# OPTIMISATION DU CYCLE DE PESAGE D'UNE BALANCE INDUSTRIELLE PAR INTELLIGENCE ARTIFICIELLE

Mémoire présenté

dans le cadre du programme de maîtrise en ingénierie

en vue de l'obtention du grade de maître ès sciences appliquées (M.Sc.A.)

PAR

**© Chelsea MARADEI**

**Octobre 2024**

**Composition du jury :**

**Yacine Benhamed, président du jury, Université du Québec à Rimouski (UQAR)**

**Ismaïl Khriss, directeur de recherche, Université du Québec à Rimouski (UQAR)**

**Jean Brousseau, codirecteur de recherche, Université du Québec à Rimouski (UQAR)**

**Martin Otis, examinateur externe, Université du Québec à Chicoutimi (UQAC)**

Dépôt initial le 23 juillet 2024 Dépôt final le 3 octobre 2024

## REMERCIEMENTS

Pour commencer, je tiens à remercier Ismaïl Khriss, mon directeur de recherche. Son encadrement, son expertise et sa disponibilité ont été essentiels pour mener à bien ce projet. Je souhaite également adresser mes remerciements à Jean Brousseau, mon codirecteur de recherche, pour son implication et ses conseils. Leurs remarques constructives et leur implication ont contribué à l'aboutissement et à l'enrichissement de ce travail de recherche.

Je souhaite également remercier l'équipe de Premier Tech. Dans un premier temps, je remercie Alexis Darisse et Billy Lapointe pour leur accompagnement et le partage de leurs connaissances. Je veux également remercier Pierre-Luc Lebel, responsable de l'atelier, qui a partagé son savoir-faire et son temps lors de la récolte des données qui était essentielle pour la réussite des objectifs de ce projet. Je remercie Steve Santerre et Jean-Pierre Doré, pour leur expertise partagée concernant les systèmes automatisés. Leur contribution a facilité ma compréhension du système étudié. Enfin, je souhaite remercier l'ensemble de l'équipe d'innovation pour son accueil chaleureux.

Pour conclure, je souhaite exprimer ma profonde gratitude aux membres du jury pour le temps et l'attention qu'ils ont consacrés à l'évaluation de mon travail. Leurs précieux commentaires et suggestions ont significativement contribué à améliorer la qualité de ce mémoire

# RÉSUMÉ

L'ensachage de produits en vrac dans l'industrie nécessite une vérification rigoureuse de la masse durant le remplissage des sacs. Cependant, ces contrôles sont fortement réglementés et impliquent des comportements du système qui tendent à ralentir le processus d'ensachage. L'objectif de ce projet est d'optimiser les étapes de la chaîne de conditionnement en améliorant la rapidité du cycle, ce qui permettrait d'accroitre le nombre de sacs produits en un temps donné. L'accélération du cycle de pesage ne doit pas compromettre la précision, qui est un critère essentiel. Il est donc nécessaire de vérifier que le poids des sacs produits soit dans l'intervalle de masse accepté. Un jeu de données recueillies en milieu industriel a été constitué. Une première approche, basée directement sur le poids brut, a d'abord été mise en place, et des modèles d'intelligence artificielle ainsi que des méthodes statistiques conçus pour la prédiction de séries temporelles ont été testés et évalués. Cette approche en une seule étape s'est révélée peu concluante, ce qui a conduit à reconsidérer la stratégie de résolution du problème. Une seconde approche composée d'une étape de filtrage suivie d'une optimisation du cycle a alors été testée. Pour l'étape de filtrage, des modèles d'apprentissage supervisé et des filtres numériques ont été comparés pour leur efficacité à atténuer les bruits causés par l'environnement industriel de la balance. Les filtres numériques se sont distingués par leurs bons résultats et leur facilité d'implémentation. L'apprentissage par renforcement a ensuite été utilisé sur les données filtrées pour optimiser le cycle de pesage commandé par le système. Les premiers résultats obtenus à ce jour montrent que cette méthode est intéressante et prometteuse pour notre application. Cependant, l'optimisation des hyperparamètres et du système de récompenses représente un travail conséquent nécessitant des ressources computationnelles importantes, ainsi que du temps pour être pleinement réalisée.

Mots clés : Intelligence artificielle, balance industrielle, ensachage industriel, optimisation du cycle de pesage, séries temporelles, filtrage de bruit, apprentissage supervisé, apprentissage par renforcement, réseaux de neurones.

## ABSTRACT

Bulk product packaging in the industry requires rigorous weight control during bag filling. However, these controls are highly regulated and involve system behaviors that tend to slow down the bagging process. This project aims to optimize the steps in the packaging chain by improving the speed of the cycle, thereby increasing the number of bags produced in a given time. Acceleration of the weighing cycle must not compromise accuracy, an essential criterion. It is, therefore, necessary to check that the weight of the bags produced is within the accepted weight range. A dataset collected in an industrial environment was set up. An initial approach, based directly on raw weight, was implemented, and artificial intelligence models and statistical methods designed for time series prediction were tested and evaluated. This one-step approach proved inconclusive, leading to reconsidering the problem-solving strategy. A second approach consisting of a filtering step followed by cycle optimization was then tested. For the filtering step, supervised learning models and digital filters were compared for their effectiveness in attenuating noise caused by the scale's industrial environment. The digital filters stood out for their good results and ease of implementation. Reinforcement learning was then used on the filtered data to optimize the weighing cycle controlled by the system. The first results show that this method is interesting and promising for our application. However, optimizing the hyperparameters and the reward system represents a substantial task requiring significant computational resources and time to be fully realized.



*Keywords*: Artificial intelligence, industrial scales, industrial bagging, weighing cycle optimization, time series, noise filtering, supervised learning, reinforcement learning, neural networks.

## TABLE DES MATIÈRES

## LISTE DES TABLEAUX

## LISTE DES FIGURES

## LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

**ADALINE** Neurone linéaire adaptatif (*Adaptive Linear Neuron*)

**ADF** Test augmenté de Dickey-Fuller (*Augmented Dickey-Fuller test*)

**ANFIS** Système d'inférence neuro-floue adaptatif (*Adaptive Neuro Fuzzy Inference System*)

**API** Automate Programmable Industriel

**ARIMA** Moyenne mobile autorégressive intégrée (*Auto-Regressive Integrated Moving Average*)

**ARMA** Moyenne mobile autorégressive (*Auto-Regressive Moving Average*)

**CNN** Réseaux de neurons convolutifs (*Convolutional Neural Networks*)

**CPDAE** Auto-encodeur avec pooling par canal (*Chanel-wise Pooling Denoising Autoencoder*)

**CVC** Chauffage, Ventilation, Climatisation

**CWAP** Channel-Wise Average Pooling

**DDPG** Gradient profonde de politique déterministe (*Deep Deterministic Policy Gradient*)

**ECG** Électrocardiogramme

**EMFC** Compensation des forces électromagnétiques (*ElectroMagnetic Force Compensation*)

**FIR** Réponse impulsionnelle finie (*Finite Impulse Response*)

**GRU** Gated Recurrent Unit

**IA** Intelligence Artificielle

**IIR** Réponse impulsionnelle infinie (*Infinite Impulse Response*)

**KPSS** Kwiatkowski-Phillips-Schmidt-Shin

**LQR** Régulation quadratique linéaire (*Linear Quadratic Regulation*)

**LSTM** Réseau à mémoire à long et court terme (*Long Short-Term Memory*)

**MAE** Erreur Moyenne absolue (*Mean Absolute Error*)

**MAPE** Moyenne de l'erreur absolue en pourcentage (*Mean Absolute Percentage Error)*

**MILP** Programmation linéaire en nombres entiers mixtes (*Mixed Integer Linear Programming*)

**NLP** Traitement du langage naturel (*Natural Language Processing*)

**PILCO** Inférence probabiliste pour le contrôle de l'apprentissage (*Probabilistic Inference for Learning Control*)

**PPO** Optimisation de politique proximale (*Proximal Policy Optimization*)

**ReLu** Unité Linéaire Rectifiée (*Rectified Linear Unit*)

**RMSE** Racine de l'erreur quadratique moyenne (*Root Mean Square Error*)

**RNN** Réseau de neurones récurrent (*Recurrent Neural Network*)

**RSB** Rapport Signal/Bruit

**SARIMA** Moyenne mobile intégrée autorégressive saisonnière (*Seasonal Auto-Regressive Integrated Moving Average*)

**SARIMAX** Moyenne mobile intégrée autorégressive saisonnière avec entrées exogènes (*Seasonal Auto-Regressive Integrated Moving Average with exogenous inputs*)

**SVM** Machine à vecteur de support (*Support Vector Machine*)

**SVR** Régression à vecteur de support (*Support Vector Regression*)

**XGBoost** Boosting de gradient extrême (*eXtreme Gradient Boosting*)

## LISTE DES SYMBOLES

Symbole de la relation d'agrégation dans le diagramme de définition de blocs en SysML

Symbole de la relation d'association dans le diagramme de définition de blocs en SysML



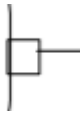
Symbole de la relation de composition dans le diagramme de définition de blocs en SysML

Symbole montrant le flux de contrôle qui se divise en flux parallèles dans le diagramme d'activités en SysML

Symbole désignant un point de décision dans le diagramme d'activités en SysML

Symbole indiquant la fin d'une activité dans le diagramme d'activités en SysML

Symbole représentant l'interruption d'une activité dans le diagramme d'activités en SysML



Symbole montrant l'entrée ou la sortie pour un flux d'objets ou de données dans le diagramme d'activités en SysML

# INTRODUCTION GÉNÉRALE

Le pesage industriel est présent dans de nombreux secteurs tels que l'agroalimentaire, la chimie ou la logistique. Cette technique est très utilisée, par exemple, pour faire du contrôle qualité, du suivi de production, du commerce de gros ou de la vérification dans le transport de marchandises. Dans tous ces secteurs, les données collectées lors du pesage sont stockées et analysées pour améliorer les procédés de production et en suivre les performances, ce qui permet de les optimiser et de détecter et corriger les erreurs.

Le pesage peut être réalisé de différentes façons : mécanique, électronique et hydraulique. Le pesage mécanique est réalisé à l'aide de ressorts ou de balances afin de mesurer la masse d'un objet ou d'un produit. Le pesage électronique utilise des capteurs de charge, appelés cellules de charge, qui vont permettre de mesurer la déformation induite par le poids. Il est utilisé dans de nombreux secteurs (Muller, Brito et al. 2010), car ce type de pesage est considéré comme très précis et fiable. Quant au pesage hydraulique, il se base sur des fluides sous pression pour mesurer la masse.

Le pesage industriel est donc présent dans de nombreux secteurs et est essentiel pour garantir la qualité et la quantité d'un produit ou pour permettre aux entreprises d'optimiser leurs processus de production en mesurant la quantité de matière utilisée par rapport à la quantité de produit fini. Cependant, les entreprises doivent se conformer à des normes et réglementations assez strictes. L'innovation a donc une place importante dans ce secteur pour améliorer la précision et la fiabilité de ces procédés et des mesures.

Concernant les récentes avancées dans le secteur industriel (Ribeiro, Lima et al. 2021), cela concerne autant la partie physique du capteur que la partie numérique de l'acquisition des données. Pour ce qui est des avancées physiques, les capteurs sont miniaturisés et de nouveaux matériaux sont étudiés et développés afin de fabriquer des

capteurs plus précis et plus résistants aux vibrations. Des capteurs intelligents sont mis en place pour améliorer, mais également intégrer directement un traitement des données. Les logiciels reliés à la chaîne d'acquisition sont améliorés pour avoir une meilleure gestion des données puis gérer et surveiller la quantité de matière pesée en temps réel.

L'intelligence artificielle (IA) est de plus en plus intégrée dans le pesage pour améliorer la précision et la fiabilité des mesures en agissant par exemple, sur la calibration ou la prédiction des données futures (Bahar and Horrocks 1998, Ribeiro, Lima et al. 2021). Elle sert également à détecter rapidement des irrégularités dans les mesures et à gérer les problèmes plus efficacement.

Le pesage électronique est utilisé dans l'ensachage de matières, où la quantité à ensacher est mesurée par pesage. Les cellules de charge connectées à une chaîne d'acquisition renvoient les informations de masse à l'automate programmable industriel (API), qui prend des décisions basées sur l'état du système et les informations perçues. Lors du traitement des données de capteurs, différentes étapes se succèdent afin de convertir les données brutes transmises par les capteurs en données exploitables par l'API et ainsi optimiser le cycle. Ces étapes sont la calibration, le filtrage des données brutes et l'optimisation des performances du système. Chaque étape peut nécessiter plusieurs itérations pour converger vers un résultat optimal lors de leurs mises en application. Ce nombre d'itérations dépend de la complexité du problème ou du niveau de bruit ambiant. Répéter ces itérations permet de progressivement trouver la meilleure optimisation.

La calibration des balances industrielles équipées de cellules de charge comprend d'abord une étape d'étalonnage des masses. Elle consiste à mesurer la masse de différents poids étalons et à établir une relation entre les masses de référence et les valeurs fournies en millivolts par volts par les capteurs de charge. Une fois l'étalonnage réalisé, une étape d'ajustage est réalisée afin d'avoir une répétabilité dans le temps et suivant des conditions environnantes changeantes. Cela comprend par exemple, le choix de la résolution de sortie, la correction des effets de la température, la compensation des effets gravimétriques et

l’ajustement de la plage de mesure. Ces ajustements des mesures visent à éviter les erreurs systématiques dans les mesures. Les erreurs systématiques affectent les mesures de manière constante et répétitive, et sont liées à la mesure faite par les capteurs. Un ajustement du réglage permet donc de diminuer cette erreur systématique et ainsi obtenir des mesures plus justes. Cependant, ces réglages ne suffisent pas à garantir que les mesures ne seront pas impactées par les bruits issus de l’environnement dans lequel se trouve la balance et peuvent donc être très variables de ceux compensés lors de la calibration. Ce type d’erreur est qualifié d’erreur aléatoire. L’utilisation de filtres appliqués au signal vient donc compléter la calibration afin de minimiser les impacts des conditions environnantes sur la mesure effectuée. Il s’agit donc de l’étape de filtrage.

Le filtrage des données brutes permet d’atténuer le bruit enregistré par les capteurs, et très présent dans l’environnement industriel. Cette étape améliore la qualité des mesures en réduisant les bruits environnants et en extrayant les informations pertinentes du signal. Dépendamment des résultats souhaités, différents types de filtrage peuvent être appliqués. Certains filtres sont conçus pour atténuer des fréquences spécifiques ou éliminer des valeurs aberrantes, tandis que d’autres méthodes plus complexes utilisent des modèles mathématiques pour approximer et lisser les données.

Une fois que les capteurs ont été calibrés et que les données bruitées ont été filtrées, il est possible d’optimiser les performances du système afin de le rendre plus rapide et fiable, suivant les exigences de performance demandées. Ces optimisations peuvent être réalisées en utilisant différentes méthodes qui sont adaptées en fonction de la complexité du système et des objectifs d’optimisation. L’optimisation est une étape essentielle qui permet d’avoir des systèmes de pesage plus performants et plus précis, mais elle peut demander plusieurs itérations avant d’être mise en place et requiert généralement une connaissance approfondie du système.

