro_pressure_pa , rho), une variable d'horodatage et 2 autres variables qui vont être supprimées car inutiles.

On part du principe qu'en plus d'un fichier avec des temps d'exécution, on peut recevoir avec un fichier de n'importe quel autre type de données. Nous allons alors essayer d'universaliser le code et ses fonctions pour qu'il puisse faire des régressions linéaires et des classifications (non-supervisées) avec tous les fichiers mis à disposition.

Pour utiliser les nouvelles données, il a fallu déjà passer par plusieurs étapes avec chacune une fonction pour avoir des données nettoyées :

- Conversion en un *Dataframe* grâce au package pandas ;
- Suppression de toutes les variables autres que « timestamp » qui avaient le mot « timestamp » dans le nom ;
- Suppression des variables avec certains mots (« integral ») ;
- Suppression des variables avec moins de 3 valeurs différentes dedans ;

- Normalisation des données (sauf temps d'exécution, numéro de module et horodatage).

Ensuite, en fonction des différents besoins, on concatènera certains tableaux puis on supprimera les variables d'horodatage et d'événement.

Voici les distributions de variable sensorielle de chacun des 3 nouveaux fichiers :

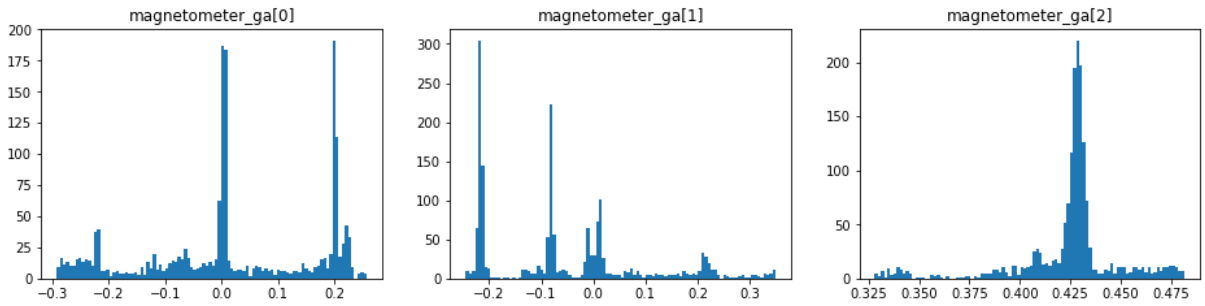


Figure 12 Histogramme de chaque variable de ulog_vehicle_magnetometer_0.csv normalisée

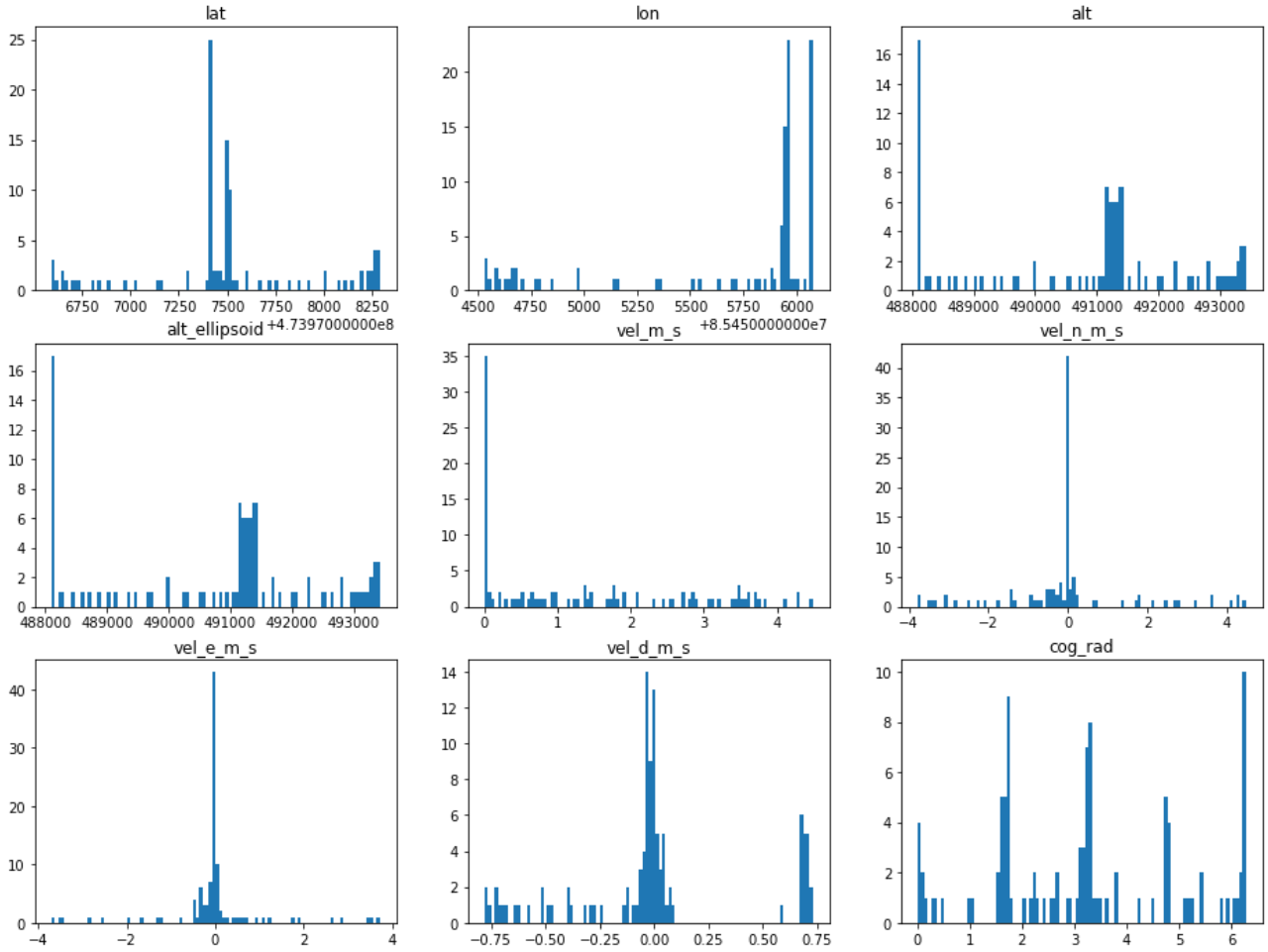


Figure 13 Histogramme de chaque variable de ulog_vehicle_gps_position_0.csv normalisée

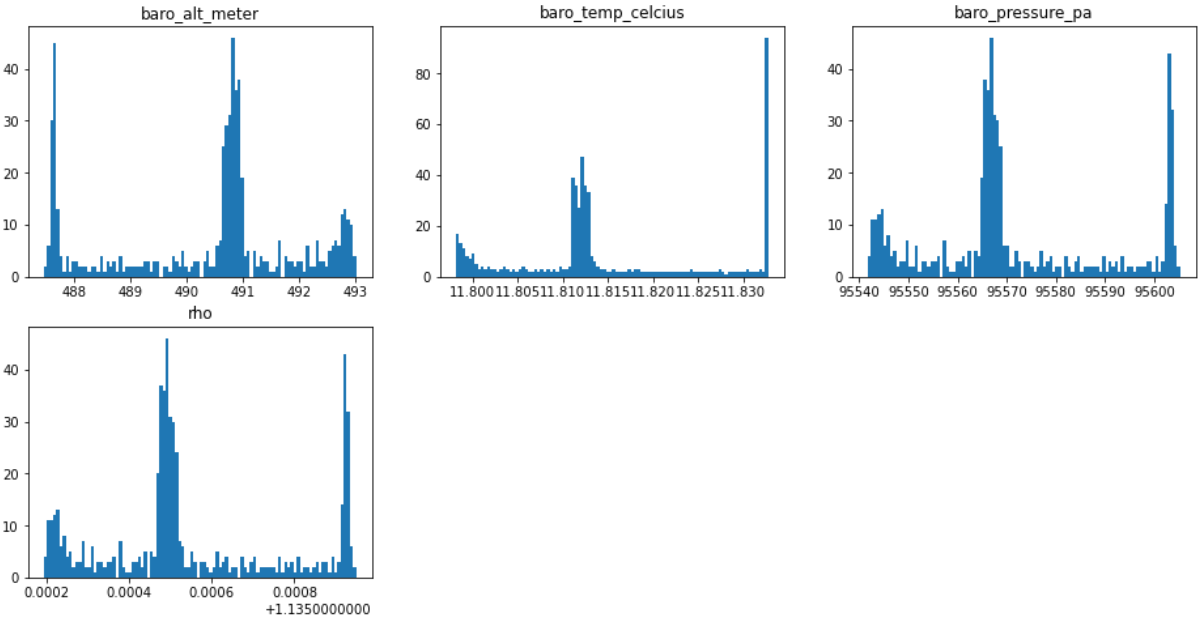


Figure 14 Histogramme de chaque variable de ulog_vehicle_air_data_0.csv normalisée

La variable « event » présente dans ulog_scheduler_0 apporte une information : son dernier chiffre correspond à un des 9 modules qui est lancé et dont le temps d'exécution est donné.

Nous allons donc créer 9 tableaux (de 0 à 8) dans un objet de type dictionnaire.