## OBJECTIFS ET CONTRIBUTION

Dans notre démarche pour intégrer l'IA à ce système, deux approches différentes ont été mises en place. La première approche consiste à optimiser le système en développant un modèle basé sur la prédiction de données. La seconde approche appliquée est composée de deux étapes et se base sur le fonctionnement courant des systèmes de pesage comme détaillé précédemment, mais les méthodes appliquées sont différentes. Elle comprend donc une étape de filtrage puis une étape d'optimisation des décisions prises par l'API. L'IA est testée et mise en place sur ces deux étapes. Pour les deux approches, une grande variabilité de modèles existe et plusieurs autres modèles qui ne sont pas présentés dans ce mémoire pourraient convenir pour résoudre le problème. Cependant, nous avons dû faire des choix parmi les modèles possibles et leurs paramètres testés, car l'optimisation des résultats peut prendre plusieurs mois. La Figure 1 résume la démarche appliquée pour résoudre notre problème.

Ainsi, dans le cadre de cette étude, l'IA est utilisée pour améliorer le système de pesage lié à l'ensachage de produits. Les données utilisées seront les données de masses bruitées, transmises par les cellules de charge. Cette amélioration va agir à différents moments du cycle de pesage afin de minimiser le temps d'un cycle et de garantir une fiabilité dans la précision malgré un environnement bruyant. En effet, les contrôles de poids étant soumis à de fortes réglementations, il est courant d'ajouter un surplus de sécurité à chaque sac afin de garantir leur conformité en cas de variabilités des mesures. Ces variabilités peuvent être causées autant par un mauvais paramétrage de la machine que par un changement de la densité du produit ou de son environnement. En suivant cette démarche, les objectifs de ce travail de recherche sont les suivants :

- Rédiger une revue de littérature des approches d'IA pouvant être envisagées pour le projet.

- Constituer une base de données de cycles de pesage pouvant servir à entrainer et tester des approches.

- Développer un modèle basé sur le signal brut et la prédiction pour optimiser le cycle de pesage.

- Développer un modèle qui combine le filtrage des données brutes et l'apprentissage par renforcement pour optimiser le cycle.

- Proposer des recommandations permettant à l'entreprise partenaire de continuer le projet.

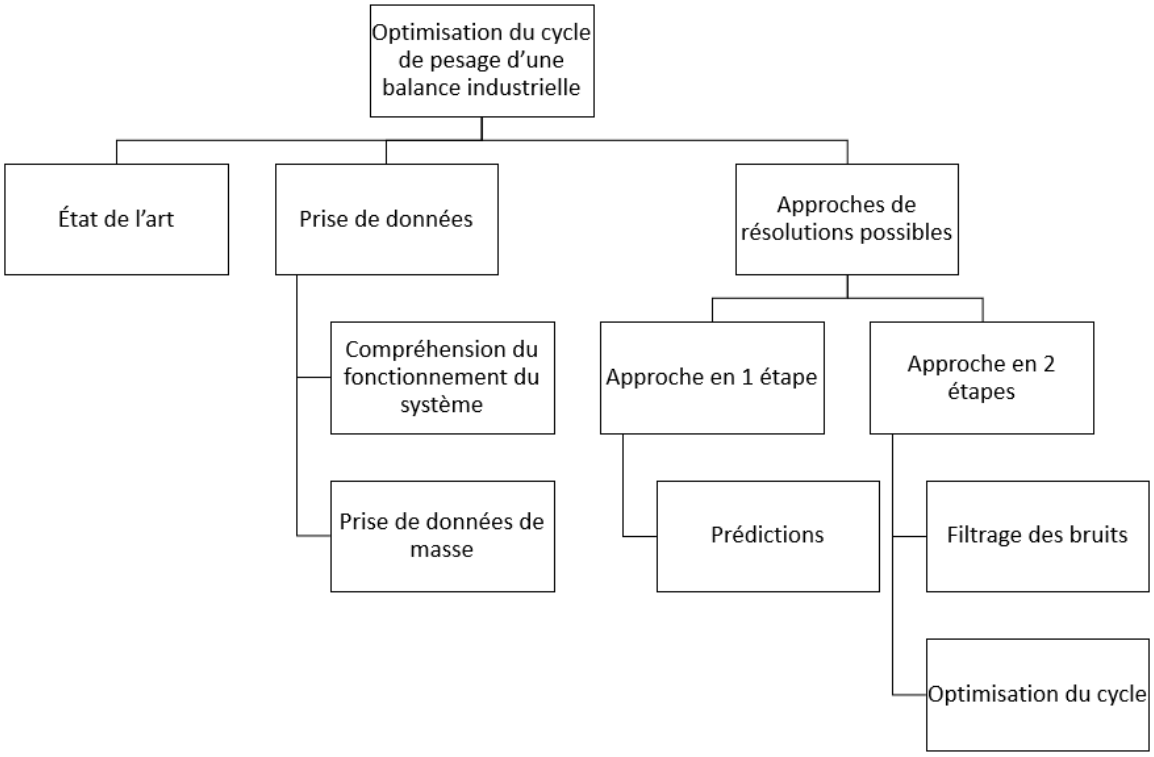



Figure 1. Méthodologie adoptée pour résoudre le problème étudié

La contribution de ce projet de recherche est de proposer une approche qui permet d'intégrer l'IA à un système de pesage qui n'en contient pas et dans le but d'optimiser l'ensachage des produits. Parmi les objectifs d'optimisation possibles, l'attention est mise sur la justesse du poids final et la réduction du temps de pesée. Ce projet permettra donc

d'identifier et de tester les modèles d'IA capables d'optimiser un système existant, mais également ceux qui ne sont pas adaptés à cette application. Ce projet et la stratégie employée pourront être réutilisés pour continuer d'améliorer et d'optimiser des systèmes industriels à l'aide de l'IA. Les résultats obtenus à la suite d'une première approche basée uniquement sur le poids brut, se sont montrés peu concluants, ce qui nous a amenés à la repenser. Bien que l'API actuel intègre une étape de filtrage, la demande de l'entreprise partenaire a conduit à la recherche d'une méthode basée sur le poids brut. La seconde approche développée a donc été divisée en deux étapes : la recherche d'une méthode de filtrage puis d'une optimisation basée sur l'apprentissage par renforcement. Nous avons testé des modèles d'IA pour le filtrage, mais les filtres numériques ont permis d'obtenir de meilleurs résultats. Concernant l'apprentissage par renforcement, les premiers résultats se sont montrés intéressants.

## ORGANISATION DU MÉMOIRE

Le mémoire est composé de cinq chapitres. Le premier chapitre permet de présenter les notions de base utilisées en traitement du signal, telles que les méthodes statistiques ou les filtres mathématiques. Dans ce même chapitre, des notions liées à l'IA sont ensuite présentées pour mettre en avant les concepts et les techniques des différents types d'apprentissage et modèles employés dans l'approche de résolution du problème. Le deuxième chapitre présentera l'état de l'art des méthodes employées pour la résolution des différentes tâches de filtrage et d'optimisation appliquées principalement dans un contexte industriel. Cela comprend autant des modèles utilisant l'IA, que des modèles s'appuyant sur des modèles mathématiques. Dans le chapitre suivant, le fonctionnement du système de pesage étudié et son utilisation dans son contexte seront décrits en détail. Ce troisième chapitre comprend également l'explication de la méthodologie employée pour résoudre le problème, ainsi que la façon dont a été constituée la base de données et ce qu'elle nous a permis de comprendre. Enfin, le quatrième et le cinquième chapitre regroupent respectivement, les résultats des approches de l'optimisation par prédiction et de celle par

filtrage et apprentissage par renforcement. Une conclusion du projet global est réalisée à la fin de ce mémoire.

# CHAPITRE 1
# NOTIONS DE BASE

Ce chapitre rassemble les notions fondamentales nécessaires à la compréhension des domaines du filtrage et de l'IA, et des concepts associés. Les techniques utilisées pour atteindre les objectifs du projet sont présentées de façon théorique, ainsi que d'autres informations pertinentes. Le chapitre commence par des notions de traitement du signal en abordant les différents types de filtres et en fournissant des exemples de filtres numériques. Ensuite, il aborde les concepts relatifs à l'IA, en présentant les méthodes de prédiction qui offrent des solutions potentielles au problème de filtrage du bruit. En conclusion, le chapitre détaille les méthodes d'optimisation employées dans le contrôle des systèmes, incluant les approches heuristiques et statistiques. L'apprentissage par renforcement, un pan important de l'IA, sera également utilisé dans la seconde stratégie développée.

## 1.1 FILTRAGE DES BRUITS

L'étape de filtrage est essentielle dans le traitement du signal. Elle sert à atténuer, supprimer ou faire ressortir certaines fréquences du signal en fonction des applications dans lesquelles le filtre est utilisé. En particulier dans le traitement des signaux issus de capteurs, le filtrage rend les mesures plus fiables en réduisant le bruit enregistré, qui est très présent dans les environnements industriels. Pour les données issues de cellules de charges, il est nécessaire avant d'effectuer d'autres étapes ou de prendre des décisions. Ce sont des capteurs très sensibles aux bruits. De plus, les applications industrielles ont des sources de bruits ambiants qui peuvent être multiples, comme les vibrations et les mouvements à proximité des capteurs. Certaines erreurs dues aux fluctuations de la température ou aux interférences

électriques peuvent être corrigées lors de la calibration, tandis que d'autres perturbations nécessitent un filtrage. Cependant, il est essentiel de distinguer le bruit et les perturbations indésirables des variations liées à la dynamique de mesure du remplissage. Dans ce dernier cas, un filtrage excessif pourrait nuire à la précision en supprimant des informations pertinentes sur l'évolution du remplissage. L'étape de filtrage est donc appliquée pour obtenir des mesures plus précises et fiables.

### 1.1.1 Les différents types de bruits

Comme évoqué précédemment, les signaux transmis par des cellules de charge sont bruités. Il existe plusieurs types de bruits qui peuvent être présents dans ces signaux. L'ensemble de ces bruits varient en fonction des conditions environnantes ou des autres composants du système. La liste des bruits présentés ci-dessous n'est pas exhaustive, mais regroupe ceux couramment observés dans ce type de signaux.

Le bruit de fond est induit par l'environnement autour des capteurs. Les cellules de charge peuvent percevoir un bruit dû aux vibrations causées par le fonctionnement d'une machine à proximité ou le déplacement d'un engin. Un autre type de bruit proche du bruit de fond est celui dû à la vibration mécanique des composants du système qui contient les cellules de charge.

Le bruit de repliement spectral se produit lorsque les fréquences sont supérieures à la fréquence de Shannon-Nyquist, ce qui correspond à la moitié de la fréquence d'échantillonnage. Ce phénomène peut être évité en plaçant un filtre analogique anti-repliement en amont du convertisseur analogique-numérique.

Le bruit thermique est très courant dans les composants électroniques. Il est aussi appelé « bruit de Johnson-Nyquist », et est causé par l'agitation thermique des électrons qui sont à l'intérieur du capteur et de ses conducteurs. Cette agitation augmente avec la

température et est également proportionnelle à la bande passante du système et à la résistance des conducteurs. Le bruit thermique est qualifié de bruit blanc, car il est présent à toutes les fréquences et ses fluctuations sont aléatoires.

Les interférences magnétiques et la conversion des signaux analogiques en signaux numériques peuvent également être des sources de bruits dans les données. Ainsi, cela permet de constater que les sources de bruits sont très variées et qu'il est difficile, voire impossible dans certains cas, d'agir sur les causes directement. Ces éléments permettent de confirmer que l'étape de filtrage est une étape essentielle dans le traitement du signal.

Afin de mesurer et quantifier la présence de bruit dans les données, le rapport signal sur bruit peut être utilisé. Il permet d'évaluer la qualité du signal en comparant les puissances du signal utile et du signal bruité. Ces puissances sont déterminées soit à l'aide d'appareils de mesure, soit calculées en prenant la moyenne des carrés des valeurs du signal ou du bruit, dépendamment de la puissance calculée. Sa valeur est donnée en décibels et il peut être calculé en utilisant la formule suivante :

$$RSB(dB) = 10 * log_{10}\left(\frac{P_s}{P_b}\right) , \qquad (1)$$

où $P_s$ est la puissance du signal utile et $P_b$ est la puissance du bruit.

Ainsi, plus la valeur de RSB est élevée, plus le signal est de bonne qualité et est faiblement bruité. Si au contraire sa valeur est faible, alors cela signifie que le signal est très bruité.

Le bruit et les perturbations dans les signaux issus de capteurs peut donc avoir de nombreuses causes. Cependant, il est à noter que le bruit blanc gaussien ne peut pas se corriger avec un filtre. Pour filtrer les perturbations, il existe différents types de filtres permettant de réaliser cette étape. Ces filtres sont séparés en deux grandes familles : les filtres analogiques et les filtres numériques.

### 1.1.2 Les filtres analogiques

Le filtrage analogique est réalisé par des circuits électriques qui agissent sur le signal en filtrant les fréquences indésirables. Il existe deux catégories de filtres analogiques : les filtres passifs et les filtres actifs. Les filtres passifs sont des éléments passifs tels que les résistances, les condensateurs ou les inductances. Ces composants sont sensibles et moins performants que les filtres actifs. Les filtres analogiques actifs sont des composants tels que les amplificateurs analogiques ou les transistors. Cependant, l'avantage principal des filtres analogiques est leur capacité à réduire le repliement spectral. Ils sont placés à côté du convertisseur analogique-numérique.

### 1.1.3 Les filtres numériques

Le filtrage numérique est réalisé en appliquant des opérations mathématiques ou des algorithmes à des échantillons du signal. Le choix du filtre dépend des caractéristiques du signal, du type de bruit à supprimer et des contraintes liées au temps de traitement en temps réel. Il existe trois grandes catégories de filtres numériques : les filtres à réponse impulsionnelle finie, les filtres à réponse impulsionnelle infinie et les filtres adaptatifs (Rabiner and Gold 1975).

Les filtres à réponse impulsionnelle finie ou FIR (en anglais : *Finite Impulse Response*) dépendent d'un nombre fini d'échantillons du signal fourni en entrée. Ils sont appliqués à des données issues de capteurs, car ils peuvent filtrer les données en temps réel, mais ils ajoutent beaucoup de phase. Les principaux paramètres de conception des filtres FIR incluent la taille de la fenêtre et les coefficients des filtres. Ces filtres offrent une stabilité en temps réel et une simplicité d'implémentation, mais la qualité du filtrage peut être limitée par la taille de la fenêtre et nécessiter un temps de latence constant.

Le filtre de Savitzky-Golay est un exemple de filtre FIR (Schafer 2011). Il lisse les données du signal en utilisant des polynômes ajustés sur une fenêtre glissante de valeurs successives. Les paramètres à définir sont la taille de la fenêtre de valeurs et l'ordre du polynôme appliqué sur les fenêtres. Ce filtre est efficace pour réduire le bruit et les changements brusques dans les données, mais il peut nécessiter plusieurs itérations pour trouver les bons paramètres. De plus, la phase de ce filtre doit être déduite car elle n'est pas connue lors de la conception de celui-ci.

Les filtres de moyenne mobile et de moyenne mobile exponentielle sont également des filtres FIR. Le filtre de moyenne mobile se base sur le calcul de la moyenne des valeurs comprises dans une fenêtre glissante (Smith 2003). Le principal hyperparamètre pour ce filtre est la taille de la fenêtre. Il est simple et rapide à mettre en place. Le filtre moyenne mobile exponentielle attribue davantage d'importance aux valeurs récentes, en utilisant un facteur de lissage comme hyperparamètre. Cela permet d'atténuer les fluctuations rapides du bruit tout en conservant les tendances du signal. Le filtre médian est un filtre non linéaire. Il utilise également une fenêtre glissante de données et calcule la médiane de ces valeurs (Justusson 1981). La valeur au centre de la fenêtre est remplacée par la valeur médiane obtenue à partir des données de la fenêtre. Ce filtre est efficace pour éliminer le bruit impulsionnel et préserver les contours du signal. Pour ces trois filtres, la qualité du filtrage dépend de la taille de la fenêtre utilisée, qui est une valeur généralement déterminée de manière exploratoire. Ainsi, si le signal varie sur le long terme, cette valeur peut s'avérer inadaptée pour réaliser un bon filtrage.

Les filtres à réponse impulsionnelle infinie ou IIR (en anglais : *Infinite Impulse Response*) utilisent le principe de la rétroaction (Ambardar 1995). La sortie produite par le filtre dépend à la fois des nouveaux échantillons qu'il reçoit en entrée et des précédents. Les paramètres importants pour les filtres IIR comprennent les coefficients de rétroaction et l'ordre du filtre. Ces filtres peuvent générer une instabilité numérique due à la résolution des nombres et l'accumulation d'erreurs.

Le filtre de Butterworth est un exemple de filtre IIR (Selesnick and Burrus 1998). Ses paramètres incluent la fréquence de coupure, l'ordre du filtre, la fréquence d'échantillonnage du signal et le type de filtre appliqué. Il atténue progressivement les fréquences indésirables. L'ordre du filtre permet de définir la pente d'atténuation et la fréquence de coupure détermine le seuil d'atténuation de la largeur de bande. Selon le type (passe-bas, passe-haut…), les fréquences supérieures ou inférieures à la fréquence de coupure seront atténuées.

Les filtres adaptatifs sont des filtres numériques capables de s'adapter en temps réel aux changements dans le signal (Diniz 1997). Pour pouvoir s'adapter aux variations, les coefficients sont modifiés en temps réel par rapport aux signaux perçus. Cette caractéristique le différencie des filtres présentés précédemment, qui ont des coefficients fixes. Les paramètres incluent l'algorithme d'adaptation des coefficients. Les exemples de filtres adaptatifs comprennent :

1. LMS (*Least Mean Squares*)(Diniz 1997) : Ce filtre ajuste ses coefficients pour minimiser l'erreur entre le signal filtré et un signal de référence. Il est simple à mettre en œuvre et s'adapte rapidement aux variations du signal.

2. RLS (*Recursive Least Squares*)(Diniz 1997) : Ce filtre offre une convergence plus rapide que le LMS en utilisant une approche récursive pour minimiser l'erreur quadratique moyenne. Il est plus complexe et nécessite plus de calculs.

3. Kalman Adaptatif (Diniz 1997) : Ce filtre adapte dynamiquement les paramètres du modèle pour les systèmes où les caractéristiques du signal varient. Il utilise des probabilités pour estimer les états futurs du signal et s'ajuste en conséquence.

Certains types de filtres peuvent être réalisés de façon analogique et numérique. Les filtres de fréquence de type passe-bas, passe-haut, passe-bande et coupe-bande peuvent être mis en œuvre avec des composants analogiques, mais également avec des filtres numériques, en fonction de l'application. Le filtre passe-bas sert à éliminer le bruit à haute fréquence et les variations rapides, laissant passer les basses fréquences. Le filtre passe-haut laisse passer

les hautes fréquences tout en atténuant les basses. Le filtre passe-bande permet de laisser passer les fréquences dans un intervalle prédéfini et d'atténuer les fréquences en dehors de celui-ci, tandis que le filtre coupe-bande fait l'inverse en atténuant les fréquences à l'intérieur de l'intervalle désigné.

### 1.1.4 Les séries temporelles

Les informations présentent dans cette section sont tirées du livre de Hyndman et Athanasopoulos (Hyndman and Athanasopoulos 2018).

Les séries temporelles représentent l'évolution d'une variable dans le temps et sont souvent utilisées pour analyser l'évolution des valeurs au cours du temps et anticiper les données futures afin de prendre des décisions. Elles sont présentes dans de nombreux domaines tels que la météorologie et la finance. Les données issues de capteurs et notamment les données de pesage sont aussi séries temporelles [1].

Les séries temporelles sont composées de la tendance, de la saisonnalité et des cycles. La tendance représente le mouvement général dans le temps, la saisonnalité les changements de comportement au cours d'une saison, et les cycles les variations irrégulières. La décomposition d'une série temporelle inclut généralement une composante tendance-cycle, une composante saisonnière et une composante de bruit.

Il existe deux catégories de séries temporelles : univariées et multivariées. Les séries univariées se basent sur une seule variable pour les prédictions, tandis que les séries multivariées incluent plusieurs variables qui dépendent du temps et interagissent entre elles.

[1] Les techniques d'IA sont employées pour analyser les données de pesage traitées comme une série temporelle, l'idée est de tester leur capacité à détecter et exploiter des relations complexes et des dépendances à long terme dans les données. Ils sont entrainés sur des données historiques de cycles pour comprendre les dynamiques de variation typiques du processus, permettant ainsi de prédire les comportements dans un cycle de remplissage en cours.

Les variables peuvent être endogènes (affectées par d'autres variables du modèle) ou exogènes (déterminées en dehors du modèle).

L'autocorrélation mesure la corrélation entre les valeurs d'une variable d'une même série temporelle à différents décalages temporels. Elle permet de déterminer la présence d'une tendance ou de données saisonnières. Elle est évaluée par la fonction d'autocorrélation. La stationnarité indique que les propriétés statistiques de la série (moyenne, variance, autocorrélation) sont constantes dans le temps. Les séries stationnaires sont plus faciles à modéliser et à prévoir. La stationnarité peut être vérifiée avec des tests statistiques comme le test de Dickey Fuller Augmenté (ADF) et le test KPSS.

Les modèles ARIMA et ses variantes (ARIMAX, SARIMA, SARIMAX) sont couramment utilisés pour la modélisation et la prédiction de séries temporelles. Ces modèles prennent en compte des composantes autorégressives, de moyenne mobile et, pour certains, des variables exogènes.

- ARIMA (*AutoRegressive Integrated Moving Average*) : Utilisé pour les séries non stationnaires sans variations saisonnières. Il combine des termes autorégressifs (AR), des différences intégrées (I) pour rendre la série stationnaire, et des moyennes mobiles (MA).

- ARIMAX (*AutoRegressive Integrated Moving Average with eXogenous variables*) : Inclut des variables exogènes pour améliorer les prédictions, en ajoutant des termes qui représentent l'influence de facteurs externes.

- SARIMA (*Seasonal AutoRegressive Integrated Moving Average*) : Utilisé pour les séries avec tendances et variations saisonnières, en ajoutant des composantes saisonnières aux termes ARIMA.

- SARIMAX (*Seasonal AutoRegressive Integrated Moving Average with eXogenous variables*) : Combinaison de SARIMA et ARIMAX, incluant des variables exogènes pour des données périodiques. Il ajoute les influences externes aux composantes saisonnières et non saisonnières du modèle.

Les prédictions des modèles ARIMA oscillent autour de la valeur moyenne des données, tandis que les modèles ARIMAX prennent en compte des variables exogènes, comme le point de consigne de fermeture de la trémie, impactant les poids finaux. Cependant, les modèles plus complexes ne sont pas adaptés pour des prédictions en temps réel en raison de la grande variabilité des données et des temps de calcul élevés. Les prédictions peuvent être effectuées en temps masqué lors de l'exécution de la pesée.

Dans notre domaine d'application, notre hypothèse de base est que les modèles ARIMA et ses variantes peuvent être utilisés pour prédire les poids finaux des cycles de pesée en se basant sur les derniers poids bruts enregistrés à la fin de chaque pesée avant l'évacuation du produit. Ces poids formeraient une nouvelle série temporelle, permettant ainsi de prédire les poids finaux des cycles suivants.

Nous posons aussi l'hypothèse que le traitement des données de séries temporelles ainsi que la prédiction peuvent être réalisés à l'aide de l'IA. Dans la suite du chapitre, le domaine de l'IA et de ses applications possibles sur ce type de données est donc détaillé.

## 1.2 APPRENTISSAGES AUTOMATIQUE ET PROFOND

Une grande partie des informations présentes dans cette partie sont tirées du livre de Goodfellow et al. (Goodfellow, Bengio et al. 2016).

En 1956, lors d'une conférence au Dartmouth College (États-Unis), le terme « intelligence artificielle » est employé pour la première fois (McCarthy, Minsky et al. 2006). Des chercheurs souhaitaient trouver un moyen de modéliser l'intelligence. Par la suite, de nombreux concepts tels que l'apprentissage automatique et les réseaux de neurones ont été développés. Cependant, les dernières décennies ont permis de faire de grandes avancées dans cette discipline grâce aux progrès faits dans les puissances de calcul et la mise à disposition de quantités de données plus importantes.

L’IA est maintenant utilisée dans tous les domaines : médecine, industrie, finance ou encore dans les transports. Ses applications dans chacun de ces domaines sont très variées. En médecine, elle permet d’aider aux diagnostics ou d’être d’une précieuse aide pour des opérations. Quant à l’industrie, elle permet d’automatiser des tâches répétitives et contraignantes physiquement ou d’améliorer des processus de production. Elle est également utilisée pour le filtrage de données numériques, car elle permet d’identifier les informations pertinentes du signal.

Le principe de l’IA est de créer des systèmes qui sont capables de reproduire des comportements de l’intelligence humaine, comme comprendre, raisonner et interagir. Cela est possible, car les modèles sont capables d’analyser et de traiter de grandes quantités de données. L’IA regroupe différentes approches et modèles qui permettent de répondre à ces objectifs et peut être divisée en sous-domaines imbriqués. Ces sous-domaines sont entre autres l’apprentissage automatique, l’apprentissage de la représentation et l’apprentissage par renforcement, tels que représentés dans la Figure 2.

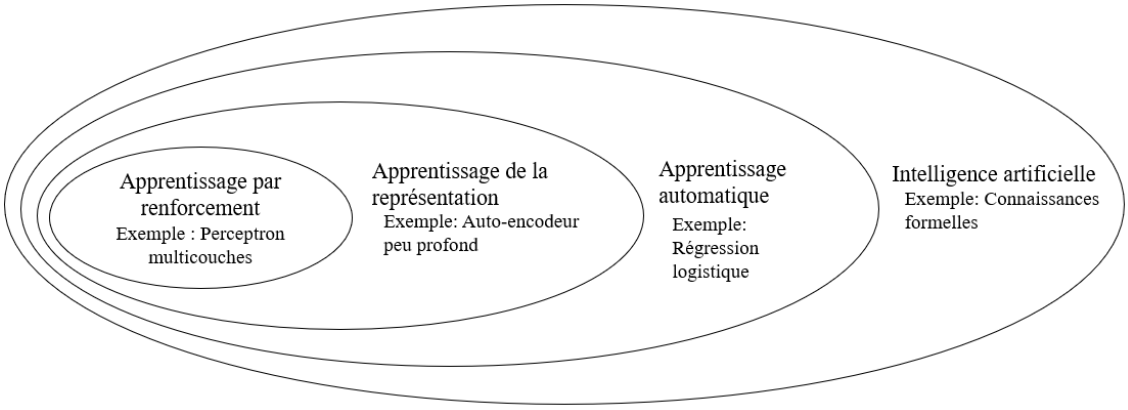



Figure 2. Diagramme de Venn regroupant les domaines de l’IA (traduit de : Deep Learning, (Goodfellow, Bengio et al. 2016))

L’apprentissage automatique (en anglais : *Machine Learning*) est un sous-domaine de l’IA qui permet aux machines d’apprendre et de trouver des relations ou des modèles cachés

dans des données fournies. Pour apprendre, un modèle d'apprentissage automatique reçoit des données d'entrées et les réponses attendues en sorties. Il ajuste ses paramètres en fonction des relations trouvées, ce qui lui permet de faire des prédictions lorsqu'il est confronté à de nouvelles données qu'il n'a jamais vu. Parmi les modèles d'apprentissage automatique, il y a les réseaux de neurones et les machines à vecteurs de support, qui seront présentés plus en détail dans la suite de ce chapitre.

L'apprentissage de la représentation est un sous-domaine de l'apprentissage automatique. L'objectif de cet apprentissage est d'apprendre de lui-même des caractéristiques significatives dans les données. Pour cela, il crée de façon automatique et hiérarchique des représentations des données d'entrées qui lui permettent de comprendre la structure des données et les informations importantes.

L'apprentissage profond (en anglais : *Deep Learning*) apprend à partir de réseaux de neurones artificiels profonds qui imitent le fonctionnement du cerveau. Ils utilisent des couches successives de neurones qui sont connectés entre eux. Il est très utilisé pour faire de la reconnaissance vocale, de la traduction automatique ou encore de la vision par ordinateur. L'inconvénient de cet apprentissage est qu'il nécessite d'avoir un grand ensemble de données pour s'entrainer, ce qui implique des ressources informatiques de calculs importantes. Ainsi, l'apprentissage profond est un domaine de l'apprentissage automatique. La différence est que l'apprentissage profond utilise des réseaux de neurones profonds pour apprendre.

### 1.2.1 Les différents types d'apprentissage

L'IA comprend trois types d'approches d'apprentissage distinctes. Ces approches sont l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. En fonction du problème à résoudre et de la forme des données à disposition, une approche peut être choisie par rapport à une autre.

L’apprentissage supervisé est utilisé dans de nombreuses applications. Les modèles entrainés dans le cadre de cet apprentissage se basent sur des couples de données entrées-sorties. Il s’agit donc de données étiquetées qui sont transmises au modèle pour qu’il apprenne à partir d’exemples afin de réaliser des tâches comme la prédiction ou la classification.

L’apprentissage non supervisé se base sur des données non étiquetées en entrées et le modèle doit apprendre à détecter des modèles cachés dans les données sans aucune indication préalable. Il n’y a donc pas de bonnes ou mauvaises réponses, contrairement à l’apprentissage supervisé. Cet apprentissage est souvent utilisé pour segmenter des données, détecter des anomalies, recommander des contenus ou faire de la réduction de dimensionnalité. On parle parfois d’apprentissage semi-supervisé, il s’agit d’une combinaison de données étiquetées et non-étiquetées.

Ces deux types d’apprentissages ne prennent pas directement de décisions, ils font des prédictions puis un agent externe permet de prendre des décisions en fonction des prédictions faites.

Dans l’apprentissage par renforcement, un agent apprend à prendre des décisions en interagissant avec son environnement. L’agent choisit des actions menant à des états qui dépendent de son environnement. En fonction de l’état dans lequel se retrouve l’agent, il va obtenir des récompenses positives ou négatives. Son but est de trouver la bonne succession d’actions qui lui permettent de maximiser ses récompenses. Des modèles issus des autres types d’apprentissage peuvent être utilisés pour entrainer l’agent dans ses prises de décisions.

Les réseaux de neurones artificiels sont utilisés dans le domaine de l’IA pour résoudre différents problèmes. Il s’agit d’un modèle computationnel qui a été créé de façon à imiter le fonctionnement du cerveau. Ces réseaux sont composés de différentes couches qui comportent des neurones et divisées en trois catégories : la couche d’entrée, les couches cachées et la couche de sortie. La couche d’entrée est celle qui reçoit les données à comprendre. Le nombre de neurones sur cette couche dépend du nombre de caractéristiques

qu'ont les entrées. Les couches cachées vont traiter les informations qui viennent de la couche d'entrée pour extraire les caractéristiques des données. Ces couches sont composées de plusieurs neurones qui sont connectés avec les neurones des autres couches cachées. Plus il y a de couches, plus le réseau est capable d'apprendre des informations complexes. La couche de sortie permet d'obtenir les résultats issus du réseau de neurones. Ces réseaux sont très employés, car ils sont capables de réaliser des tâches complexes et d'apprendre des informations à partir d'une grande quantité de données.