Voici les distributions des temps d'exécution pour chaque module :

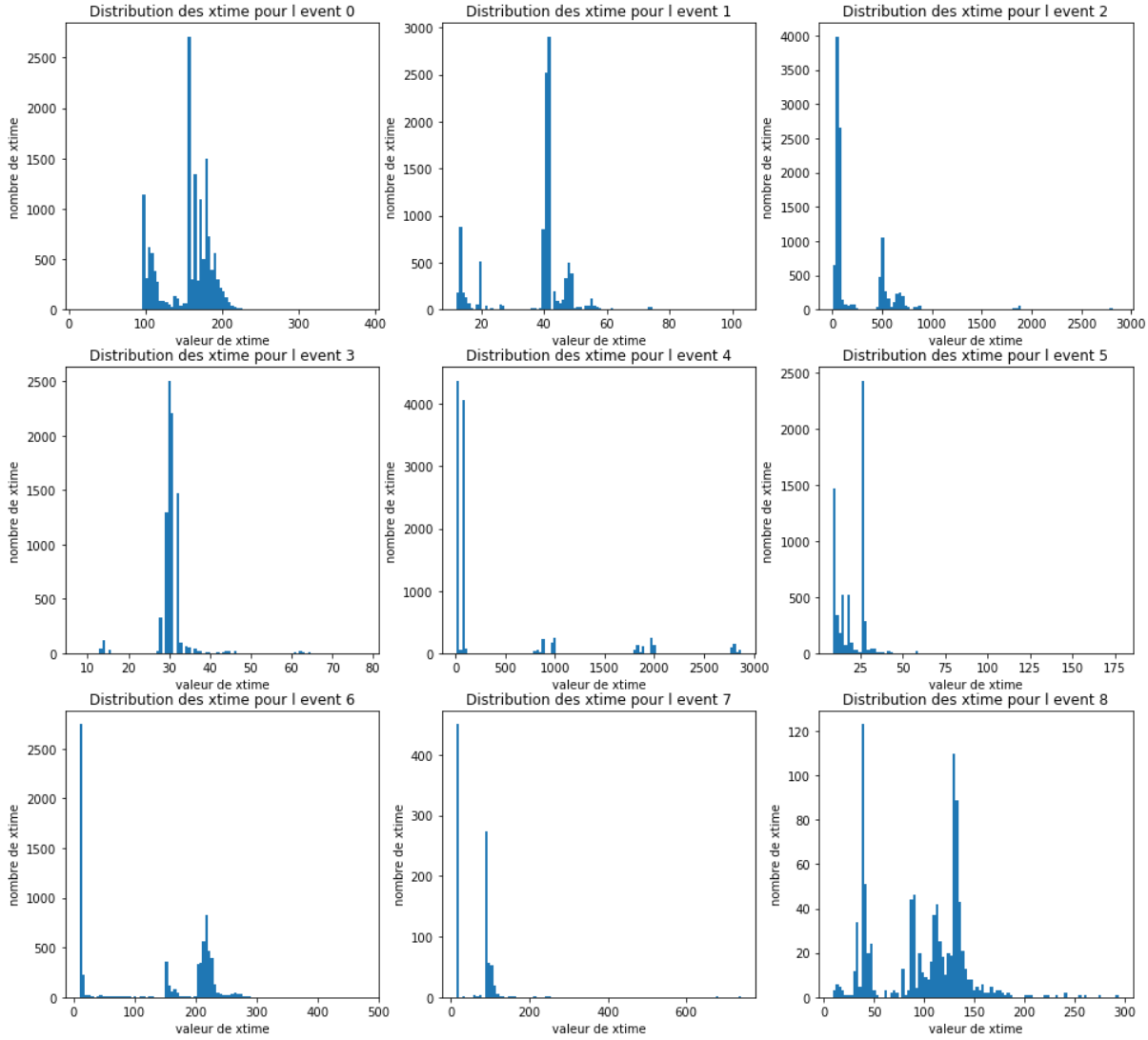


Figure 15 Histogramme des temps d'exécution de chaque module

Nous allons nous intéresser en particulier aux données dont « event » finit par 3 : ce sont celles dont les temps d'exécution sont les plus corrélés avec les données des capteurs et celles qui correspondent au module numéro 3.

En calculant le BIC, le nombre de classes est le plus pertinent à 7 pour le module numéro 3.

Figure 16 Courbe sur l'évolution de la valeur du BIC en fonction du nombre de clusters choisi

Quand nous faisons une régression linéaire pour chaque classe, nous obtenons les graphiques suivants :

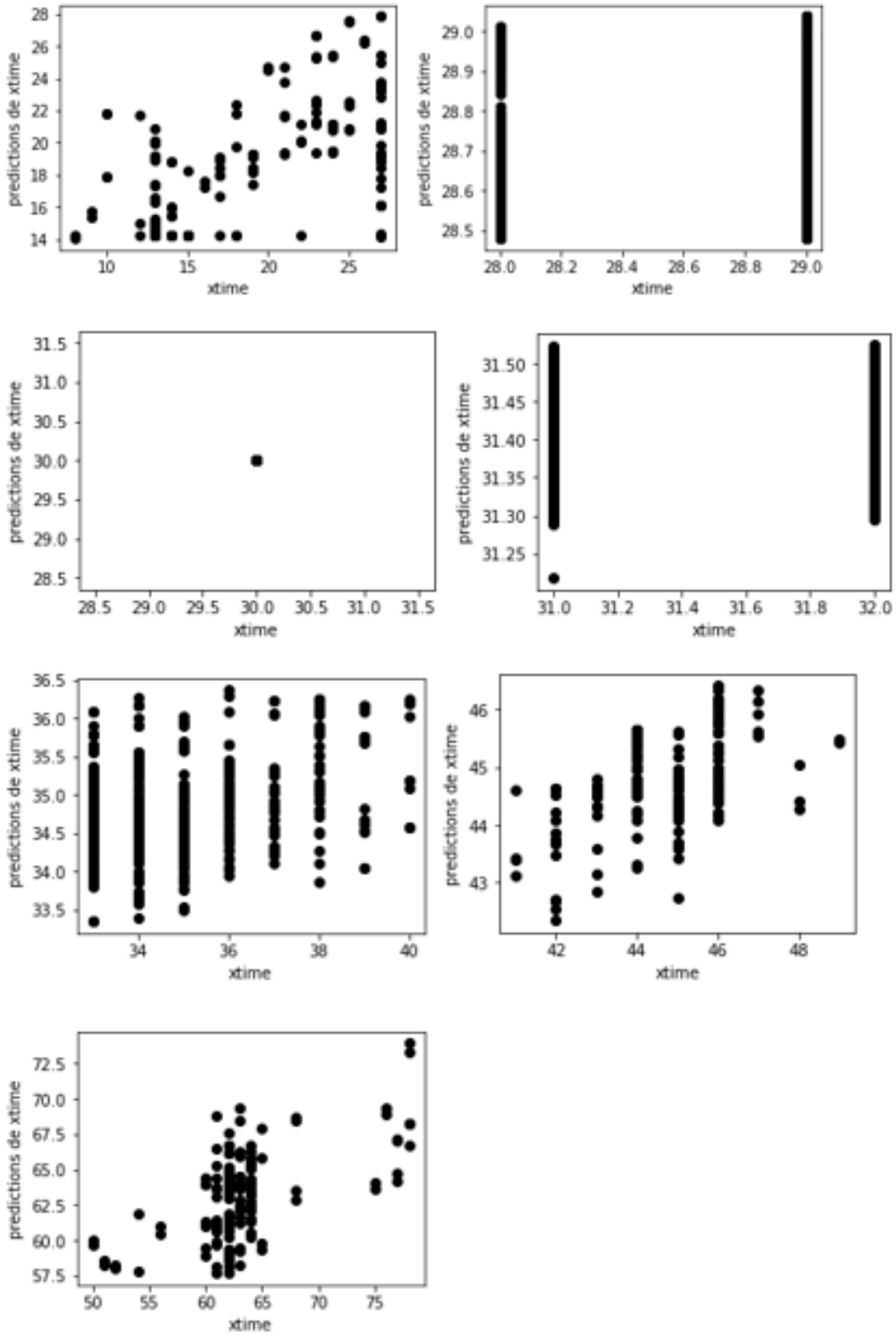


Figure 17 Régression linéaire des temps d'exécution selon les autres variables pour chacune des 7 classes

Les prédictions sont assez proches des temps d'exécution originaux. Nous pouvons même voir dans les 3 derniers graphiques que les points ont l'air de suivre une courbe de fonction

linéaire. Cependant, la corrélation semble assez faible et ne se voit pas dans les autres graphiques.

Toujours avec les données du module numéro 3, une matrice de corrélation va montrer si certaines variables influencent temps d'exécution.