Des modèles utilisant des réseaux de neurones et étant couramment utilisés pour des séries temporelles, sont détaillés dans la suite de ce mémoire. L'IA comprend d'autres approches qui n'utilisent pas de réseaux de neurones, comme les arbres de décision ou les machines à vecteurs de support. Un modèle de machine à vecteurs de support et un modèle d'arbre de décision, qui sont utilisés dans le domaine des séries temporelles, seront également détaillés dans la suite de ce mémoire.

### 1.2.2 Techniques d'apprentissage automatique

Parmi les différentes méthodes d'apprentissage automatique qui ont été introduites au fil des ans, les modèles suivants sont des méthodes classiques utilisées pour supporter des données de séries temporelles. Parmi les différentes méthodes d'apprentissage automatique, les modèles suivants sont des techniques classiques utilisées pour les données de séries temporelles.

Les machines à vecteurs de support (SVM) sont utilisées pour des tâches de classification et de régression (Smola and Schölkopf 2004). En classification, les SVM cherchent à déterminer l'hyperplan qui sépare le mieux les classes d'échantillons en maximisant la marge entre les vecteurs de support et l'hyperplan. Lorsqu'il s'agit de régression, on utilise la *Support Vector Regression* (SVR) (Awad and Khanna 2015), qui

définit une fonction approximant les données tout en minimisant l'écart par rapport à celles-ci.

Les principaux avantages de la SVR incluent sa capacité à gérer des relations non linéaires grâce à l'utilisation de noyaux, ainsi que de bonnes performances sur des données complexes et bruitées. Les paramètres ajustables, tels que la régularisation (paramètre $C$), le noyau (paramètre $\gamma$) et la marge de tolérance (paramètre $\varepsilon$), permettent une flexibilité et une adaptation à divers cas d'utilisation. Le choix des paramètres peut être complexe et demande souvent de nombreux tests pour optimiser le modèle. Toutefois, contrairement aux modèles d'apprentissage profond, la SVR, à l'instar des autres techniques d'apprentissage automatique, est relativement plus facile à interpréter, bien que les ajustements nécessaires puissent toujours représenter un défi.

XGBoost (*eXtreme Gradient Boosting*) est un modèle basé sur des arbres de décision, utilisé pour des tâches de classification et de régression. Il est couramment utilisé sur des séries temporelles (Chen and Guestrin 2016). Il utilise une technique de *boosting* pour améliorer les prédictions en corrigeant progressivement les erreurs des arbres précédents. XGBoost optimise les paramètres via des méthodes d'optimisation du gradient et gère automatiquement les valeurs manquantes.

Les principaux avantages de XGBoost incluent sa haute performance et la précision des prédictions grâce à la régularisation intégrée, ainsi que sa capacité à gérer les valeurs manquantes sans intervention supplémentaire. Il est également bien adapté aux séries temporelles et capable de capturer des relations complexes dans les données. Les hyperparamètres essentiels de XGBoost comprennent la profondeur maximale de l'arbre, le taux d'apprentissage et l'objectif d'apprentissage. La profondeur maximale de l'arbre détermine le nombre maximal de niveaux que l'arbre de décision peut atteindre. Le taux d'apprentissage définit l'impact de chaque arbre sur le modèle final, et l'objectif d'apprentissage spécifie la fonction de perte à minimiser.

Le Tableau 1 résume les méthodes discutées, telles que le SVM/SVR et XGBoost, en indiquant leurs hyperparamètres, avantages et inconvénients.

Tableau 1. Synthèse des méthodes d'apprentissage automatique

| Nom | Hyperparamètres | Avantages | Inconvénients |
|---|---|---|---|
| SVM/SVR | C (régularisation), γ (noyau), ε (tolérance) | Gère les relations non linéaires, robuste aux données bruitées | Difficile d'ajuster les hyperparamètres, interprétation complexe des résultats |
| XGBoost | Profondeur maximale, taux d'apprentissage, nombre d'estimations | Précision élevée, gestion automatique des valeurs manquantes | Exige beaucoup de ressources, temps de calcul élevé |

Pour appliquer et évaluer ces modèles, il est essentiel de diviser les données en ensembles d'entrainement, de validation et de test. La validation croisée permet de tester différentes combinaisons de paramètres et d'affiner le modèle. Les techniques de recherche systématique ou aléatoire aident à trouver la meilleure configuration d'hyperparamètres. Dans nos applications, la SVR est utilisée pour filtrer les données en temps réel, en éliminant le bruit des signaux issus des capteurs. Les paramètres doivent être bien ajustés pour garantir la qualité du filtrage. XGBoost, quant à lui, est utilisé pour des prédictions sur des valeurs continues dans des séries temporelles. Il est essentiel de l'entrainer en amont pour réduire le temps de traitement en temps réel. XGBoost peut gérer des valeurs manquantes et intégrer des techniques de régularisation pour améliorer la précision des prédictions.

En conclusion, ces modèles d'apprentissage automatique peuvent donc être utilisés pour prédire des données temporelles lissées. L'avantage de ces modèles est qu'ils ont divers hyperparamètres dont les actions sur l'apprentissage sont facilement compréhensibles. Cependant, trouver la bonne combinaison de paramètres peut être compliqué et nécessiter plusieurs itérations. La justesse du filtrage dépendra de ses paramètres, mais également de la qualité des données d'entrainement des modèles. En effet, il faut que ces données soient les plus représentatives de la réalité, car ces modèles ne sont pas conçus pour ajuster leurs

paramètres en temps réel lorsqu'ils reçoivent de nouvelles données. Il est donc important de surveiller de façon régulière les performances du modèle et de faire de nouveaux entrainements si les données évoluent dans le temps. Ce phénomène où les données réelles ont évolué et ne correspondent plus aux données d'entrainement s'appelle le phénomène de dégradation. Ces limitations sont donc à prendre en compte dans le cas du filtrage de données en temps réel. Si les données n'ont pas de changements significatifs, alors ces modèles permettent d'obtenir de bons résultats, même s'il s'agit de données complexes.

### 1.2.3 Architectures d'apprentissage profond

Les réseaux de neurones récurrents (RNN) sont conçus pour la modélisation de séquences de données en utilisant une mémoire interne. Ils traitent les données séquentielles en mettant à jour un état caché à chaque pas de temps. Cependant, ils rencontrent des difficultés avec les dépendances à long terme et les problèmes de gradient. Le problème de gradient qui disparaît survient lors de la rétropropagation, limitant la capacité des RNN à apprendre sur de longues séquences. Pour pallier ce problème, des variantes comme les réseaux LSTM (*Long Short-Term Memory*) et GRU (*Gated Recurrent Unit*) intègrent des mécanismes de portes pour réguler le flux d'informations et la propagation des gradients, permettant de capturer des dépendances à long terme.

Les LSTM sont très utilisés pour le traitement du langage naturel et la prédiction de séries temporelles (Hochreiter and Schmidhuber 1997). Ils peuvent capturer des motifs complexes et des dépendances à long terme, ce qui les rend adaptés aux tâches appliquées aux données séquentielles. Les LSTM utilisent des cellules de mémoire et trois types de portes (entrée, oubli, sortie) pour réguler les flux d'informations. Ils peuvent être superposés en plusieurs couches pour apprendre des relations plus complexes. Les hyperparamètres importants pour les LSTM incluent le nombre d'unités de mémoire, le taux d'apprentissage, la taille des lots et le nombre d'époques. Le nombre d'unités de mémoire détermine la capacité du réseau à capturer des informations à long terme, tandis que le taux

d'apprentissage influence la vitesse à laquelle le modèle converge. La taille des lots affecte la stabilité de l'entraînement et le nombre d'époques détermine combien de fois le modèle voit l'ensemble des données. Cependant, les LSTM nécessitent de grandes quantités de données et un temps d'entraînement conséquent. De plus, comme pour la plupart des réseaux de neurones, il est difficile d'interpréter leurs résultats.

Les réseaux LSTM avec couche d'attention améliorent les performances en accordant plus d'importance à certaines parties des séquences de données (Vaswani, Shazeer et al. 2017). La couche d'attention pondère les sorties des LSTM pour donner plus d'importance aux informations pertinentes. Les hyperparamètres clés incluent ceux des LSTM ainsi que les mécanismes d'attention, comme le nombre de têtes d'attention et la dimension de l'espace latent. Le nombre de têtes d'attention permet au modèle de se concentrer sur différentes parties des données simultanément, tandis que la dimension de l'espace latent détermine la complexité des représentations internes. Bien que cela améliore les performances, ce modèle nécessite également des ressources informatiques adaptées et un temps d'entraînement prolongé. Les poids d'attention peuvent être facilement interprétés, rendant cette couche plus compréhensible. Cependant, une mauvaise interprétation des données peut affecter les poids d'attention et le résultat final.

Les réseaux de neurones convolutifs (CNN) sont très utilisés pour la vision par ordinateur, mais peuvent également être appliqués aux séries temporelles. Les CNN utilisent des couches de convolution pour extraire des motifs ou des caractéristiques importantes des séries temporelles. Les couches de sous-échantillonnage (*max pooling* et *average pooling*) permettent de réduire la dimension des données tout en conservant les informations essentielles. Les hyperparamètres essentiels des CNN incluent la taille du noyau de convolution, le nombre de filtres, la fonction d'activation et la taille des lots. La taille du noyau de convolution détermine la fenêtre d'observation pour l'extraction de caractéristiques, tandis que le nombre de filtres indique combien de motifs différents le réseau peut apprendre. La fonction d'activation, comme ReLU, introduit des non-linéarités dans le modèle, et la taille des lots influence la stabilité et la vitesse de l'entraînement. Les

CNN sont capables de reconnaître des motifs récurrents, mais peuvent perdre des informations importantes lors du sous-échantillonnage et ne sont pas adaptés aux variations à long terme. Une grande quantité de données est nécessaire pour entraîner les modèles CNN.

Les encodeurs-décodeurs sont des architectures composées de deux parties : l'encodeur et le décodeur (Bank, Koenigstein et al. 2023). Ils peuvent être constitués de réseaux de neurones récurrents ou convolutifs et sont utilisés pour des tâches de séquence à séquence, comme la traduction ou la prédiction de séries temporelles. L'encodeur transforme les données d'entrée en une représentation compacte, tandis que le décodeur utilise cette représentation pour générer la séquence de sortie. Les hyperparamètres clés incluent ceux des réseaux sous-jacents (RNN, LSTM, CNN) ainsi que des paramètres spécifiques à l'architecture d'encodage-décodage, comme la dimension de l'espace latent et le nombre de couches d'encodeurs et de décodeurs. La dimension de l'espace latent influence la richesse des représentations apprises, et le nombre de couches détermine la profondeur du modèle. Les encodeurs-décodeurs combinent les avantages des mécanismes d'encodage-décodage avec ceux des réseaux de neurones, mais sont sensibles aux données manquantes ou aberrantes. Ils nécessitent une grande quantité de données d'entraînement et des ressources informatiques suffisantes.

Le Tableau 2 résume les méthodes discutées, telles que les RNN, LSTM, LSTM avec couche d'attention, CNN et encodeur-décodeur, en indiquant leurs hyperparamètres, avantages et inconvénients.

En résumé, les architectures d'apprentissage profond, bien qu'offrant de nombreuses capacités avancées pour le traitement des données séquentielles, présentent également des défis en termes de complexité, de besoins en données et en ressources informatiques. La sélection et l'ajustement des hyperparamètres sont cruciaux pour optimiser les performances de ces modèles.

Tableau 2. Synthèse des méthodes d'apprentissage profond

| Nom | Hyperparamètres | Avantages | Inconvénients |
|---|---|---|---|
| RNN | Nombre d'unités cachées, taux d'apprentissage, taille des lots, nombre d'époques | Capacité à modéliser des séquences temporelles | Problèmes de gradient, difficulté avec les dépendances à long terme |
| LSTM | Nombre d'unités LSTM, taux d'apprentissage, taille des lots, nombre d'époques | Gère les dépendances à long terme, robuste pour les données séquentielles | Nécessite beaucoup de données, temps d'entraînement long |
| LSTM avec couche d'attention | Nombre d'unités LSTM, nombre de têtes d'attention, dimension de l'espace latent | Améliore les performances de LSTM, focus sur les informations pertinentes | Complexité accrue, temps d'entraînement prolongé |
| CNN | Taille du noyau de convolution, nombre de filtres, fonction d'activation, taille des lots | Extrait des motifs complexes, détecte les caractéristiques locales | Perte d'informations possibles avec le pooling, nécessite beaucoup de données |
| Encodeur-Décodeur | Nombre de couches, dimension de l'espace latent, taux d'apprentissage, taille des lots | Gère les séquences à séquences, flexibilité pour différentes tâches | Sensible aux données manquantes, difficile à interpréter, nécessite des ressources élevées |

## 1.3 L'OPTIMISATION DES SYSTÈMES

Plusieurs méthodes existent pour optimiser un système. Dans le cas des systèmes électromécaniques, cela peut inclure une reconception de la structure mécanique ou des changements de composants électroniques. Par exemple, il est possible d'analyser les frottements pour choisir des matériaux qui les réduisent, ou de sélectionner des composants et capteurs plus adaptés et performants. Une autre approche consiste à optimiser le système de contrôle de la machine, qui assure la coordination entre les différents composants du

système et garantit la bonne exécution du processus. Cette optimisation vise à maintenir un fonctionnement optimal en comparant les résultats obtenus avec les résultats souhaités. Elle ajuste les paramètres et régule le système pour atteindre les objectifs de précision et de fiabilité prédéfinis. L'optimisation peut inclure l'utilisation de techniques avancées, telles que l'intégration de capteurs intelligents ou de l'IA. Notre étude n'implique pas de reconception mécanique ou électronique. Les techniques et approches détaillées par la suite se limitent donc à l'optimisation du système de prise de décision. Ainsi, un système électromécanique peut être optimisé en utilisant des approches de commande, des techniques d'optimisation ou en combinant ces deux méthodes.

Les approches de commande visent à optimiser le système en ajustant ses actions en temps réel. Ces méthodes permettent de mettre en place un algorithme qui régule et contrôle le comportement du système par rapport aux données perçues et aux performances souhaitées. Cet ajustement se fait en temps réel et la commande peut agir sur différentes variables du système.

Il existe diverses techniques d'optimisation permettant de trouver des solutions pour améliorer l'exécution du cycle et le rendre plus performant. Cette optimisation est réalisée en cherchant les valeurs des paramètres et des autres variables réglables du système, permettant d'atteindre les objectifs souhaités. Ainsi, ces techniques d'optimisation évaluent différentes configurations parmi les paramètres. Les techniques utilisées pour trouver la meilleure solution peuvent être des méthodes heuristiques ou de la programmation dynamique.

Ces deux familles de méthodes d'optimisation peuvent être appliquées à des systèmes comportant déjà une commande séquentielle. Les algorithmes des techniques d'optimisation s'appuient sur le programme existant pour trouver la meilleure configuration de paramètres. Tandis que les approches de commande nécessitent de modifier ou de remplacer le programme de contrôle, car leur objectif est de réajuster le programme de contrôle pour le rendre plus efficace.

Les sections suivantes détaillent deux grandes familles de méthodes d'optimisation : les méthodes classiques d'optimisation et l'apprentissage par renforcement. Chacune de ces familles offre des outils et techniques spécifiques pour améliorer les performances.

### 1.3.1 Les méthodes classiques d'optimisation

Les méthodes heuristiques reposent sur des règles empiriques ou des techniques d'optimisation basées sur des connaissances spécifiques du domaine d'application (Richalet, Rault et al. 1978). Elles incluent des méthodes de seuils, qui déclenchent des actions lorsqu'une valeur prédéfinie est atteinte, et des méthodes de taux de variation, qui prennent des décisions basées sur les variations observées dans les variables du système. Ces techniques simplifient la prise de décision et permettent une optimisation rapide et facile à mettre en place, particulièrement dans les systèmes complexes. Cependant, leur capacité d'optimisation est limitée et elles peuvent ne pas être adaptées aux systèmes nécessitant des ajustements plus sophistiqués.

La programmation dynamique résout un problème en le divisant en sous-problèmes, résolvant chacun individuellement avant de combiner les solutions pour résoudre le problème principal (Bellman 1966). Cette méthode optimise chaque sous-problème pour améliorer le problème global. Les hyperparamètres clés incluent le nombre d'états et d'actions, ainsi que les critères de convergence. Elle est efficace pour les problèmes d'optimisation avec de nombreux états ou actions, à condition que le modèle reste stable. Elle convient particulièrement aux systèmes stables, mais peut être utilisée pour des systèmes avec des événements aléatoires via la programmation dynamique stochastique, qui utilise des probabilités pour la prise de décision. Cependant, cette méthode peut devenir inefficace pour des problèmes avec trop d'états et d'actions en raison du temps de calcul élevé requis.

Comme discuté précédemment, le filtre de Kalman est un observateur d'état. Il comprend deux étapes principales : la prédiction basée sur le modèle existant et la correction

basée sur les nouvelles données (Kalman 1960). Lors de la prédiction, la valeur future est estimée en se basant sur le modèle. À mesure que de nouvelles données deviennent disponibles, un gain est calculé pour ajuster les prédictions, pondérant l'importance des nouvelles informations par rapport aux prédictions du modèle. Les hyperparamètres incluent les matrices de covariance et les gains d'ajustement. Le filtre de Kalman est adapté aux systèmes linéaires, mais des variantes existent pour gérer les non-linéarités. Le filtre de Kalman étendu (Julier and Uhlmann 1997) est capable de gérer des valeurs non-linéaires en les linéarisant à chaque pas de temps grâce aux matrices jacobiennes. Il est efficace et précis si les non-linéarités ne sont pas trop fortes. Le filtre de Kalman non-linéaire (en anglais, *unscented Kalman filter*) (Wan and Van Der Merwe 2000), est plus précis que le filtre étendu, notamment dans les systèmes où les non-linéarités sont importantes. Le filtre de Kalman adaptatif (Rutan 1991) ajuste dynamiquement les paramètres du modèle et est donc utilisé lorsque les caractéristiques du signal varient.

En conclusion, ces techniques d'optimisation classiques offrent diverses approches pour optimiser et contrôler des systèmes complexes. Le choix de la méthode dépend des caractéristiques du système à traiter, de la complexité des données, et des besoins en termes de rapidité et de précision des ajustements. La sélection et l'ajustement des hyperparamètres sont cruciaux pour optimiser les performances de chaque méthode.

### 1.3.2 Apprentissage par renforcement

L'apprentissage par renforcement repose sur un agent qui interagit avec son environnement en choisissant des actions à effectuer (Kaelbling, Littman et al. 1996). Après chaque action, l'agent perçoit l'état de l'environnement, qui peut être physique (comme un robot industriel) ou virtuel (comme un simulateur). L'état de l'environnement, représenté à un instant donné, peut être discret ou continu. Dans un environnement à états discrets, il y a un nombre limité d'états possibles clairement définis, tandis qu'un environnement à états

continus permet une infinité de configurations possibles. Le choix entre un environnement à état continu ou discret dépend de la nature du problème et de sa modélisation.

Lorsque l'agent prend une action, il reçoit des récompenses de l'environnement, positives, négatives ou nulles, selon les résultats obtenus. L'objectif de l'agent est de maximiser les récompenses cumulées au fil du temps. Ces récompenses aident l'agent à évaluer ses actions et à ajuster ses décisions futures. La politique de l'agent, définie comme la stratégie qu'il utilise pour choisir ses actions en fonction de l'état de l'environnement, peut être déterministe (attribuant une action spécifique à chaque état) ou probabiliste (assignant une distribution de probabilités aux actions possibles pour chaque état).

Pour choisir les meilleures actions et maximiser les gains à long terme, l'agent utilise des fonctions de valeur qui estiment le gain total possible à partir d'un état ou d'une action donnée en suivant une politique. Les équations de Bellman (Bellman 1966) sont fondamentales pour les processus de décision de Markov et les algorithmes d'apprentissage par renforcement. Elles permettent de calculer les valeurs optimales des états et des actions en réalisant plusieurs itérations pour converger vers des valeurs stables.

L'agent doit équilibrer l'exploration (essayer de nouvelles actions pour découvrir de nouvelles informations sur l'environnement) et l'exploitation (utiliser les actions connues pour maximiser les récompenses). Une politique d'exploration favorise les actions moins sélectionnées, permettant à l'agent d'explorer l'environnement, tandis qu'une politique d'exploitation privilégie les actions ayant donné de bonnes récompenses précédemment.

Les problèmes d'apprentissage par renforcement peuvent être classés en problèmes de prédiction et de contrôle[2]. Les problèmes de prédiction visent à estimer les valeurs des états ou des actions pour évaluer la performance d'une politique existante, tandis que les

[2] La traduction adéquate de « control » en français est commande. On utilise dans ce mémoire le terme « contrôle » car il est très employé dans la littérature sur l'apprentissage par renforcement écrite en français.

problèmes de contrôle cherchent à déterminer la politique optimale pour maximiser les récompenses à long terme.

Il existe deux approches principales dans l'apprentissage par renforcement : basées sur des modèles et sans modèle. Les approches basées sur des modèles utilisent des interactions préprogrammées entre actions et états pour trouver une solution, tandis que les approches sans modèle apprennent par essais et erreurs, adaptant leur comportement en fonction des récompenses perçues. Le Tableau 3 donne des exemples de modèles d'apprentissage par renforcement, classés selon qu'ils utilisent ou non un modèle et s'ils sont destinés à la prédiction ou au contrôle.

Tableau 3. Exemples de modèles (adapté et étendu de https://towardsdatascience.com/reinforcement-learning-made-simple-part-2-solution-approaches-7e37cbf2334e)

| **Approche** | **Méthode** | **Description** | **Avantages** | **Inconvénients** |
|---|---|---|---|---|
| Basé sur un modèle | Programmation dynamique par évaluation de politique (Bertsekas 2012) | Évalue la valeur de chaque état sous une politique fixe. | Simple à implémenter pour petits espaces d'état. | Inefficace pour grands espaces d'état. |
| | Programmation dynamique par itération politique (Bertsekas 2012) | Alternance entre évaluation de politique et amélioration de politique. | Converge vers la politique optimale. | Peut être computationnellement coûteux. |
| | Programmation dynamique par itération de valeur (Bertsekas 2012) | Met à jour les valeurs d'état jusqu'à convergence. | Efficace pour trouver des valeurs optimales. | Nécessite un modèle précis de l'environnement. |

| | Méthode | Description | Avantages | Inconvénients |
|---|---|---|---|---|
| Sans modèle | Différence temporelle (TD) (Sutton 1988) | Apprend des valeurs d'état en se basant sur les différences temporelles. | Apprentissage en ligne, adaptable. | Peut être instable sans réglage approprié. |
| | Prédiction Monte Carlo (Sutton and Barto 2018) | Utilise des échantillons pour estimer les valeurs. | Simple et intuitive. | Nécessite des épisodes complets pour la mise à jour. |
| | Contrôle Monte Carlo (Sutton and Barto 2018) | Apprend des valeurs d'action en utilisant des échantillons complets. | Efficace pour évaluer des politiques. | Peut être inefficace pour des environnements continus. |
| | SARSA (State-Action-Reward-State-Action) (Sutton and Barto 2018) | Apprend des valeurs d'action en suivant une politique. | Converge vers une politique sûre. | Plus lent que *Q-learning* dans certains cas. |
| | Q-Learning (Watkins and Dayan 1992) | Apprend des valeurs d'action indépendamment de la politique. | Converge vers la politique optimale. | Peut surévaluer certaines actions sans réglage approprié. |
| | Réseaux Q profonds (DQN)(Mnih, Kavukcuoglu et al. 2015) | Utilise des réseaux de neurones pour approximer les valeurs Q. | Gère des environnements complexes et continus. | Nécessite des ressources computationnelles importantes. |
| | Politique de gradient (Sutton, McAllester et al. 1999) | Optimise directement la politique par gradient. | Efficace pour des environnements complexes. | Peut être instable et sensible aux paramètres. |
| | Acteur-critique (Silver, Lever et al. 2014) | Combine la politique de gradient et les valeurs d'état. | Équilibre exploration et exploitation. | Complexe à implémenter. |

La programmation dynamique par évaluation de politique évalue la valeur de chaque état sous une politique fixe. Cette méthode est simple à implémenter pour de petits espaces d'état mais inefficace pour de grands espaces d'état. La programmation dynamique par itération politique alterne entre évaluation de politique et amélioration de politique. Elle converge vers la politique optimale, mais peut être computationnellement coûteuse. La programmation dynamique par itération de valeur met à jour les valeurs d'état jusqu'à convergence, ce qui est efficace pour trouver des valeurs optimales, mais nécessite un modèle précis de l'environnement.

La méthode de différence temporelle (TD) apprend des valeurs d'état en se basant sur les différences temporelles, offrant un apprentissage en ligne et adaptable. Cependant, elle peut être instable sans réglage approprié. La simulation de Monte Carlo a deux applications différentes en fonction de s'il s'agit d'un problème de prédiction ou de contrôle. La prédiction Monte Carlo utilise des échantillons pour estimer les valeurs d'états. Elle est simple et intuitive, mais nécessite des épisodes complets pour la mise à jour. Le contrôle Monte Carlo apprend des valeurs d'action en utilisant des échantillons complets, ce qui est efficace pour évaluer des politiques, mais peut être inefficace pour des environnements continus.

SARSA (*State-Action-Reward-State-Action*) apprend des valeurs d'action en suivant une politique, convergeant vers une politique sûre, mais est plus lent que *Q-learning* dans certains cas. *Q-Learning* apprend des valeurs d'action indépendamment de la politique, convergeant vers la politique optimale, mais peut surévaluer certaines actions sans réglage approprié. La méthode acteur-critique combine la politique de gradient et les valeurs d'états, équilibrant exploration et exploitation.

Les réseaux Q profonds (DQN) utilisent des réseaux de neurones pour approximer les valeurs Q, gérant des environnements complexes et continus, mais nécessitent des ressources computationnelles importantes. La valeur Q est une estimation de la récompense attendue à long terme pour chaque paire d'état-action. La politique de gradient optimise directement la politique par gradient, efficace pour des environnements complexes, mais peut être instable et sensible aux paramètres. Il peut utiliser également la méthode TD pour mettre à jour les

valeurs Q, permettant un apprentissage en ligne efficace et adaptable. Il utilise des techniques d'expérience rejouée et des réseaux cibles pour stabiliser l'apprentissage. L'expérience rejouée consiste à stocker les transitions d'état passées et à les échantillonner aléatoirement pour l'apprentissage, ce qui réduit la corrélation entre les transitions d'apprentissage successives. Les réseaux cibles, copies du réseau principal avec des poids fixes pendant plusieurs itérations, rendent l'entrainement plus stable en réduisant la variance des cibles d'apprentissage.

En résumé, l'apprentissage par renforcement, et en particulier le modèle DQN que nous utilisons dans notre proposition, est bien adapté aux systèmes continus et sans modèle, permettant de gérer la complexité et la variabilité de l'environnement de manière efficace et robuste.

## 1.4 SYNTHÈSE DU CHAPITRE

Pour résumer, dans un premier temps, différents éléments de traitement du signal et notamment de filtrage de données numériques ont été développés. Différentes techniques ont été discutées, mais il a été constaté qu'en fonction du type de bruit, l'éliminer entièrement des données prises en conditions réelles est impossible, car les causes du bruit sont multiples et aléatoires. Suivant l'application, certaines variations du signal qui pourraient être considérées comme du bruit, peuvent contenir des informations utiles. Dans un second temps, le domaine de l'IA a été exploré, offrant des techniques pouvant être utilisées pour prédire des données lissées. Diverses notions et modèles tels que les CNN et leurs extensions, ont été introduits. En effet, ces modèles sont adaptés et très utilisés pour traiter des données numériques temporelles. Ensuite, la notion d'optimisation de systèmes a été présentée, explorant différentes techniques utilisées dans ce domaine. Enfin, l'apprentissage par renforcement a été introduit, car il propose des pistes pour améliorer les prises de décisions d'un système grâce à l'IA. Les travaux et applications détaillés dans ce premier chapitre seront approfondis dans le chapitre suivant.

# CHAPITRE 2
# REVUE DE LITTÉRATURE

Dans les premières parties de ce mémoire, nous avons pu constater que le recours au pesage dynamique est courant pour vérifier la qualité des produits ou s'assurer qu'ils répondent à certains critères. Cette méthode permet de contrôler le poids des objets en mouvement, ce qui économise du temps et des ressources. Différentes études ont été menées dans des domaines d'applications variés, mais avec des critères similaires visant à maintenir les avantages du pesage dynamique tout en évitant d'interrompre la ligne de production et en obtenant des résultats fiables. Il est toutefois important de noter que la plupart des recherches menées se concentrent sur l'étude du pesage dynamique sur des convoyeurs. Malgré le gain de temps et de productivité qu'offre le pesage en mouvement, celui-ci peut avoir un impact sur la précision des valeurs mesurées principalement en raison des bruits captés par les cellules de charge. Cependant, ces bruits peuvent être atténués à l'aide de filtres, comme cela sera démontré dans la suite de ce chapitre. Dans ce chapitre, nous présenterons différents travaux appliqués à des tâches diverses du domaine industriel. Étant donné que le sujet de ce mémoire concerne un procédé spécifique, nous avons exploré des tâches reposant sur des données de forme similaire afin d'élargir nos recherches. Nous présentons donc les modèles les plus couramment utilisés pour le traitement et la prédiction de données de séries temporelles, qu'ils soient basés sur l'IA ou non. De plus, nous détaillons également des applications d'optimisation d'API.

## 2.1 L'INDUSTRIE 4.0 ET L'INTELLIGENCE ARTIFICIELLE

La troisième révolution industrielle est caractérisée par l'intégration de l'informatique et l'amélioration de l'automatisation dans l'industrie. Les progrès récents réalisés dans les domaines de l'informatique et du numérique ont été continuellement intégrés dans l'industrie, conduisant à la quatrième révolution industrielle, souvent désignée sous le terme d'« industrie 4.0 ». Ce concept a été employé pour la première fois en Allemagne lors d'une conférence au forum mondial de l'industrie à Hanovre en 2011 (Kagermann, Lukas et al. 2011). La différence entre ces deux révolutions réside dans le fait que la troisième révolution consistait à automatiser l'industrie et sa production, tandis que la quatrième révolution se caractérise par une industrie connectée et flexible. Les progrès qui ont amené à cette révolution comprennent notamment l'internet des objets, l'IA et l'analyse des données collectées à différentes étapes de la chaine de fabrication. Cette révolution a entrainé plusieurs améliorations dans l'industrie, telles que l'automatisation des tâches de plus en plus complexes (Kim, Kong et al. 2022), l'évolution de la maintenance préventive vers la maintenance prédictive et l'optimisation de différentes tâches grâce à l'analyse des données de la chaine de production, ce qui a permis d'accroître sa flexibilité.