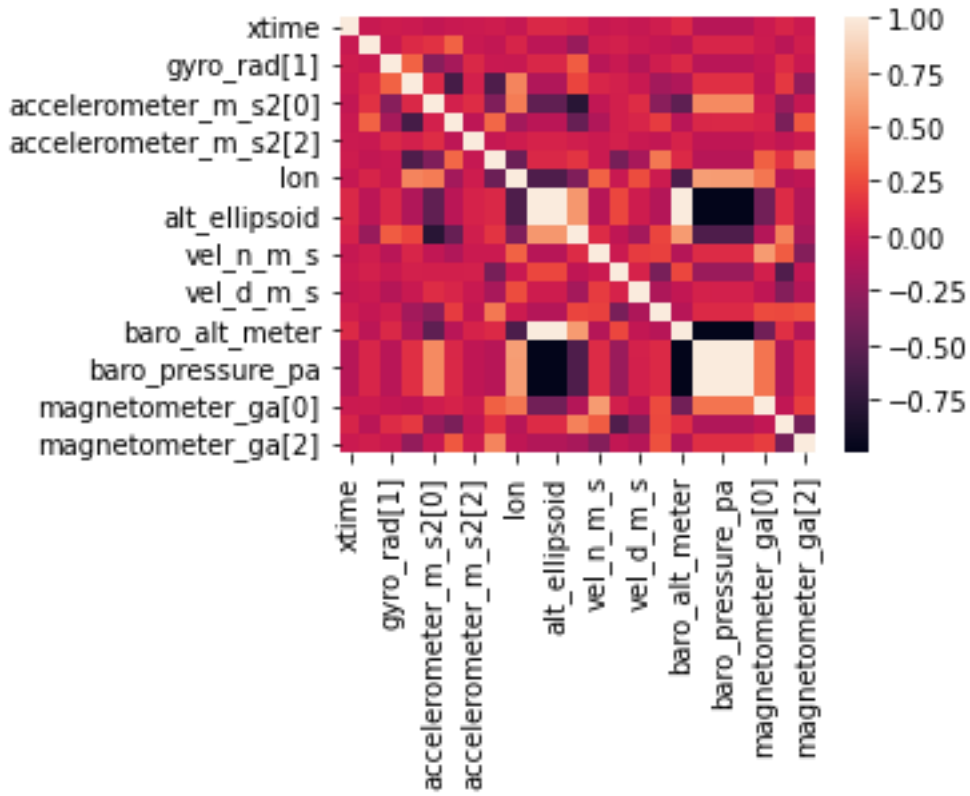


Figure 18 Matrice des corrélations de toutes les variables

Nous voyons que les variables issues de ulog_vehicle_air_data_0 sont les plus corrélées au temps d'exécution.

Nous avons coupé les temps d'exécution en 7 classes grâce au même algorithme utilisé précédemment. Nous allons vérifier que le découpage se fait bien.

Déjà pour des données découpées en 2 parties : (rouge = classe 0, orange = classe 1).

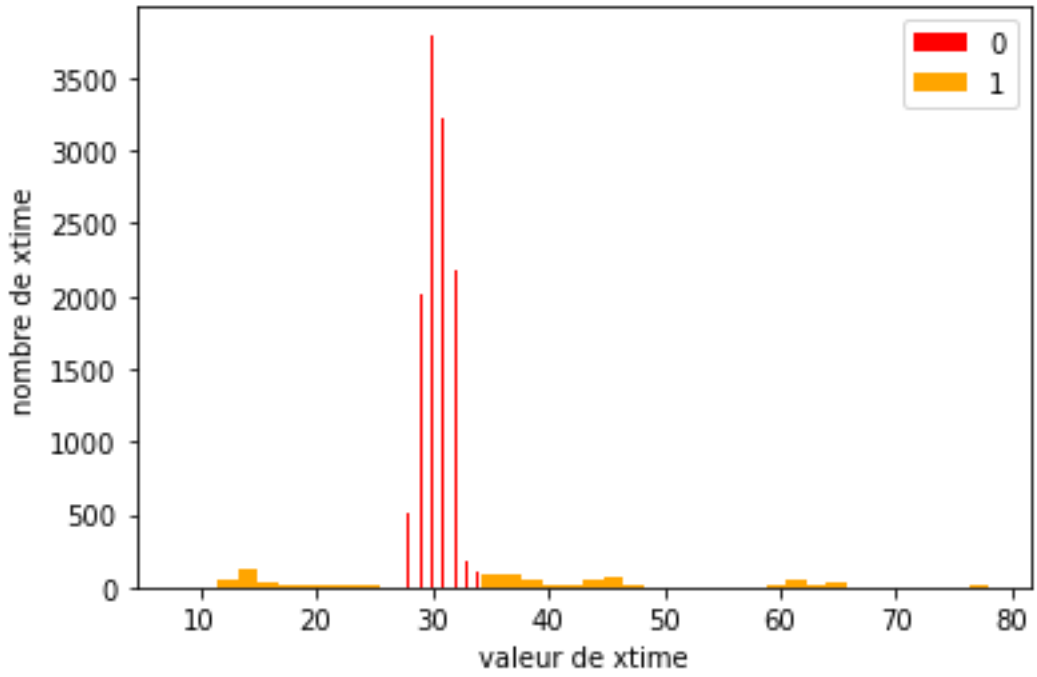


Figure 19 Histogramme des temps d'exécution

On devrait voir apparaître chaque classe l'une après l'autre. Ce n'est pas le cas, il faudra faire une autre classification.

3.5. Difficultés et limites

L'une des premières difficultés a été sur l'alignement des données, c'est-à-dire l'association entre temps d'exécution et variables sensorielles. Nous ne sommes pas sûrs à 100% qu'il soit bon.

Une des limites du projets est que les données extérieures sont simulées. Elles peuvent alors être pas suffisamment réalistes.

Aussi, nous ne sommes pas sûrs que la prédiction de temps d'exécution soit intégrable dans un cas réel de vol de drone que ce soit en termes de complexité et de temps.

Conclusion

Nous avons remarqué que les données étaient rarement ou peu corrélées linéairement. Néanmoins, plus les étapes ont avancé, plus des corrélations linéaires ont pu être observées. C'est notamment dû au fait que les données ont été divisés en plusieurs parties, chacune dans notre cas suivant une loi normale. Finalement, appliquer différentes régressions linéaires par parties d'une variable s'est révélé être plus efficace. En plus de faire des prédictions plus exactes, les données ont eu l'air de suivre des droites de fonction linéaire, ce qui est encourageant pour la suite.

Nous sommes arrivés à la conclusion que les classifications (non supervisées) appliquées ont mal découpé la variable temps d'exécution à découper, c'est-à-dire que les parties coupées auraient dû se suivre dans l'ordre, ce qui n'était pas le cas. Il faudra trouver une manière de partitionner qui répond à ce problème.

Pour les deux prochaines années, nous ferons une analyse multidimensionnelle intégrant tous les programmes en même temps. L'objectif des trois années est de finir sur un algorithme d'ordonnancement.

Bibliographie

[1] Direction-des-ressources-humaines, «Livret-synthese-RSU-2020,» 2020. [En ligne]. Available: https://www.inria.fr/sites/default/files/2022-03/Livret-synthese-RSU-2020.pdf. [Accès le Mai 2022].

[2] Ministère de l'économie et des finances, Ministère de l'enseignement supérieur, de la recherche et de l'innovation, «Contrat d'objectifs et de performance 2019-2023 entre l'Etat et l'INRIA,» 2019. [En ligne]. Available: https://www.inria.fr/sites/default/files/2020-02/COP_INRIA_2019-2023_version-finalesssignatures.pdf. [Accès le Mai 2022].

[3] Kopernic, «KOPERNIC 2021 activity report,» 2021. [En ligne]. Available: https://raweb.inria.fr/rapportsactivite/RA2021/kopernic/uid0.html. [Accès le Mai 2022].

[4] pandas development team, «User Guide - pandas 1.4.2 documentation,» 2022. [En ligne]. Available: https://pandas.pydata.org/docs/user_guide/index.html#user-guide. [Accès le Mai 2022].

[5] NumPy Developers, «NumPy Reference - NumPy v1.22 Manual,» 20 Mai 2022. [En ligne]. Available: https://numpy.org/doc/stable/reference/. [Accès le Mai 2022].

[6] John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team, «PyPlot tutorial - Matplotlib 3.5.2 documentation,» 2022. [En ligne]. Available: https://matplotlib.org/stable/tutorials/introductory/pyplot.html. [Accès le Mai 2022].

[7] Michael Waskom, «User guide and tutorial - seaborn 0.11.2 documentation,» 2021. [En ligne]. Available: https://seaborn.pydata.org/tutorial.html. [Accès le Mai 2022].

[8] The Scipy community, «SciPy - SciPy v0.16.1 Reference Guide,» 24 Octobre 2015. [En ligne]. Available: https://docs.scipy.org/doc/scipy-0.16.1/reference/. [Accès le Mai 2022].

[9] Python Software Foundation License, «statistics — Mathematical statistics functions - Python 3.10.4,» 2022. [En ligne]. Available: https://docs.python.org/3/library/statistics.html. [Accès le Mai 2022].

[10] Python Software Foundation, «itertools — Functions creating iterators for efficient looping,» 2022. [En ligne]. Available: https://docs.python.org/3/library/itertools.html. [Accès le Mai 2022].

[11] scikit-learn developers, «scikit-learn : machine learning in Python,» Mai 2022. [En ligne]. Available: https://scikit-learn.org/stable/index.html. [Accès le Mai 2022].