Grâce à l'intégration de l'IA à chaque étape de la chaîne de production, les entreprises peuvent faire la prévention et ainsi éviter des usures prématurées et allonger la durée de vie utile. La collecte des données tout au long du processus de fabrication revêt une grande importance, car ces données peuvent être utilisées pour anticiper ou détecter ces défauts (Xu, Sun et al. 2017). La mise en place de la maintenance prédictive est alors possible, particulièrement pour les machines tournantes dont les roulements constituent une pièce importante. En identifiant la présence d'un défaut et sa nature, il est possible de localiser le problème et d'intervenir avant qu'une panne ne survienne (Schwendemann, Amjad et al. 2021). L'IA a notamment intégré le domaine de la fabrication automobile. BMW, par exemple, exploite diverses technologies de l'industrie 4.0 (Seidel 2020) pour simplifier les tâches répétitives de ses employés et garantir la conformité des véhicules produits. Parmi ces technologies, BMW utilise la reconnaissance d'image pour détecter les pièces mal

assemblées ou manquantes. De même, dans d'autres applications automobiles, Amazon utilise la technologie UVeye pour vérifier le bon fonctionnement de ses véhicules de livraisons (Garsten), tandis que le fabricant de pneus Bridgestone utilise l'IA pour garantir le bon assemblage de ses pneus . Ce système est capable d'inspecter 480 éléments de qualité en temps réel, ce qui a permis d'améliorer la conformité de 15% par rapport à un processus traditionnel de fabrication.

Dans la détection de défauts, les réseaux de neurones artificiels sont capables de prédire la direction et d'autres caractéristiques des fissures dans des alliages d'aluminium, par exemple (Gope, Gope et al. 2015). De nombreux procédés, tels que le soudage laser utilisé dans l'industrie automobile, médicale et autres, bénéficient des améliorations apportées par l'IA pour garantir leur qualité (Cai, Wang et al. 2020).

Dans l'assemblage électronique, l'inspection des cartes de circuits imprimés de plus en plus complexes, nécessite d'être automatisée. L'utilisation de capteurs haute résolution obtient un résultat limité. Dans une étude, un auto-encodeur a été entrainé pour détecter des défauts d'assemblage, ouvrant ainsi la voie à des perspectives intéressantes pour améliorer la qualité dans la fabrication de produits électroniques (Kim, Ko et al. 2021). Mitsubishi tire parti de l'IA, et notamment l'apprentissage par renforcement, pour enseigner de nouvelles tâches à ses bras robotiques. Les méthodes traditionnelles sont difficiles à mettre en œuvre avec succès (Lindner, Milecki et al. 2021). En effet, cela demande soit de contrôler manuellement les mouvements, soit de prédéfinir des points de trajectoires en fonction de l'application. L'apprentissage par renforcement est également employé dans des processus interconnectés qui nécessitent de prendre des décisions en tenant compte de plusieurs critères (He, Tran et al. 2021). Il est ainsi combiné avec d'autres modèles intelligents pour optimiser des processus, par exemple dans la fabrication textile.

La fabrication additive, ou impression 3D, constitue une avancée importante dans le prototypage rapide et est devenue une composante de l'industrie 4.0 (Mehrpouya, Dehghanghadikolaei et al. 2019). L'IA est appliquée dans ce domaine pour intervenir lorsqu'un défaut est détecté. Des réseaux de neurones sont entrainés pour détecter les défauts

lors de l'impression, permettant des ajustements en temps réel pour corriger les paramètres associés aux défauts détectés (Paraskevoudis, Karayannis et al. 2020). Cette méthode de fabrication combinée à l'IA renforce la rapidité et la facilité de mise en œuvre de ces procédés (Wang, Zheng et al. 2020). L'IA est également utilisée dans des procédés d'usinage par enlèvement de matière pour améliorer la qualité des pièces usinées par des fraiseuses à commande numérique (Pimenov, Bustillo et al. 2018). Elle prédit, par exemple, la rugosité de la surface obtenue par surfaçage en se basant sur l'usure des dents de la fraise à surfacer, évitant ainsi la production des pièces non conformes et rendant l'usinage plus performant. Cette inspection en temps réel des outils de coupe est aussi applicable aux outils de rainurages (Shankar, Mohanraj et al. 2019). Le comportement des matériaux peut également être modélisé et prédit, par exemple dans des procédés de formages à froid tels que le pliage ou l'emboutissage. La rupture ductile qui est une déformation importante menant à la rupture peut ainsi être évitée (Di Lorenzo, Ingarao et al. 2006).

Dans des applications se basant sur un modèle physique pour analyser l'impact des défauts sur le produit, l'IA est utilisée pour simuler des données afin de détecter des dommages simples ou multiples, comme illustré dans une étude sur des rotors d'hélicoptères (Ganguli, Chopra et al. 1998). Le modèle physique combiné à un réseau de neurones permet de détecter les dommages potentiels sur le rotor et de renforcer la sécurité des produits.

L'IA est également une composante essentielle de l'industrie 5.0, où l'humain est placé au centre de la production et des notions de développement durable, d'économie circulaire et de diminution de la production de masse sont également considérées. Dans le cadre de cette nouvelle révolution industrielle, elle est utilisée pour favoriser l'innovation (Akundi, Euresti et al. 2022).

## 2.2 FILTRAGE DES BRUITS

Dans cette section, des études et des cas d'applications intégrants le filtrage de signaux et la prédiction de données temporelles sont présentés. Plusieurs méthodes sont développées et appliquées à des systèmes de pesage, mais également à d'autres systèmes qui comprennent des signaux bruités issus de capteurs.

Dans l'industrie alimentaire, où le poids des aliments est une contrainte majeure, le pesage dynamique est largement utilisé tout au long de la chaine de production. Le pesage d'œufs est un des exemples d'applications dans ce domaine (Yabanova 2017). Le poids est mesuré par une cellule de charge disposée sous la ligne de production, puis un filtrage numérique est appliqué pour traiter le signal en éliminant les bruits et les interférences afin d'obtenir des résultats plus précis.

Ainsi, le pesage dynamique permet d'accélérer les étapes de vérification dans la ligne de production. Cependant, l'amélioration de la vitesse se fait au détriment de la précision, notamment impactée par les vibrations générées par les éléments en mouvement à proximité des cellules de charge. Yamazaki, Sakurai et al. proposent un algorithme de traitement du signal appliqué à des convoyeurs-peseurs pour améliorer la précision (Yamazaki, Sakurai et al. 2002). Les signaux de pesage sont également perturbés par des bruits causés par les éléments roulants du convoyeur ou les objets en mouvement sur celui-ci. Pour remédier à cela, un filtre numérique à réponse impulsionnelle finie (FIR) est conçu. Différentes périodes de sous-échantillonnages ont été testées, puis un filtre passe-bas a été mis en place. Cette méthode a été appliquée à des données expérimentales pour estimer les masses des objets pesés. Pour déterminer la masse, la valeur maximale atteinte dans le signal lissé a été prise en compte. En analysant la dispersion des erreurs d'estimations, deux groupes distincts ont été identifiés. Un algorithme des moindres carrés a alors été appliqué pour définir une formule de correction. Les résultats montrent que les erreurs d'estimation sont inférieures à la précision de 0,7% qui a été demandée.

Choi se concentre sur l'analyse et le filtrage du bruit causé par les mouvements du convoyeur qui génèrent des vibrations captées par les cellules de charge (Choi 2017). Une analyse du bruit est réalisée dans les domaines temporel et fréquentiel. Le spectre de fréquences révèle des pics de résonance. Ces pics augmentent linéairement à faible vitesse puis de façon non linéaire lorsque la vitesse augmente. Pour réduire ce bruit, une méthode est proposée. Pour commencer, un filtre passe-bas est appliqué pour éliminer les fréquences liées au bruit. La fréquence de coupure est ajustée pour conserver uniquement le signal utile. Ensuite, la bande passante est ajustée dynamiquement. Cette méthode de filtrage a permis d'obtenir une meilleure précision par rapport à une méthode où la fréquence de coupure du filtre est fixe. Un filtre similaire est également appliqué par Tasaki, Yamazaki et al. pour du pesage continu d'objets sur un convoyeur à bandes multiples (Tasaki, Yamazaki et al. 2007). Les différentes bandes du convoyeur comprennent des cellules de charge afin de peser des objets de grandes tailles sans perdre de temps. Ce filtre est choisi car il revient rapidement à un état initial après le passage d'un objet. Une fois le signal filtré, la masse est estimée en prenant la valeur maximale présente dans ce signal.

Cependant, la non-uniformité du poids distribué sur le convoyeur peut entrainer plusieurs pics dans le signal. Cela se produit lorsque le poids de l'objet n'est pas uniformément réparti sur les bandes du convoyeur, ce qui peut se produire lorsque l'objet a une forme particulière ou qu'il est mal positionné. Ainsi, les différents pics que cela cause dans le signal ont pour conséquence de rendre difficile la détermination de la masse. Pour résoudre ce problème, une plage de valeurs acceptables autour de la valeur maximale est définie en fonction du temps nécessaire pour atteindre cette valeur maximale. Il s'agit d'un intervalle de 10 millisecondes autour de la valeur maximale afin d'éviter les erreurs en choisissant un pic qui ne représente pas correctement la masse de l'objet pesé. Les expériences réalisées ont les paramètres variables suivants : la longueur des objets (20 à 130 centimètres), leur masse (de 20 à 80 kilogrammes) et la distance entre deux objets sur le convoyeur (40 et 50 cm). Les résultats ont montré que cette méthode permet d'atteindre une précision satisfaisante dans l'estimation de la masse, bien que les petits produits d'une longueur de 20 cm aient tendance à avoir des erreurs d'estimation plus élevées. Ces erreurs

sont principalement causées par les comportements transitoires et les conditions dynamiques lors du chargement et des mouvements des objets sur le convoyeur. Les comportements transitoires sont causés par le dépôt rapide des objets sur le convoyeur, ce qui a un fort impact sur les petits objets. Cela amène à une hausse du rapport signal/bruit car les petits produits traversent plus rapidement la zone de pesage, ce qui réduit la fenêtre de temps pour le pesage. La distinction entre le poids réel de l'objet et les perturbations du signal dues aux comportements transitoires est donc rendue plus difficile, même pour le filtrage. Concernant les conditions dynamiques du chargement et les autres causes de vibrations, elles ont un impact proportionnellement plus important sur les petits objets. En effet, dans ce cas également, la durée pendant laquelle l'objet est en contact stable avec la zone de pesage est plus faible. Le signal a donc moins de temps pour se stabiliser. Cependant, Yabanova mentionne que la littérature rapporte qu'il est difficile de filtrer le bruit de pesage en utilisant uniquement un filtre FIR passe-bas (Yabanova 2016). Des filtres basés sur des fonctions de fenêtre ont donc été testés. Le filtre de moyenne mobile ainsi que la fenêtre de Barthlett-Hanning ont été mis en place. L'analyse des résultats obtenus a permis de conclure que les filtres avec fonction de fenêtre peuvent être utilisés avec succès pour filtrer le signal de mesure.

Dans une étude de Boschetti, Caracciolo et al., un modèle est également décrit pour compenser les vibrations environnementales captées par les cellules de pesage (Boschetti, Caracciolo et al. 2013). Selon ce travail, la méthode la plus couramment utilisée consiste à filtrer les composantes de Fourier à haute fréquence, mais cela nécessite de déterminer les paramètres de manière à ne pas ralentir le système, ce qui peut être difficile. Pour résoudre cela, un modèle est proposé pour compenser en temps réel l'effet des vibrations sur les cellules de charges. Il s'appuie sur les mesures fournies par un réseau d'accéléromètres supplémentaires et sur le modèle multicorps du système, incluant la structure et ses capteurs. Les mesures des accélérations en trois points non alignés au niveau de la structure permettent de compenser les vibrations environnementales correspondant aux basses fréquences. Ensuite, un filtre passe-bas est utilisé pour éliminer les vibrations à haute fréquence, y compris celles à la fréquence propre des capteurs. Cette méthode permet de filtrer

efficacement les vibrations environnementales à basse fréquence sans introduire de phases importantes, comme cela peut se produire avec l'utilisation des filtres à bandes passantes étroites. Les tests ont été effectués dans des condition où les cellules de charge présentent un comportement idéal, c'est-à-dire sans prendre en compte les vibrations transversales. Les résultats obtenus ont montré que cette méthode constitue une solution efficace. Cependant, son efficacité reste à déterminer dans des situations réelles, où les vibrations transversales doivent être prises en compte.

Une des techniques utilisées pour atténuer les bruits causés par les vibrations mécaniques captées par les cellules de charge des convoyeurs de pesage est l'utilisation de filtres passe-bas. Les filtres passe-bas qui sont généralement mis en place, ont une fréquence de coupure qui reste fixe tout au long du filtrage. Ainsi, Pietrzak, Meller et al. étudient l'application d'un filtre passe-bas avec une fréquence de coupure variable dans le temps et adaptée aux différentes étapes du signal (Pietrzak, Meller et al. 2014). Lorsque l'objet arrive sur la partie du convoyeur dédiée au pesage, une basse fréquence de coupure est utilisée pour atténuer les oscillations rapides du signal. Ensuite, lorsque l'objet atteint la partie balance, une fréquence de coupure élevée est appliquée, puis elle est réduite progressivement afin de minimiser les perturbations et stabiliser la mesure. Cet ajustement de la bande-passante vise à améliorer la précision des mesures, même si elle entraine un temps de réponse plus long. L'objectif de cette variation de la bande-passante est d'optimiser la réponse du système de pesage aux variations rapides de charge, sans nécessiter l'arrêt des objets sur le convoyeur. Les résultats de cette approche sont comparés à ceux d'une approche basée sur l'identification, provenant d'autres recherches (Niedźwiecki and Wasilewski 1996). Cette dernière fonctionne de façon adaptative sur un modèle masse-ressort-amortisseur du second degré, mais les résultats obtenus pour les mêmes données d'évaluation n'étaient pas concluants.

L'approche proposée par Niedźwiecki et Pietrzak est un modèle de réponse impulsionnelle finie (FIR) pour filtrer le signal du convoyeur de pesage (Niedźwiecki and Pietrzak 2016). Il combine deux approches existantes : une méthode d'identification par

sous-espace (Markovsky 2015) et l'approche de filtrage variable dans le temps décrite précédemment (Pietrzak, Meller et al. 2014). Quatre variantes de cette approche sont proposées. Les résultats montrent que cette solution de filtrage, basé sur un modèle FIR de la réponse du système, offre une précision de pesage jusqu'à quatre fois supérieure à celle des autres solutions existantes (Meller, Niedźwiecki et al. 2014, Pietrzak, Meller et al. 2014, Markovsky 2015). La méthode d'identification par sous-espace (Markovsky 2015) utilisée dans cette approche ne nécessite pas d'estimer explicitement les paramètres du modèle. Le système de mesure est modélisé comme un système dynamique où la masse de l'objet est la variable inconnue et la déviation verticale de la plateforme de pesage est celle mesurée. Une équation différentielle permet alors de décrire le processus de mesure où la modélisation prend en compte différents facteurs tels que la gravité, l'élasticité et l'amortissement. L'étude de Meller, Niedźwiecki et al. (Meller, Niedźwiecki et al. 2014) dont les résultats servent à comparer l'efficacité du modèle développé par Niedźwiecki et Pietrzak (Niedźwiecki and Pietrzak 2016), compare une approche d'identification du système et une approche de filtrage à largeur de bande variable. Dans l'approche d'identification du système, le signal est mesuré comme étant la réponse à une impulsion d'un système dynamique du second ordre dont les paramètres sont inconnus. À chaque nouveau cycle de pesage, les paramètres sont estimés et la réponse du modèle est calculée. Dans l'approche basée sur le filtrage à bande passante variable, un filtre passe-bas linéaire est utilisé pour atténuer rapidement les perturbations dans la bande passante sélectionnée.

L'étude réalisée par Martin et Molteno vise à mesurer la masse d'un sac de lait en poudre en cours de remplissage (Martin and Molteno 2015). Le sac est suspendu à un pivot qui subit des mouvements oscillatoires causés par la gravité et le remplissage progressif, ce qui peut affecter la précision des mesures en temps réel. Pour résoudre ce problème, des algorithmes d'inférence séquentielle, tels que le filtre de Kalman non linéaire *(Unscented Kalman Filter*) (Wan and Van Der Merwe 2000) et le filtre particulaire (Del Moral 1997), sont testés et comparés à la modélisation physique du système. Dans la modélisation physique, le système est représenté comme étant un pendule avec une masse variable. Les mouvements et les changements physiques du sac lors du remplissage sont pris en compte

pour améliorer la précision des prédictions. Des tests ont été réalisés en utilisant uniquement les algorithmes d'inférence séquentielle, ainsi qu'en les combinant avec le modèle physique. Les résultats ont montré que la combinaison des algorithmes avec le modèle physique améliorait la performance des méthodes d'inférence séquentielle. Pour le filtre de Kalman non linéaire, deux versions ont été testées : une avec état augmenté et une sans. Les deux versions associées au modèle physique ont fourni des prédictions rapides et précises avec de meilleurs résultats pour la version avec état augmenté. Quant au filtre particulaire, il s'agit d'une méthode non paramétrique qui utilise un ensemble de particules pour représenter la distribution de probabilité des états estimés. Il s'est avéré moins performant que les deux versions du filtre de Kalman, en particulier lorsqu'il n'était pas combiné au modèle physique.

Le domaine du pesage industriel comprend également le pesage des moyens de transports dédiés à la logistique des marchandises tels que des camions ou des trains. Liang et Qingli se concentrent sur le pesage rapide des camions, où le conducteur s'arrête sur la plateforme mais garde le moteur allumé pendant le processus de pesage pour gagner du temps (Liang and Qingli 2021). Cependant, cela peut affecter la précision des mesures. Pour garantir une certaine précision, les auteurs ont opté pour des améliorations des éléments matériels en plus de l'utilisation des méthodes de filtrage. Parmi ces éléments, il y a un module de conversion analogique-numérique offrant une haute résolution et un régulateur de tension linéaire afin de réduire le bruit électrique dans le système. En ce qui concerne la méthode de filtrage, elle combine un filtre médian avec un filtre de moyenne mobile. Le filtre médian remplace chaque valeur du signal par la médiane des valeurs voisines, éliminant ainsi les valeurs aberrantes et les impulsions brusques. Le filtre de moyenne mobile lisse le signal sur une fenêtre de valeurs pour réduire le bruit de fond, telles que les interférences périodiques causées par le moteur du camion lors du pesage. Cette combinaison de filtres permet d'obtenir des valeurs précises et stables, même en présence d'interférences périodiques ou d'impulsions dans le signal, assurant ainsi une mesure précise lors du pesage des camions.

Pour réduire le temps de pesage, une approche consiste à prédire la masse d'un objet en mouvement avant qu'il ne soit stabilisé sur le convoyeur. Pour cela, un perceptron multicouche, un type de réseaux de neurones, peut être utilisé (Bahar and Horrocks 1998). Ce réseau est entrainé sur différentes séquences de pesage enregistrés, qui comprennent des formes d'ondes correspondant à des objets de différentes masses. Les résultats ont montré que le perceptron multicouche était capable de prédire de manière précise les masses appliquées sur le convoyeur, malgré des vibrations variées. Dans une étude réalisée par Ma, Li et al., un réseau de neurones est appliqué dans un système innovant qui permet de peser des poulets de manière précise tout en réduisant le stress de l'animal (Ma, Li et al. 2021). Pour atteindre cet objectif, la méthode commence par utiliser un filtre combinant un filtrage par médiane et moyenne mobile. Ensuite, le réseau de neurones est utilisé pour améliorer la précision du pesage. Les tests réalisés en conditions réelles ont montré que cette méthode a permis de réduire l'erreur de pesage de 6% à 3%.

He, Li et al. réalisent une étude qui concerne la pesée dynamique de vaches laitières, traditionnellement réalisée de façon statique (He, Li et al. 2023). Cela a pour conséquence d'être une activité très chronophage mais également stressante pour les animaux. Un algorithme utilisant la transformée par ondelettes empiriques est d'abord employée pour obtenir une première prédiction du poids. Puis, un modèle SVR est appliqué pour corriger les erreurs. Il est utilisé pour prédire les erreurs entre le poids réel et le poids prédit par la transformée. Pour cela, il se base sur les caractéristiques du signal en entrées et les erreurs de prédictions initiales en sortie. L'erreur absolue moyenne de la masse de la vache obtenue en utilisant uniquement la transformée est de 6,33 kg, tandis qu'en la combinant avec le modèle SVR, celle-ci est de 3,81 kg. Cela démontre l'intérêt de combiner ces deux modèles pour améliorer la précision des mesures et permettre de réaliser des pesées dynamiques des animaux.

D'autres domaines sont confrontés à des problématiques similaires lorsqu'il s'agit de traiter des signaux numériques provenant de capteurs. Les exemples détaillés ensuite présentent des défis similaires à ceux rencontrés dans le domaine du pesage.

Dans le domaine médical, les signaux enregistrés sont souvent des signaux de faible amplitude et contiennent beaucoup de bruits qui sont causés par divers facteurs tel que le mouvement des électrodes, les interférences électriques ou les mouvements dans le corps humain. Pour atténuer ces bruits, différentes techniques sont utilisées, notamment les filtres à réponse impulsionnelle finie ou les filtres de Butterworth (AlHinai 2020). Cependant ces méthodes traditionnelles ne garantissent pas des résultats satisfaisants car les bruits enregistrés sont généralement non stationnaires et peuvent contenir des fréquences similaires au signal souhaité. Cette superposition de fréquences est nommée chevauchement spectral et rend difficile la distinction entre le bruit et le signal attendu. Ainsi, des méthodes d'IA ont été employées pour résoudre ce problème (Poungponsri and Yu 2013). Nous retrouvons les autoencodeurs qui sont des modèles qui permettent de reconstruire les données sans bruit à partir des données brutes. Chiang, Hsieh et al. ont comparé trois modèles : un CNN, un réseau de neurone profond et un autoencodeur composés de couches de convolution (Chiang, Hsieh et al. 2019). Les résultats ont montré une meilleure performance de l'autoencodeur. Il a permis de préserver plus de détails tout en supprimant efficacement le bruit du signal.

Toujours pour améliorer la qualité des signaux ECG (électrocardiogramme), Arsene, Hankins et al. ont comparé l'efficacité d'un modèle CNN et d'un modèle LSTM avec une technique basée sur les ondelettes (Arsene, Hankins et al. 2019). Le modèle CNN utilisé dans l'étude est composé de six couches convolutionnelles, suivie chacune d'une couche de normalisation par lots, d'une couche d'unité linéaire rectifiée (ReLU) et d'une couche de *pooling* moyen. Le réseau se termine par une couche entièrement connectée afin de réaliser la régression. Ce modèle est conçu pour apprendre comment filtrer le bruit afin de récupérer les signaux ECG sans bruits. Le modèle LSTM est composé de deux couches LSTM suivies de couches ReLu, permettant d'apprendre les dépendances à long terme dans les données pour filtrer les signaux ECG. Concernant la technique basée sur les ondelettes, elle décompose le signal en plusieurs couches en utilisant la transformée par ondelettes. Le signal est ainsi décomposé suivant différentes bandes de fréquences. Les couches supérieures correspondent aux hautes fréquences et captent donc les détails tel que le bruit. Les couches inférieures se focalisent sur les basses fréquences et permettent de capter la tendance globale

du signal. En identifiant les coefficients d'ondelettes associés au bruit ou au signal ECG, cette méthode permet de reconstruire le signal filtré à partir de ces coefficients. Ces modèles ont été évalués sur deux ensembles de données synthétiques puis sur un ensemble de données réelles. Les résultats obtenus ont montré que les modèles CNN et LSTM étaient capables de gérer efficacement un bruit important et variable dans le temps, avec une performance supérieure pour le modèle CNN. En revanche, le modèle utilisant la transformée par ondelettes a été efficace pour supprimer le bruit aléatoire mais moins adapté aux bruits complexes et très variables.

Jhang, Wang et al. cherchent également à filtrer les signaux ECG à l'aide de l'IA (Jhang, Wang et al. 2022). Un auto-encodeur de filtrage avec *pooling* par canal (*Chanel-wise pooling denoising autoencoder*, CPDAE) a été développé. Cette technique est utilisée pour filtrer les bruits causés par la respiration, les mouvements du patient et les interférences électriques présents lors de l'enregistrement des signaux ECG. L'auto-encodeur de filtrage cherche à apprendre une représentation des données avec moins de bruit. Pour cela, l'encodeur fait passer le signal bruité à travers plusieurs couches du réseau et le comprime en un ensemble plus petit composé des caractéristiques essentielles. Chaque couche extrait et conserve les informations importantes tout en supprimant le bruit progressivement. Quant au décodeur, il reconstruit le signal en ne conservant que les caractéristiques essentielles. Une technique appelée « *channel-wise average pooling* » (CWAP) est utilisée pour réduire la dimension des données et conserver les informations importantes. Lorsque le signal passe dans l'encodeur, il est transformé en un ensemble de caractéristiques à plusieurs niveaux aussi appelés canaux. Avant de transférer les caractéristiques de l'encodeur vers le décodeur, une moyenne de chaque canal est transmise au décodeur pour réduire la quantité de données à traiter, ce qui permet de diminuer la puissance de calcul nécessaire. Pour améliorer la résolution du signal décodé, une technique de mélange de pixels, généralement utilisée pour l'encodage-décodage d'images, est adaptée pour fonctionner avec des données unidirectionnelles. Les caractéristiques obtenues lors de l'encodage sont réorganisées avant de passer à travers le décodeur pour reconstruire un signal de qualité malgré la réduction de dimension appliquée lors de l'encodage. Trois versions de ce modèle ont été développées

dans l'étude : CPDAELite, CPDAEFull et CPDAERegular. CPDAELite comprend moins de couches dans l'encodage et le décodage ainsi que moins de paramètres pour être utilisé en temps réel ou sur des petits appareils. CPDAEFull offre de meilleures performances de filtrage des ECG mais nécessite une plus grande capacité de calculs. CPDAERegular se situe entre les deux autres versions en termes de capacités de filtrage et de complexité. Les résultats ont montré que ces modèles permettent de réduire efficacement le bruit dans les signaux ECG sans nécessiter une grande puissance de calcul, grâce aux techniques CWAP et de mélange de pixels.

Keshavarzi compare un modèle de régression à vecteurs de support (SVR) avec un modèle de réseaux de neurones récurrents (RNN) pour la prédiction des signaux ECG (Keshavarzi 2022). Les deux modèles ont été testés sur des signaux ayant différents niveaux de bruits et entrainés avec des jeux de données de tailles différentes. Leurs performances de filtrage ont été évaluées à l'aide de deux métriques : l'erreur quadratique moyenne et la corrélation de Pearson. Les résultats ont montré que le modèle RNN a permis d'obtenir de meilleurs résultats dans tous les cas testés et donc de prédire des signaux ECG plus nets.

Dans le domaine sismique, l'intérêt du filtrage consiste à éliminer les bruits aléatoires présents dans les données afin de faciliter leur interprétation (Jinhuan, Weay et al.). Les méthodes traditionnelles nécessitent souvent la modélisation du signal à l'aide de paramètres définis au préalable. Cela limite l'application, car le signal sismique est très variable en fonction de la localisation et les paramètres sont difficiles à estimer pour des régions reculées. Pour surmonter ces défis, Jinhuan, Weay et al. utilisent un réseau CNN pour éliminer le bruit aléatoire tout en préservant les caractéristiques essentielles du signal sismique. Pour cela, il apprend directement à partir des caractéristiques des données plutôt que de dépendre de paramètres prédéfinis. Cette approche offre ainsi l'avantage de ne pas nécessiter une modélisation préalable des paramètres du signal et la rend particulièrement utile dans des contextes où ces paramètres sont difficiles à estimer.

De plus, comme cela a été évoqué dans la partie sur l'industrie 4.0, l'utilisation de l'IA pour la maintenance prédictive est un domaine en pleine expansion, comme le montre l'étude

réalisée par Zarei, Tajeddini et al. sur la détection des défauts dans les roulements des moteurs à induction (Zarei, Tajeddini et al. 2014). Les défauts de roulements sont une source majeure de problèmes dans ces moteurs (Prieto, Roura et al. 2011). Dans cette étude, les défauts sont détectés et classés suivant plusieurs catégories. Tout d'abord, un filtre est utilisé pour éliminer les vibrations qui ne sont pas liées à des défauts de roulement, ce qui permet de détecter les défauts propres aux roulements. L'algorithme de filtrage utilisé, appelé ADALINE (Widrow and Hoff 1960), est un neurone linéaire adaptatif (en anglais : Adaptative Linear Neuron) conçu spécialement pour le filtrage adaptatif et le traitement du signal (Carpenter 1989). L'entrainement du filtre est réalisé sur des signaux de roulement qui n'ont pas de défauts, ce qui permet au réseau d'apprendre à identifier le bruit qu'il y a autour et de ne pas supprimer celui causé par un défaut. Ensuite, une fois que le filtre est appliqué à un signal qui présente un défaut, seuls les aspects caractéristiques du défaut restent dans le signal qui est transmis à l'algorithme de classification. L'avantage de cette technique est qu'elle ne nécessite pas une connaissance détaillée du système étudié. Seules les caractéristiques du domaine temporel sont utilisées pour détecter les défauts. Des tests pour classer ces défauts ont été réalisés avec et sans application du filtre. Les résultats ont démontré que le filtre améliore la classification, car il permet de faire ressortir les caractéristiques spécifiques des défauts apparaissant dans le signal.

Étant donné que les valeurs issues de capteurs sont des données temporelles, des méthodes statistiques telles que la méthode ARIMA et ses variantes ont été employées pour filtrer des données et réaliser des prédictions à partir de séries temporelles. Par exemple, une étude de Williams et Hoel vise à modéliser et prédire le flux de trafic routier en utilisant le modèle SARIMA, qui est une adaptation du modèle ARIMA pour les séries temporelles saisonnières (Williams and Hoel 2003). Pour rendre les données stationnaires, une opération de différenciation est appliquée, ce qui améliore la précision des prédictions grâce à une modélisation plus efficace. Les résultats obtenus avec ce modèle ont été comparés à ceux de méthodes heuristiques basiques, telles que la marche aléatoire et la moyenne. Le modèle SARIMA a permis d'obtenir des prédictions fiables à court terme.

Privitera, Bellissima et al. présentent une méthode basée sur le modèle ARIMA pour améliorer la précision des dosages effectués par des pompes péristaltiques (Privitera, Bellissima et al. 2023). Ces pompes sont couramment utilisées dans le domaine pharmaceutique pour leur capacité à limiter les risques de contamination. Néanmoins, elles montrent une variation de précision influencée par la température du produit dosé. Pour éviter ce problème, un système de dosage adaptatif utilisant le modèle ARIMA a été développé. Ce système analyse les données de dosage enregistrées et ajuste le volume dosé en fonction des prévisions obtenues par le modèle. Des tests effectués sur des lignes de production ont montré une amélioration de la précision des dosages pouvant atteindre 30%.