Liste des figures

Annexes

1. Pré-traitement et description des données

Fichier ulog_scheduler_0 :

- temps d'execution xtime ;
- variable d'horodatage timestamp qui permettra d'aligner les xtime avec les données de l'autre tableau ;
- variable event (qu'on va supprimer car elle nous sert pas encore)

ulog_sensor_combined_0 :

- timestamp ;
- 3 variables gyro_rad[0], gyro_rad[1], gyro_rad[2] : données de gyroscopes avec chacune des 3 dimensions ;
- 3 variables accelerometer_m_s2[0], accelerometer_m_s2[1], accelerometer_m_s2[2] : données d'accelerometres avec chacune des 3 dimensions ;
- à supprimer : gyro_integral_dt et accelerometer_integral_dt (inutiles), accelerometer_timestamp_relative et accelerometer_clipping (1 seule valeur).

```
# argument : un fichier csv
# cette fonction convertit fichier tableau csv to un dataframe
# elle retourne le dataframe avec ajustements
# sortie (en return) : un dataframe
def conversion(fichier):
    import pandas as pd
    # on met dans data on met un dataframe qui lit le csv, qu on trie par l
a colonne timestamp
    data = pd.DataFrame(pd.read_csv(fichier)).sort_values(by=['timestamp'],
ignore_index=True)
    return data

# argument : un dataframe
# cette fonction va supprimer toutes les variables qui ont certains mots re
nseignes
# en plus de creer un dataframe nettoyé, elle retourne une description des
changements
# sortie (en return) :
#  [0] un string , [1] un dataframe

def suppression_variables_mot(data) :
    texte=""
    colonnes_a_supprimer=[]
    # pour chaque colonne danes les colonnes de data
    for colonne in data.columns :

        # si il y a le mot event dans le nom de colonne
        if "event" in colonne or "integral" in colonne :
            # alors on met cette colonne dans la liste des colonnes_a_suppr
```

```
imer
            colonnes_a_supprimer.append(colonne)

    # si il n y a aucune colonne à supprimer
    if len(colonnes_a_supprimer)==0:
        # on met ça (et seulement ça) dans texte
        texte="Aucun changement."

    # sinon (si des colonnes à supprimer)
    else:
        # on met dans texte :
        #  ce qu il y avait deja
        #  et la liste des colonnes_a_supprimer
        texte=texte+ ", ".join(colonnes_a_supprimer)

    # on supprime ces colonnes
    data=data.drop(colonnes_a_supprimer, axis=1)

    return texte, data

# argument : un dataframe
# cette fonction supprime toutes les variables qui ont trop peu de valeurs
differentes
#  (on a choisi moins de 3)
# en plus de creer un dataframe nettoyé, elle retourne une description des
changements
# sortie (en return) :
#  [0] un string
#  [1] un dataframe
def suppression_variables_peu_de_valeurs(data) :
    texte=" "
    colonnes_a_supprimer=[]

    # pour chaque colonne dans les colonnes de data
    for colonne in data.columns :
        # si le nombre de valeurs differentes dedans ou si que des nan
        if len(np.unique(data[colonne])) <= 3 or  np.unique(np.isnan(data[c
olonne])):

            # alors on ajoute cette colonne a la liste des colonnes_a_suppr
imer
            colonnes_a_supprimer.append(colonne)

            # on met une ligne dans texte qui décrit le nombre de valeurs d
ifferentes de la variable supprimee
            texte = texte +"\nLa variable "+colonne+" avait "+ str(len(np.u
nique(data[colonne]))) +" valeur(s) differente(s)."

    # on enleve à data ces colonnes
    data=data.drop(colonnes_a_supprimer, axis=1)

    # on rajoute à texte ça :
    texte = texte+ "\nDistribution de chaque colonne restante :"
```

```
    # pour chaque colonne dans les colonnes de data
    for colonne in data.columns:

        # on met une ligne dans texte qui décrit le nombre de valeurs diffe
rentes de la variable supprimee
        texte = texte +"\nLa variable "+colonne+" a "+ str(len(np.unique(da
ta[colonne]))) +" valeurs differentes."

    return texte, data

# argument : un dataframe
# cette fonction lance les differentes precedentes fonctions de tris :
#  - suppression_variables_mot
#  - suppression_variables_peu_de_valeurs
# en plus de creer un dataframe nettoyé, elle retourne une description des
changements
# sortie (en return) :
#  [0] un string
#  [1] un dataframe
def tri_total(data):
    texte=""
    # on ajoute a texte un titre et le texte de suppression_variables_mot
    texte=texte+"\n\nEtape de suppression de variable(s) qui ont certains n
oms :\n"
    texte=texte+suppression_variables_mot(data)[0]
    data=suppression_variables_mot(data)[1]

    # on ajoute a texte un titre et le texte de suppression_variables_peu_d
e_valeurs
    texte=texte+"\n\nEtape de suppression de variable(s) qui ont trop peu d
e valeurs differentes :"
    texte=texte+suppression_variables_peu_de_valeurs(data)[0]
    data=suppression_variables_peu_de_valeurs(data)[1]


    return texte, data

import pandas as pd
import numpy as np
for tableau in ["ulog_scheduler_0.csv",
                "ulog_sensor_combined_0.csv"] :
    texte=""
    if __name__ == "__main__":

        print("\n",str(tableau), " originel :")

        data=conversion(tableau)

        print("\nColonnes originelles :\n", data.columns)
        print(tri_total(data)[0])

        data=tri_total(data)[1]

        print("\n",str(tableau), " final (head) :")
```

```
        print(data.head())
        print("\nColonnes finales :\n", data.columns)


 ulog_scheduler_0.csv  originel :

Colonnes originelles :
 Index(['timestamp', 'xtime', 'event'], dtype='object')


Etape de suppression de variable(s) qui ont certains noms :
event

Etape de suppression de variable(s) qui ont trop peu de valeurs differentes
:
Distribution de chaque colonne restante :
La variable timestamp a 70395 valeurs differentes.
La variable xtime a 1012 valeurs differentes.

 ulog_scheduler_0.csv  final (head) :
   timestamp  xtime
0   15101432    194
1   15101629     60
2   15101692    117
3   15102433     32
4   15104433    173

Colonnes finales :
 Index(['timestamp', 'xtime'], dtype='object')

 ulog_sensor_combined_0.csv  originel :

Colonnes originelles :
 Index(['timestamp', 'gyro_rad[0]', 'gyro_rad[1]', 'gyro_rad[2]',
       'gyro_integral_dt', 'accelerometer_timestamp_relative',
       'accelerometer_m_s2[0]', 'accelerometer_m_s2[1]',
       'accelerometer_m_s2[2]', 'accelerometer_integral_dt',
       'accelerometer_clipping'],
      dtype='object')


Etape de suppression de variable(s) qui ont certains noms :
gyro_integral_dt, accelerometer_integral_dt

Etape de suppression de variable(s) qui ont trop peu de valeurs differentes
:
La variable accelerometer_timestamp_relative avait 1 valeur(s) differente(s
).
La variable accelerometer_clipping avait 1 valeur(s) differente(s).
Distribution de chaque colonne restante :
La variable timestamp a 10451 valeurs differentes.
La variable gyro_rad[0] a 10451 valeurs differentes.
La variable gyro_rad[1] a 10451 valeurs differentes.
La variable gyro_rad[2] a 10451 valeurs differentes.
```

```
La variable accelerometer_m_s2[0] a 10449 valeurs differentes.
La variable accelerometer_m_s2[1] a 10450 valeurs differentes.
La variable accelerometer_m_s2[2] a 10345 valeurs differentes.

 ulog_sensor_combined_0.csv  final (head) :
   timestamp  gyro_rad[0]  gyro_rad[1]  gyro_rad[2]  accelerometer_m_s2[0]
\
0   15168302     0.002325     0.000685    -0.000189               0.011836
1   15172183     0.002522     0.002322    -0.000569               0.008263
2   15175850     0.001504     0.001473    -0.001233               0.008147
3   15180358    -0.001356    -0.001964    -0.000898               0.028387
4   15251215     0.000525    -0.003064     0.002747               0.030895

   accelerometer_m_s2[1]  accelerometer_m_s2[2]
0              -0.035288              -9.800760
1              -0.006083              -9.777782
2               0.001566              -9.780920
3              -0.028178              -9.784050
4               0.000947              -9.798773

Colonnes finales :
 Index(['timestamp', 'gyro_rad[0]', 'gyro_rad[1]', 'gyro_rad[2]',
       'accelerometer_m_s2[0]', 'accelerometer_m_s2[1]',
       'accelerometer_m_s2[2]'],
      dtype='object')
```

```python
# arguments :
#  -un fichier csv (ulog_sensor_combined_0)
#  -un fichier csv (ulog_scheduler_0)
# cette fonction va concatener les fichiers ulog_scheduler_0.csv et ulog_se
nsor_combined_0 passés en argument
# elle va permettre de faire correspondre des xtime avec des lignes de donn
ees en alignant des timestamp (differents)
# sortie (en return) : un dataframe

def concatenation(data_fichier_xtimes, data_fichier2):
    import pandas as pd
    import numpy as np

    # on met dans un objet data la concatenation de ulog_scheduler_0 avec l
e fichier csv passe en argument
    #  les 2 fichiers seront concatenes l un au dessus l autre
    #   mais ensuite on les trie par la variable timestamp dans l'ordre cro
issant
    data = pd.concat([data_fichier_xtimes, data_fichier2]).sort_values(by=[
'timestamp'], ignore_index=True)

    # on va chercher la premiere variable qui n a ni xtime ni timestamp dan
s son nom
    #  pour chaque colonne dans les colonnes de data
    for colonne in data.columns :
        # si on a ni le mot timestamp ni le mot xtime dans le nom de la col
onne
        if ("timestamp" in colonne or "xtime" in colonne)==False :
```

```
            # alors on met dans la variable choix_variable cette variable
            choix_variable=colonne
            # on arrete la boucle des que ça arrive
            break

    # il faut mettre la variable timestamp dans une variable à part, qu'on
va mettre en format numpy, sinon ça bug
    A=data["timestamp"].to_numpy()

    # pour chaque n°j jusqu a la fin de data
    for j in range(len(data)-1):

        # si dans une ligne n°j, sa variable ("choix_variable" choisie prec
edemment) n est pas vide
        #  et si sa ligne suivante (n°j+1) est vide
        if np.isnan(data[choix_variable][j]) == False and np.isnan(data[cho
ix_variable][j+1]) == True:
            # alors on remplace le timestamp (dans A) de cette ligne (n°j+1
) par le timestamp de la ligne n°j
            A[j+1] =A[j]


    # et donc on remet cette variable A (avec les timestamp changés) a la p
lace de timestamp dans data
    data["timestamp"]=A

    # pour chaque groupe de timestamps egaux on garde seulement le premier,
on supprime le reste
    #  on enleve l index
    data = data.groupby(['timestamp']).min().dropna().reset_index(inplace=F
alse,
                                                                  drop=True
)

    # ces 2 etapes precedentes ont permis d ajouter un xtime aux lignes san
s xtime
    #  c est a dire dans les lignes issues du tableau sans xtime on a mis l
e xtime de la ligne suivante
    #   (les lignes ont ete rangee dans l ordre coissant des timestamp)

    return data

# argument : un dataframe
# cette fonction va supprimer la variable nommee timestamp dans le data de
concatenation car on en a plus besoin
# sortie (en return) : un dataframe
def suppression_timestamp(data):
    # suppression de colonne(s) avec le mot timestamp dans le nom
    for colonne in data.columns :
        if ("timestamp" in colonne)==True :
            data=data.drop(colonne, axis=1)

    return data
```

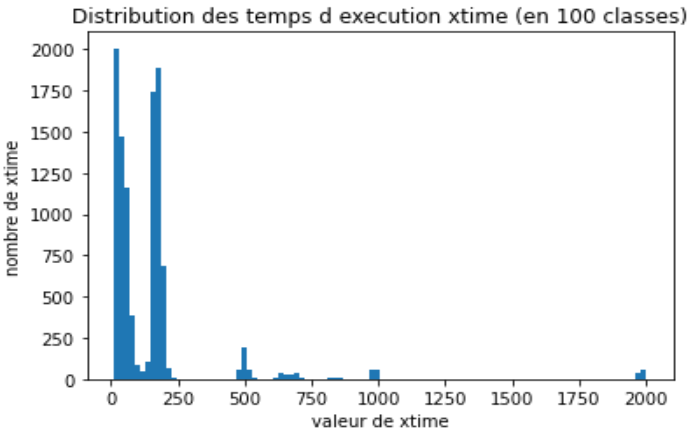
```
# argument : un dataframe
# cette fonction normalise les donnees sauf xtime (la premiere colonne) ent
re 0 et 1
# sortie (en return) : un dataframe
def normalisation(data):
    from sklearn import preprocessing
    import pandas as pd

    # on met dans un objet (type pandas.core.indexes.base.Index) les colonn
es de base
    col=data.columns

    # on met dans mms le modele MinMaxScaler de normalisation, une normalis
ation de 0 à 1
    mms=preprocessing.MinMaxScaler(feature_range=(0,1))

    # on applique ce modele de normalisation à partir de la 2 colonne jusqu
'à la derniere et on remplace dans data
    data.values[:,1:]=mms.fit_transform(data.values[:,1:])

    # on remet data en dataframe en precisant en noms de colonnes les colon
nes de base
    data=pd.DataFrame(data,columns=col)

    return data

# on met dans data la concatenation du fichier avec ulog_scheduler_0.csv
data=concatenation(tri_total(conversion("ulog_scheduler_0.csv"))[1],
                   tri_total(conversion("ulog_sensor_combined_0.csv"))[1])

# on lui supprime ses timestamp
data=suppression_timestamp(data)

# on normalise les colonnes (sauf xtime)
data=normalisation(data)

print(data.head())
```

```
   xtime  gyro_rad[0]  gyro_rad[1]  gyro_rad[2]  accelerometer_m_s2[0]  \
0   16.0     0.539389     0.628170     0.501068               0.464922
1   50.0     0.539565     0.629149     0.500877               0.464798
2  174.0     0.538653     0.628641     0.500543               0.464794
3   19.0     0.536090     0.626586     0.500711               0.465495
4  167.0     0.538545     0.627787     0.501683               0.464896

   accelerometer_m_s2[1]  accelerometer_m_s2[2]
0               0.542967               0.974114
1               0.553677               0.974372
2               0.556481               0.974337
3               0.545574               0.974302
4               0.533710               0.974070
```

```
print(data.describe())
```

```
              xtime   gyro_rad[0]   gyro_rad[1]   gyro_rad[2]  \
count  10351.000000  10351.000000  10351.000000  10351.000000
mean     150.498696      0.522105      0.648220      0.515068
std      230.901346      0.119022      0.105857      0.229539
min       10.000000      0.000000      0.000000      0.000000
25%       32.000000      0.523617      0.620539      0.484492
50%      131.000000      0.536906      0.628728      0.501230
75%      177.000000      0.543499      0.653655      0.510377
max     2002.000000      1.000000      1.000000      1.000000

       accelerometer_m_s2[0]  accelerometer_m_s2[1]  accelerometer_m_s2[2]
count           10351.000000           10351.000000           10351.000000
mean                0.451353               0.504100               0.974165
std                 0.018138               0.119264               0.012558
min                 0.000000               0.000000               0.000000
25%                 0.438880               0.523677               0.973328
50%                 0.457186               0.539803               0.974738
75%                 0.464586               0.549538               0.975724
max                 1.000000               1.000000               1.000000
```

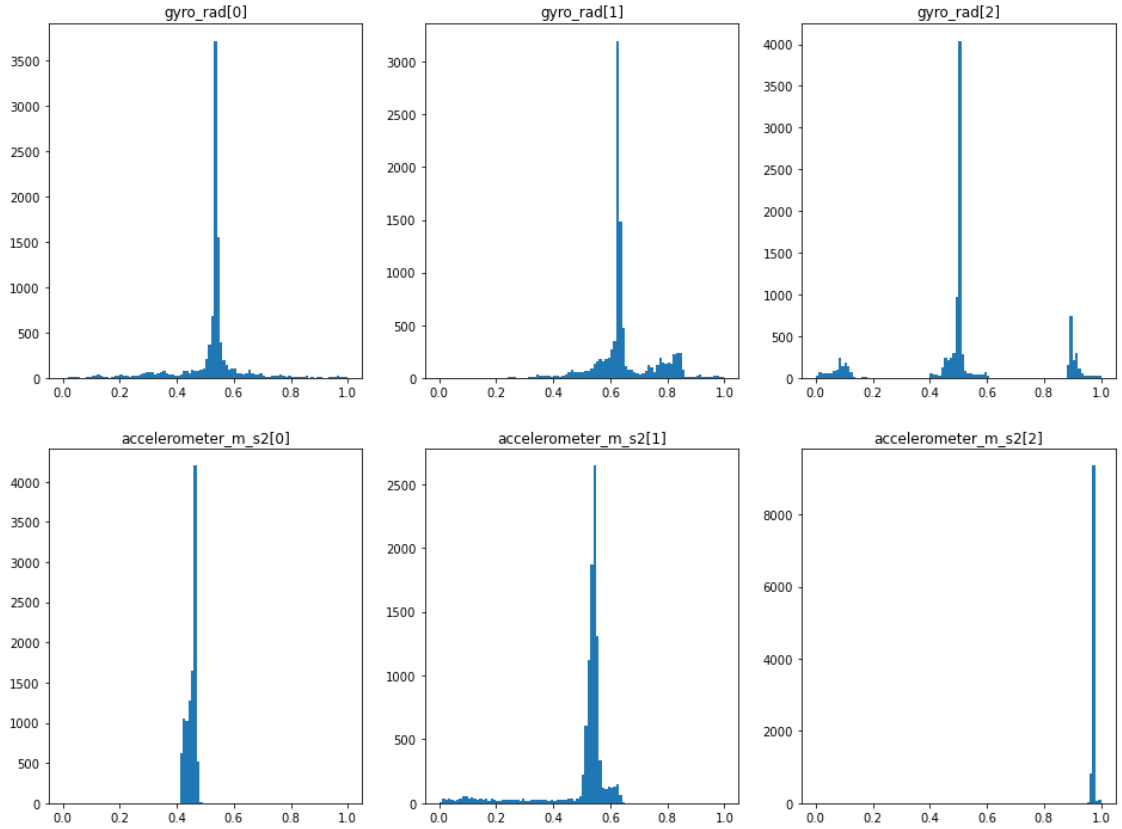
```
#Histogramme des xtimes
import matplotlib.pyplot as plt
import numpy as np
plt.hist(np.array(data["xtime"]),bins=100)
plt.xlabel("valeur de xtime")
plt.ylabel("nombre de xtime")
plt.title("Distribution des temps d execution xtime (en 100 classes)")
plt.show()
```




```
#Histogramme par variable
import matplotlib.pyplot as plt
import numpy as np

data_variables=data.copy()
data_variables=data_variables.drop(["xtime"], axis=1)

print("Distribution de chaque variable gyro et accelerometer (normalisée)")

place=0
plt.figure(figsize=(16,12))
for colonne in data_variables.columns :
```

```
    place=place+1
    plt.subplot(2,3,place)
    plt.hist(np.array(data_variables[colonne]),bins=100)
    plt.title(colonne)
plt.show()
```

Distribution de chaque variable gyro et accelerometer (normalisée)




```
import seaborn

data_variables=data.copy()
data_variables=data_variables.drop(["xtime"], axis=1)

liste_colonnes=data_variables.columns

place=0
plt.figure(figsize=(16,24))
for colonne in liste_colonnes :
    place=place+1
    plt.subplot(4,3,place)
    seaborn.scatterplot(x="xtime",
                        y=colonne,
                        data=data)
    plt.title("".join([colonne,"~xtime"]))

for colonne in liste_colonnes :
    place=place+1
    plt.subplot(4,3,place)
    seaborn.scatterplot(x=colonne,
```

```
                        y="xtime",
                        data=data)
    plt.title("".join(["xtime~",colonne]))
plt.show()
```

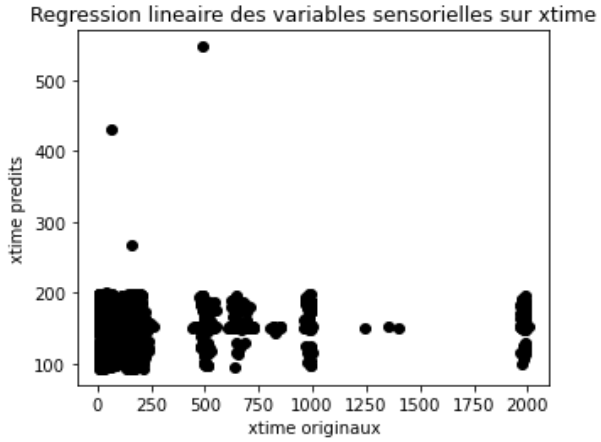
2. Premières régressions linéaires

```
# On run le précédent fichier afin d'avoir accès aux fonctions
%run "1_premiers_traitements_donnees.ipynb"

# on met dans data la concatenation du fichier avec ulog_scheduler_0.csv
data=concatenation(tri_total(conversion("ulog_scheduler_0.csv"))[1],
                   tri_total(conversion("ulog_sensor_combined_0.csv"))[1])

# on lui supprime ses timestamp
data=suppression_timestamp(data)

# on normalise les colonnes (sauf xtime)
data=normalisation(data)

print(data.head())
```

```
   xtime  gyro_rad[0]  gyro_rad[1]  gyro_rad[2]  accelerometer_m_s2[0]  \
0   16.0     0.539389     0.628170     0.501068               0.464922
1   50.0     0.539565     0.629149     0.500877               0.464798
2  174.0     0.538653     0.628641     0.500543               0.464794
3   19.0     0.536090     0.626586     0.500711               0.465495
4  167.0     0.538545     0.627787     0.501683               0.464896

   accelerometer_m_s2[1]  accelerometer_m_s2[2]
0               0.542967               0.974114
1               0.553677               0.974372
2               0.556481               0.974337
3               0.545574               0.974302
4               0.533710               0.974070
```

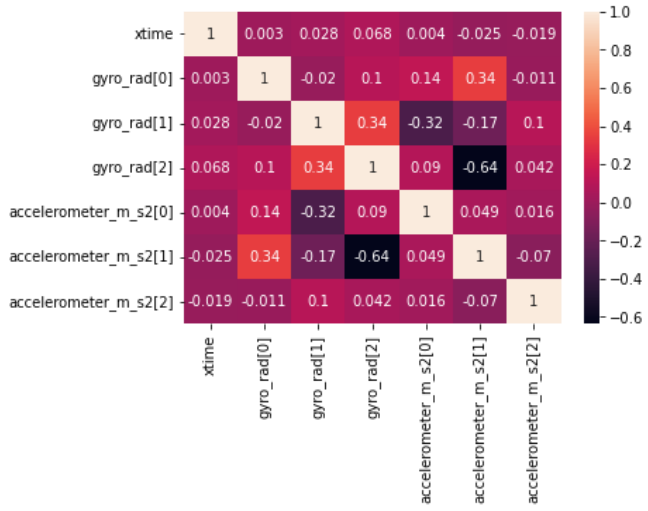
```
#Fonction régressions linéaires avec nuages (xtime observés, xtime prédits)

def regression_lineaire(data):
    from sklearn.linear_model import LinearRegression
    import matplotlib.pyplot as plt

    y=data["xtime"]
    x=data.drop(["xtime"], axis=1)

    lmodellineaire = LinearRegression(normalize=True)
    lmodellineaire.fit(x,y)

    y_predict=lmodellineaire.predict(x)

    plt.figure(figsize=(5,4))
    plt.scatter(y,
```

```
                y_predict,
                color='black')
    plt.title("Regression lineaire des variables sensorielles sur xtime")
    plt.xlabel("xtime originaux")
    plt.ylabel("xtime predits")

    plt.show()

regression_lineaire(data)
```

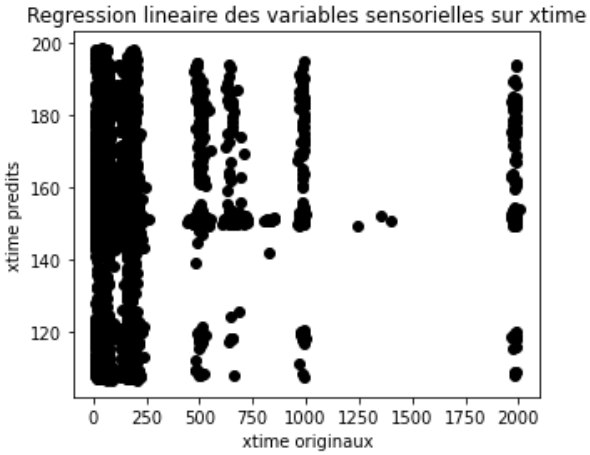



```
# Matrice des corrélations des donnees normalisees
import seaborn as sns
matrice_corr = data.corr().round(3)
sns.heatmap(data=matrice_corr, annot=True)
```

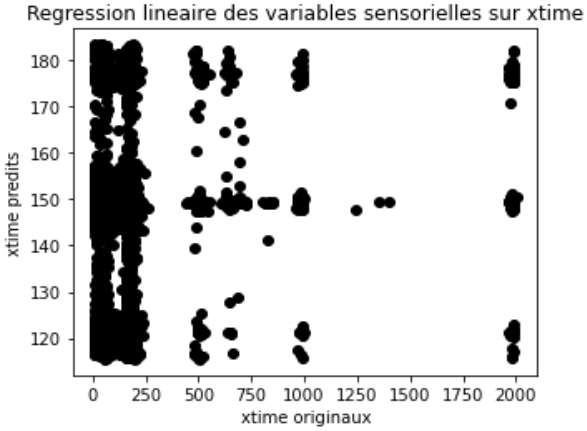
```
<AxesSubplot:>
```




```
data_4_variables=data[["xtime","gyro_rad[1]","gyro_rad[2]","accelerometer_m_s2[0]","accelerometer_m_s2[1]"]]

regression_lineaire(data_4_variables)
```




```
data_4_variables=data[["xtime","gyro_rad[2]","accelerometer_m_s2[0]"]]

regression_lineaire(data_4_variables)
```



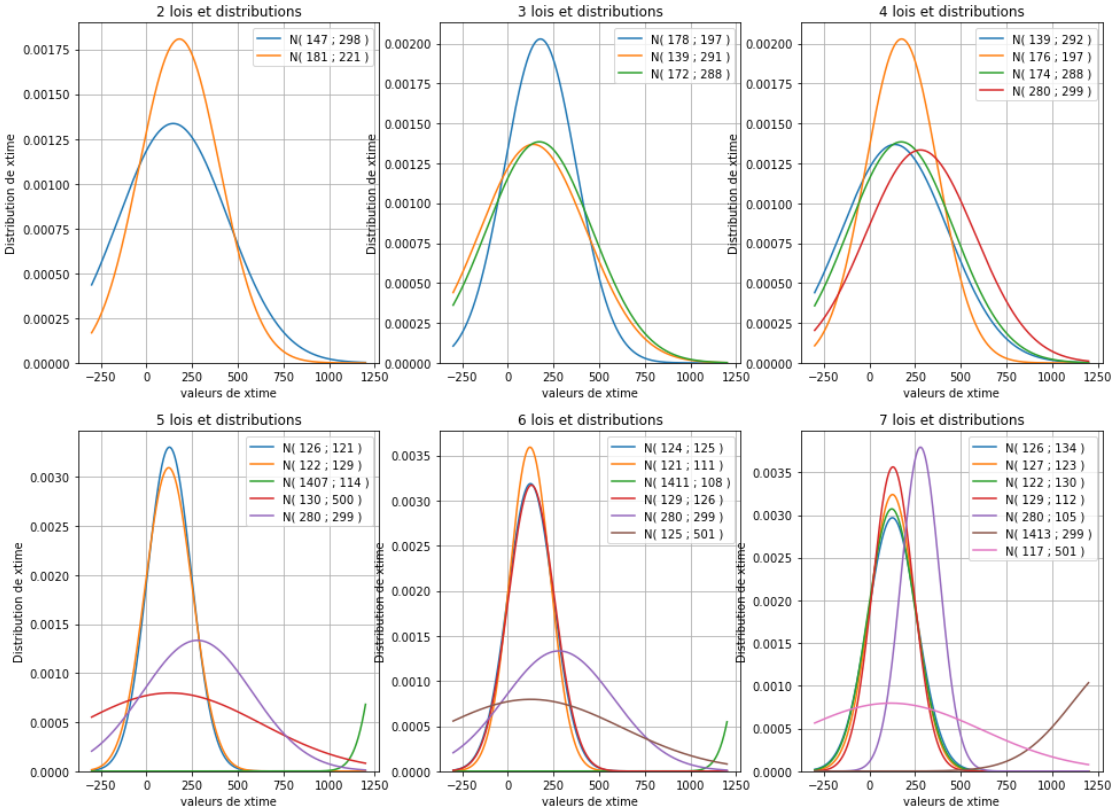
3. Régressions linéaires optimisées avec classification non-supervisée

```
%run "2_premieres_regressions_lineaires.ipynb"
```

```
    #Graphique a courbe de la distribution des xtime suivant une loi normal
e
import numpy as np
import seaborn as sns
import scipy
import statistics
import matplotlib.pyplot as plt

X=data["xtime"]

mean=np.mean(X)
sd=statistics.stdev(X)

x = np.linspace(-400, 700, 100)
y = scipy.stats.norm.pdf(x,mean,sd)

plt.plot(x,y, color='red')
plt.grid()
```

```
plt.ylim(0,0.002)
plt.title('Courbe de xtime',fontsize=10)
plt.xlabel('x')
plt.ylabel('Distribution de xtime')
plt.show()

print("xtime suit la loi[",mean,",",sd,"]")

import numpy as np
import seaborn as sns
import scipy
import statistics
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

#k=nombre_classes
def kmeans(nombre_classes):
    features=data
    scaler = StandardScaler()
    features_std = scaler.fit_transform(features)
    cluster = KMeans(n_clusters=nombre_classes)
    model = cluster.fit(features_std)

    return model.labels_

def graphiques_lois_normales(nombre_classes):
    import matplotlib.pyplot as plt

    # moyennes des xtime par classe
    moyennes=data.groupby(kmeans(nombre_classes))['xtime'].mean()
    # ecarts-types des xtime par classe
    ecarts_types=data.groupby(kmeans(nombre_classes))['xtime'].std()

    # on va faire 10000 predictions entre -300 et 1200
    x = np.linspace(-300, 1200, 10000)

    #pour chaque classe i
    for i in range(nombre_classes):
            #on fait une courbe avec en abscisse x et en ordonnees la loi,
labelisee par un texte avec sa moyenne et son ecart type
        plt.plot(x,
                 scipy.stats.norm.pdf(x,
                                      moyennes[i],
                                      ecarts_types[i]),
                 label="".join(["N( ",
                                str(round(moyennes[i])),
                                " ; ",
                                str(round(ecarts_types[i])),
                               " )"] ) )

    plt.legend(loc="upper right")
    plt.grid()
    plt.ylim(0)
    plt.title("".join([str(i+1), " lois et distributions"]))
```

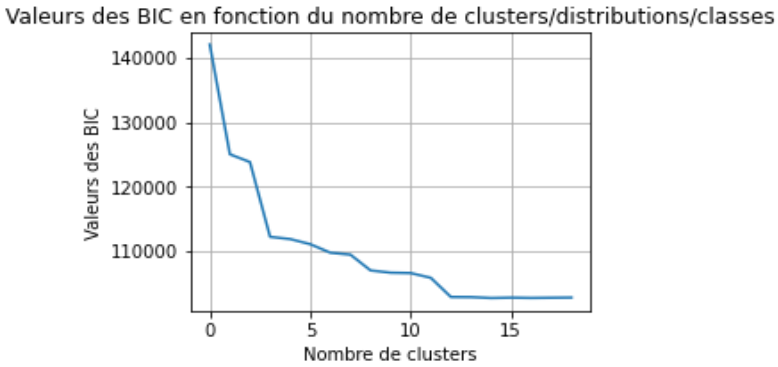
```
    plt.xlabel('valeurs de xtime')
    plt.ylabel('Distribution de xtime')

place=0
plt.figure(figsize=(16,12))
for i in range(2,8):
    place=place+1
    plt.subplot(2,3,place)
    graphiques_lois_normales(i)
plt.show()
```



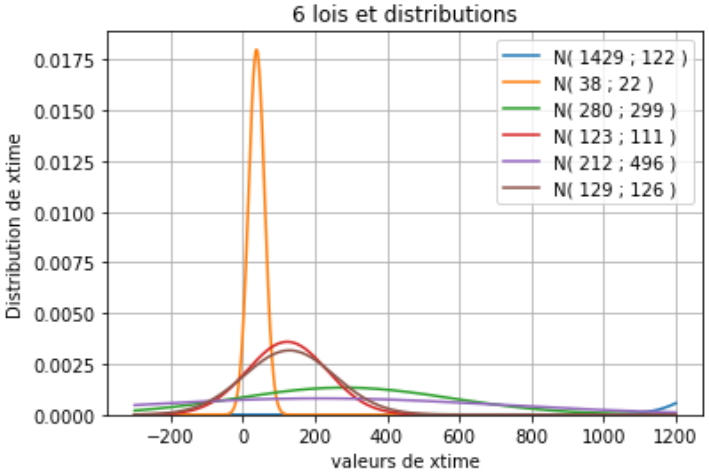

```
#BIC le plus petit ?
import numpy as np
from sklearn.mixture import GaussianMixture

X = np.array(data)[:, 0:1]

listebic=[]

for i in range(1, 20):
    gmmodel=GaussianMixture(n_components=i, random_state=0).fit(X)
    listebic.append(gmmodel.bic(X))
    print(i,":",gmmodel.bic(X)," ; ",end='')


plt.figure(figsize=(4, 3))
plt.plot(listebic)
plt.grid()
plt.title("Valeurs des BIC en fonction du nombre de clusters/distributions/
classes")
```

```
plt.xlabel('Nombre de clusters')
plt.ylabel('Valeurs des BIC')
plt.show()
```

```
1 : 142052.44343574328  ; 2 : 125025.51503375747  ; 3 : 123820.13294426129
; 4 : 112218.61681458337  ; 5 : 111873.52718655438  ; 6 : 111079.3114042817
2  ; 7 : 109753.79682001006  ; 8 : 109455.31084462839  ; 9 : 107005.5937444
5121  ; 10 : 106657.85863275368  ; 11 : 106596.32982077594  ; 12 : 105859.1
2737728537  ; 13 : 102884.85345225477  ; 14 : 102864.4184345657  ; 15 : 102
728.73528186102  ; 16 : 102782.32352204039  ; 17 : 102739.82810117143  ; 18
: 102767.78768158254  ; 19 : 102794.60030595535  ;
```



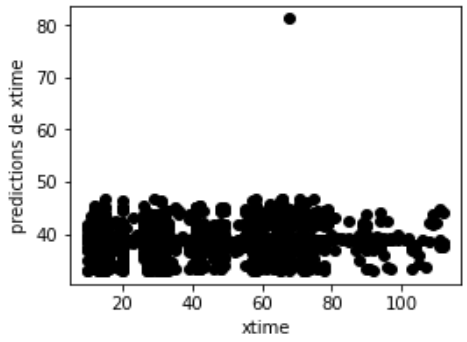

```
graphiques_lois_normales(6)
```

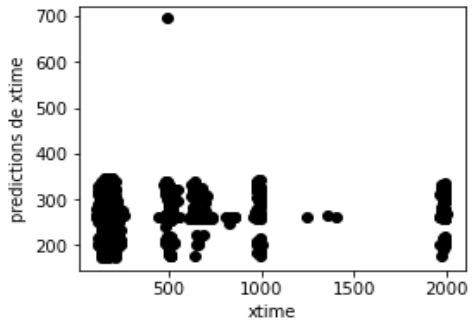



```
import matplotlib.pyplot as plt
from sklearn import mixture
from sklearn.linear_model import LinearRegression

data=data.copy()
# xtime :
x=data.values[:,0].reshape(-1, 1)

# classification GMM en 6 classes
g = mixture.GaussianMixture(n_components=6,covariance_type='full')
g.fit(x)

classes=[]
# transformation des 6 classes en 2 classes avec pour critere une moyenne a
```

```
u dessus ou en dessous de :
separateur= round(data.values[:,0].mean()) #environ 100

# pour chaque c dans g.predict(x)
for c in g.predict(x):
    # si c'est <100 on met dans la nouvelle classe n°0
    if g.means_[c, 0]<=separateur:
        classes.append(0)
    # sinon (si >100) on met dans la nouvelle classe n°1
    else:
        classes.append(1)

# on met cette liste dans le tableau
data['classes'] = classes

print(data.head())

# pour chaque d dans la liste [classe=n°0 ; classe=n°1],
#  regression lineaire
for d in data[data["classes"]==0], data[data["classes"]==1]:

    # X est les valeurs d un dataframe compose des variable hors xtime et h
ors classes
    X=d.values[:,1:-1]

    # en abscisse = xtime originaux
    y=d.values[:,0]

    # regression lineaire sur xtime y selon les variables dans X
    lmodellineaire = LinearRegression()
    lmodellineaire.fit(X,y)
    # en ordonnee = xtime predits
    y_predict = lmodellineaire.predict(X)

    plt.figure('Linear model plot',figsize=(4, 3))
    plt.scatter(y, y_predict,  color='black')
    plt.ylabel("predictions de xtime")
    plt.xlabel("xtime")
    plt.show()
```

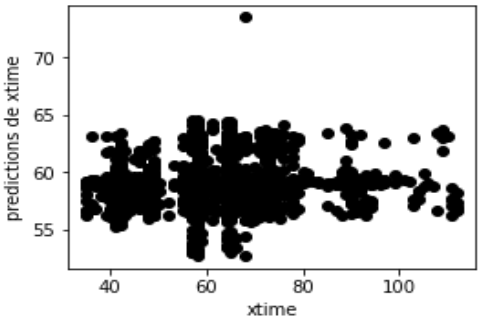
```
   xtime  gyro_rad[0]  gyro_rad[1]  gyro_rad[2]  accelerometer_m_s2[0]  \
0   16.0     0.539389     0.628170     0.501068               0.464922
1   50.0     0.539565     0.629149     0.500877               0.464798
2  174.0     0.538653     0.628641     0.500543               0.464794
3   19.0     0.536090     0.626586     0.500711               0.465495
4  167.0     0.538545     0.627787     0.501683               0.464896

   accelerometer_m_s2[1]  accelerometer_m_s2[2]  classes
0               0.542967               0.974114        0
1               0.553677               0.974372        0
2               0.556481               0.974337        1
3               0.545574               0.974302        0
4               0.533710               0.974070        1
```



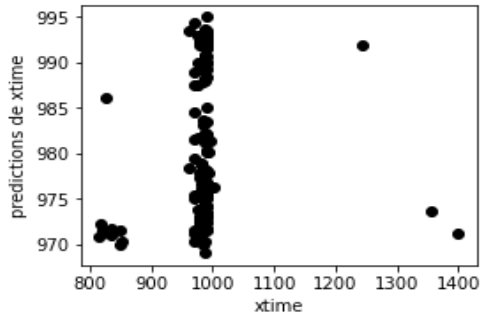



```
nombre_classes_1=6

import matplotlib.pyplot as plt
from sklearn import mixture
from sklearn.linear_model import LinearRegression

def creation_classes_gmm(data):
    import matplotlib.pyplot as plt
    from sklearn import mixture
    from sklearn.linear_model import LinearRegression

    data=data.copy()

    x=data.values[:,0].reshape(-1, 1)

    # classification GMM en 6 classes
    g = mixture.GaussianMixture(n_components=nombre_classes_1,covariance_ty
pe='full')
    g.fit(x)
    data['classes']= g.predict(x)

    return data


data=creation_classes_gmm(data)

print("describe xtime :\n",data["xtime"].describe())
print("classes :",np.unique(data["classes"]))
print("shape :",data.shape)

for d in [data[data["classes"]==i] for i in range(nombre_classes_1)]:

    # X est les valeurs d un dataframe composé des variable hors xtime(1) e
```

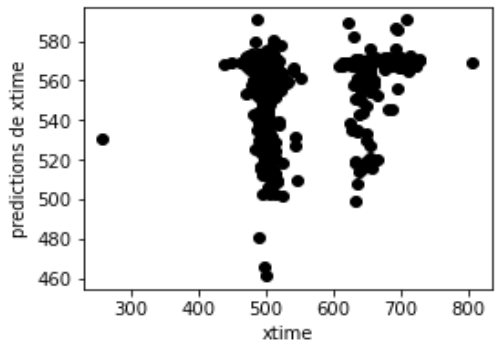
```
t hors classes(-1)
    X=d.values[:,1:-1]

    # en abscisse = xtime originaux
    y=d.values[:,0]

    # regression lineaire sur xtime y selon les variables dans X
    lmodellineaire = LinearRegression()
    lmodellineaire.fit(X,y)
    # en ordonnee = xtime predits
    y_predict = lmodellineaire.predict(X)

    print(np.unique(d["classes"]))
    plt.figure('Linear model plot', figsize=(4,3))
    plt.scatter(y, y_predict,  color='black')
    plt.ylabel("predictions de xtime")
    plt.xlabel("xtime")
    plt.show()


describe xtime :
 count    10351.000000
mean       150.498696
std        230.901346
min         10.000000
25%         32.000000
50%        131.000000
75%        177.000000
max       2002.000000
Name: xtime, dtype: float64
classes : [0 1 2 3 4 5]
shape : (10351, 8)
[0]
```



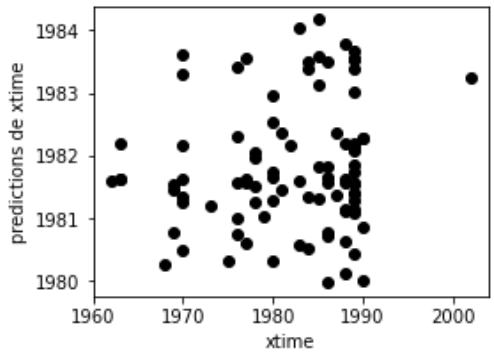

```
[1]
```


predictions de xtime
xtime
995
990
985
980
975
970
800
900
1000
1100
1200
1300
1400

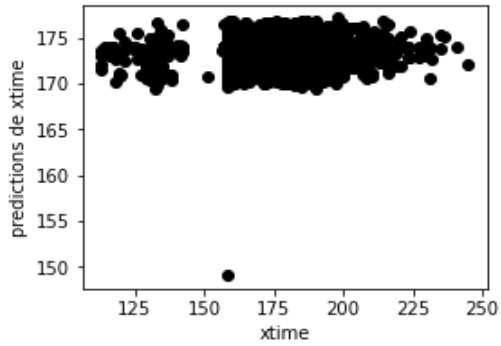
[2]


predictions de xtime
xtime
580
560
540
520
500
480
460
300
400
500
600
700
800

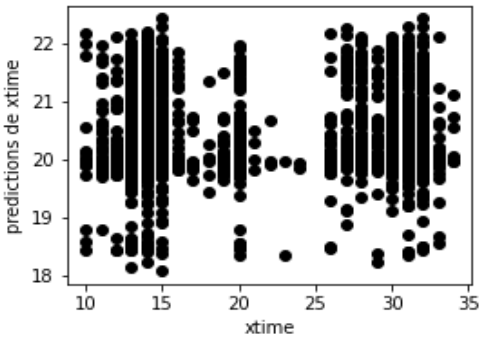
[3]


predictions de xtime
xtime
1984
1983
1982
1981
1980
1960
1970
1980
1990
2000

[4]


predictions de xtime
xtime
175
170
165
160
155
150
125
150
175
200
225
250

[5]


predictions de xtime
xtime
22
21
20
19
18
10
15
20
25
30
35

Résumé :

Le drone nommé PX4 étudié par l'équipe Kopernic à l'INRIA a un jumeau numérique. Accompagné d'un environnement de vol simulé à l'aide d'un logiciel, son microcontrôleur exécute l'autopilote du drone comme lors d'un vol réel. Ces simulations temps-réel permettent de tester le code sans avoir à faire voler un vrai drone.

De là, des donnés sont produites à chaque simulation : des données de temps d'exécutions et des données de capteurs sensoriels, non associées entre elles.

L'autopilote a un ordonnancement qui tend à être optimal, c'est-à-dire un ordre des programmes optimal selon les contraintes de temps. Cet ordonnancement peut être encore amélioré à l'aide de méthodes statistiques basées sur l'effet des données environnementales sur les données de temps d'exécution.

Les algorithmes (de *machine learning*) qui aideront à améliorer l'ordonnancement en question sont complexes. Avant de les décider, il faut d'abord appliquer des régressions linéaires aux temps d'exécutions expliqués selon les données de capteurs sensoriels.

Mots clés : ordonnancement, autopilote, temps d'exécution, temps-réel, capteurs sensoriels, microcontrôleur.

Summary:

The drone named PX4 studied by the Kopernic team at INRIA has a digital twin. Accompanied by a flight environment simulated with software, its microcontroller executes the drone's autopilot as during a real flight. These real-time simulations allow to test the code without having to fly a real drone.

From there, data are produced at each simulation: execution time data and sensory sensor data, not associated with each other.

The autopilot has a scheduling that tends to be optimal, i.e. an optimal order of programs according to time constraints. This ordering can be further improved using statistical methods based on the effect of environmental data on the execution time data.

The (machine learning) algorithms that will help improve the scheduling in question are complex. Before deciding on them, linear regressions must first be applied to the runtimes explained according to the sensory sensor data.

Keywords: scheduling, autopilot, execution time, real-time, sensory sensors, microcontroller.