Bien que le modèle ARIMA soit efficace pour analyser des séries temporelles, il est limité avec des données trop complexes. Dans de tels cas, l'intégration de l'IA peut conduire à de meilleurs résultats. Siami-Namini, Tavakoli et al. illustrent cette approche dans une étude où une comparaison entre les modèles ARIMA et LSTM est effectuée sur des séries temporelles financières (Siami-Namini, Tavakoli et al. 2018). Les résultats montrent que le modèle LSTM, grâce à sa capacité à gérer des dépendances à long terme, surpasse le modèle ARIMA dans le traitement de ces données complexes.

Le filtre de Kalman est utilisé pour traiter des signaux bruités. Il commence par donner une estimation initiale de la valeur à filtrer et fait une prédiction pour la valeur suivante, basée sur les données historiques et leur tendance. Lorsqu'il reçoit une nouvelle valeur, elle est comparée à la prédiction qui a été faite pour celle-ci. Un affinage des paramètres est alors réalisé suivant l'écart entre la valeur prédite et celle observée (Welch and Bishop 1995). Manju et Sneha explorent l'élimination du bruit dans les signaux ECG (Manju and Sneha 2020) en comparant le filtre de Kalman avec le filtre de Wiener (Wiener 1949). Le filtre de Wiener utilise les propriétés spectrales du signal et du bruit pour optimiser le filtrage. Il s'est avéré plus efficace que le filtre de Kalman car les signaux ECG contenaient des non-linéarités qu'il a su prendre en considération. Postalcioglu, Erkan et al. évaluent le filtre de Kalman par rapport à un filtre à ondelettes pour des signaux non stationnaires (Postalcioglu, Erkan et al. 2005). Le filtre à ondelettes a prouvé une meilleure efficacité en décomposant le signal selon

différentes fréquences.  Toutefois, des variantes du filtre de Kalman, telles que le filtre de Kalman étendu présenté par Sayadi, Sameni et al., peuvent traiter efficacement les signaux non linéaires (Sayadi, Sameni et al. 2007). Le rapport signal-bruit a été significativement amélioré, ce qui signifie que moins de bruit est présent dans les signaux ECG. Louka, Galanis et al. utilisent également le filtre de Kalman pour rendre plus précises les prévisions de vitesse du vent et améliorer la gestion de la production d'énergie éolienne (Louka, Galanis et al. 2008). Le filtre estime les erreurs présentent dans les données et corrige les prédictions. La meilleure qualité des données météorologiques et de vitesses du vent ont permis d'obtenir des prédictions de la production d'énergie plus précises, tout en réduisant le temps de calcul nécessaire. Enfin, Kumar décrit l'utilisation du filtre de Kalman pour prédire le flux de circulation routière (Kumar 2017). Il propose donc une alternative intéressante aux modèles ARIMA et SARIMA et nécessite moins de données disponibles.

Song, Baek et al. présentent une étude qui combine une technique de filtrage et des modèles d'apprentissage profond pour prédire les indices boursiers (Song, Baek et al. 2021). L'objectif est d'améliorer la précision des prédictions en atténuant l'impact du bruit présent dans les données financières. La méthode de filtrage employée utilise la transformée de Fourier, pour laquelle un remplissage par des zéros (« *padding* ») est appliqué aux extrémités du signal. Ce *padding* vise à ajuster la longueur du signal à une puissance de deux, facilitant ainsi le calcul numérique par la transformation de Fourier rapide (*Fast Fourier Transform*). Il est important de noter que ce *padding* ne modifie pas le contenu informatif du signal original. Après cette étape, le signal est transformé du domaine temporel au domaine fréquentiel via la transformée de Fourier. Le domaine fréquentiel permet d'identifier les fréquences élevées qui sont généralement associées au bruit. Le bruit est ensuite éliminé et le signal est reconverti en domaine temporel. Les données ajoutées lors du remplissage sont retirées pour obtenir un signal sans bruits. Ce signal filtré est transmis à trois modèles d'apprentissage profond : RNN, LSTM et GRU. Ces modèles sont testés avec et sans données filtrées pour évaluer l'efficacité du filtrage. Les performances sont mesurées par l'erreur absolue moyenne (MAE), la racine de l'erreur quadratique moyenne (RMSE), l'erreur moyenne en pourcentage absolu (MAPE) et le ratio de succès. Les résultats indiquent une

amélioration significative des prédictions avec filtrage, particulièrement pour le modèle LSTM qui s'est avéré plus précis. Le filtrage a limité le surajustement aux données bruitées et a réduit le temps de prédiction, ce qui est non négligeable pour une application en temps réel.

Huang, Algahtani et al. comparent trois modèles d'apprentissage automatique pour prédire la consommation d'énergie d'une résidence universitaire (Huang, Algahtani et al. 2022). Les trois modèles testés sont le modèle LSTM, le modèle SVR et le modèle XGBoost. Ils ont été entrainés sur des données collectées toutes les trente minutes pendant un an. Elles ont été prétraitées pour enlever les valeurs aberrantes et les transformer avec les fonctions sinus et cosinus afin de mieux représenter la périodicité quotidienne et hebdomadaire. Le coefficient de variation a été utilisé pour comparer les résultats en évaluant la dispersion des données autour de leur moyenne. Le modèle SVR a obtenu les meilleurs résultats pour une prédiction de la consommation pour l'heure suivante à partir des données des 30 minutes précédentes. Le coefficient de variation était de 14,28%. Le modèle XGBoost a obtenu un coefficient de 15,27% pour prédire la consommation suivante avec quatre données précédentes. Le modèle LSTM a obtenu des valeurs de coefficients de variation plus élevées. Il était donc moins performant pour cette application.

Wei, Zhang et al. évaluent l'efficacité de sept algorithmes d'apprentissage automatique pour prédire la charge thermique d'un quartier de Shanghai (Wei, Zhang et al. 2021) . Les algorithmes testés sont : SVR, XGBoost, LSTM, CNN, perceptron multicouches, forêt aléatoire, K plus proches voisins. Ils ont été entrainés sur des données issues de capteurs électriques, thermiques, météorologiques et des prédictions météorologiques. Le modèle SVR a obtenu les meilleures performances d'après la métrique d'erreur moyenne absolue en pourcentage (MAPE), ce qui signifie qu'en moyenne il permet d'obtenir une précision de prédictions proches des valeurs réelles. Le modèle XGBoost a était le meilleur suivant la métrique d'erreur quadratique moyenne (RMSE), montrant une bonne efficacité pour minimiser les grandes erreurs de prédictions. La longueur des données historiques a été ajustée pour trouver la meilleure configuration. Néanmoins, le modèle LSTM a montré une

grande capacité pour capturer les caractéristiques des séries temporelles, mais les résultats obtenus étaient moins précis que ceux des modèles SVR et XGBoost.

Le Tableau 4 donne un récapitulatif des travaux discutés dans cette section, en détaillant les domaines abordés, les techniques utilisées, ainsi que les conclusions principales tirées de ces études.

Tableau 4. Aperçu de travaux du domaine du filtrage des bruits

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Yabanova 2017 | Industrie alimentaire pesage d'œufs sur convoyeur | Filtre numérique | Meilleure précision des mesures |
| Yamazaki, Sakurai et al. 2002 | Pesage continu sur convoyeur | Filtre numérique à réponse impulsionnelle finie (FIR) | Les erreurs d'estimation sont inférieures à 0,7% |
| Choi 2017 | Pesage continu sur convoyeur | Filtre passe-bas et ajustement du ratio d'impulsion du signal | Réduction des pics de bruit et amélioration de la précision |
| Tasaki, Yamazaki et al. 2007 | Pesage continu sur convoyeur | Filtre numérique à réponse impulsionnelle finie (FIR) | Efficace pour réduire les bruits de haute fréquence |
| Yabanova 2016 | Pesage continu sur convoyeur | Filtre de moyenne mobile et filtre de Barthlett-Hanning | Filtres à fenêtre efficaces pour filtrer le bruit |
| Boschetti, Caracciolo et al. 2013 | Pesage sur convoyeur | Compensation dynamique des vibrations | Après compensation, les erreurs résiduelles maximales sont réduites à moins de 7% de l'amplitude de la perturbation initiale. |
| Pietrzak, Meller et al. 2014 | Pesage continu sur convoyeur | Filtre passe-bas avec fréquence de coupure variable | Accélère la détermination du poids tout en conservant la précision |
| Niedźwiecki et Wasilewski 1996 | Pesage continu sur convoyeur | Approche d'identification | Résultats non concluants |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Niedźwiecki et Pietrzak 2016 | Pesage sur convoyeur | Filtre FIR qui combine une approche d'identification par sous-espace avec un filtrage variable dans le temps | Amélioration de la précision jusqu'à 4 fois supérieure par rapport aux solutions existantes |
| Markovsky 2015 | Pesage dynamique | Approche d'identification par sous-espace | Rapide et précis sans nécessiter une connaissance détaillée du système |
| Meller, Niedźwiecki et al. 2014 | Pesage dynamique | Approche d'identification et une approche de filtrage à largeur de bande variable | Ces deux approches sont peu efficaces par rapport à d'autres approches. |
| Martin et Molteno 2015 | Mesure d'une masse lors d'un remplissage | Filtre de Kalman non linéaire, filtre particulaire et modélisation physique (pendule avec masse variable) | Filtre particulaire moins performant que le filtre de Kalman. Filtre de Kalman combiné à la modélisation physique a obtenu des prédictions rapides et précises. |
| Liang et Qingli 2021 | Pesage des transports de marchandises | Combinaison d'un filtre médian et d'un filtre de moyenne-mobile | Valeurs précises et fiables même en présence d'interférences périodiques ou d'impulsions |
| Bahar and Horrocks 1998 | Pesage continu sur convoyeur | Perception multi-couches pour prédire la masse de l'objet | Il est capable de prédire les valeurs de façon précise |
| Ma, Li et al. 2021 | Pesage de poulets | Combinaison d'un filtre par médiane et d'un filtre de moyenne mobile | Réduction de l'erreur de pesage de 6% à 3%, |
| He, Li et al. 2023 | Pesage dynamique d'animaux | Transformée par ondelettes puis modèle SVR | L'erreur absolue moyenne sans SVR : 6,33 kg et avec SVR : 3,81 kg. |
| Alhinai 2020 | Signaux dans le médical | Filtre de Butterworth | Peu efficace car les bruits sont non stationnaires et peuvent avoir des fréquences proches du signal souhaité |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Poungponsri and Yu 2013 | Médical, signaux ECG | Modèle combinant la transformée par ondelettes avec un reseau de neurones artificiel | Réduction efficace du bruit malgré le chevauchement spectral |
| Chiang, Hsieh et al. 2019 | Médical signaux ECG | Comparaison entre un réseau CNN, un réseau de neurone profond et un autoencodeur | L'autoencodeur a obtenu les meilleures performances. |
| Arsene, Hankins et al. 2019 | Médical, signaux ECG | Comparaison entre une transformée par ondelettes et des réseaux CNN et LSTM | La transformée par ondelettes a été efficace pour supprimer le bruit aléatoire mais moins adapté que le réseau CNN pour les autres bruits. |
| Jhang, Wang et al. 2022 | Médical, signaux ECG | Autoencodeur de débruitage avec pooling par canal (CPDAE) | Trois modèles efficaces développés en fonction de la performance de filtrage et du temps de calcul souhaités. |
| Keshavarzi 2022 | Médical, signaux ECG | Modèles RNN et SVR | Le modèle RNN a obtenus les meilleurs résultats dans tous les cas testés. |
| Jinhuan, Weay et al. | Domaine sismique | Réseau CNN | Il élimine le bruit aléatoire tout en préservant les caractéristiques essentielles sans nécessiter une modélisation préalable. |
| Zarei, Tajeddini et al. 2014 | Détection des défauts dans les roulements des moteurs à induction | Algorithme ADALINE puis algorithme de classification | L'utilisation du modèle ADALINE permet d'améliorer les résultats de l'étape de classification. |
| Williams et Hoel 2003 | Prédiction du traffic routier | SARIMA | Le modèle SARIMA a permis d'obtenir des prédictions fiables à court terme. |
| Privitera, Bellissima et al. 2023 | Domaine pharmaceutique, dosage | ARIMA | Une amélioration de la précision jusqu'à 30% |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Siami-Namini, Tavakoli et al. 2018 | Séries temporelles financières | Comparaison ARIMA et LSTM | Le modèle LSTM peut gérer les dépendances à long terme contrairement au modèle ARIMA. |
| Manju et Sneha 2020 | Médical, signaux ECG | Comparaison entre le filtre de Kalman et le filtre de Wiener | Le filtre de Wiener est plus efficace. |
| Postalcioglu, Erkan et al. 2005 | Systèmes oscillatoires | Comparaison entre le filtre de Kalman et un filtre à ondelettes | Le filtre à ondelettes a été plus efficace pour décomposer le signal et éliminer le bruit. |
| Sayadi, Sameni et al. 2007 | Médical, signaux ECG | Filtre de Kalman étendu | Il permet de filtrer les signaux issus d'un modèle dynamique non linéaire. |
| Louka, Galanis et al. 2008 | Prévisions de vitesse du vent et de production d'énergie | Filtre de Kalman | Les prédictions de la production d'énergie sont plus précises et nécessitent un temps de calcul plus faible. |
| Kumar 2017 | Circulation routière | Filtre de Kalman | Il nécessite moins de données que les modèles ARIMA. |
| Song, Baek et al. 2021 | Prédiction des indices boursiers | Filtrage par transformée de Fourier avec remplissage puis prédictions comparées suivant les modèles : RNN, LSTM et GRU | Le filtrage permet une amélioration significative des résultats de prédictions. Le modèle LSTM est celui qui obtient les résultats les plus précis. |
| Huang, Algahtani et al. 2022 | Consommation d'énergie | Modèles LSTM, SVR et XGBoost | Le modèle SVR a obtenu le coefficient de variation le plus faible. |
| Wei, Zhang et al. 2021 | Consommation d'énergie | Modèles : SVR, XGBoost, LSTM, CNN, perceptron multicouches, forêt aléatoire, K plus proches voisins | Les modèles SVR et XBGoost ont été les plus performants. |

## 2.3 OPTIMISATION DES RÉGULATEURS DANS L'INDUSTRIE

L'apprentissage par renforcement a trouvé des applications dans de nombreux domaines, couvrant une vaste gamme d'utilisations. L'application aux jeux stratégiques comme les échecs et le jeu de Go (Schaul, Quan et al. 2015) sont parmi les exemples les plus célèbres, où il a réussi à surpasser les compétences humaines. Ces succès ont largement contribué à sa renommée. Un autre exemple très connu, développé par Agility Robotics, a utilisé l'apprentissage par renforcement pour apprendre à un robot à marcher de manière autonome (Hurst, Abate et al. 2020). Il s'est également développé dans l'industrie. Il est particulièrement bien adapté à ces contextes en raison de sa capacité à résoudre des problèmes complexes et à gérer des systèmes variés. Cependant, des techniques plus conventionnelles, qui n'impliquent pas d'IA, sont très employés dans ce secteur. En effet, ces techniques sont généralement favorisées dans des environnements où la sécurité et la fiabilité sont primordiales, comme dans l'aérospatiale ou des systèmes industriels critiques. Dans la suite de cette section, nous décrirons différents travaux où l'apprentissage par renforcement et des techniques plus conventionnelles ont été appliquées dans des contextes industriels et à différentes technologies.

Comme introduit dans le premier chapitre de ce mémoire, l'apprentissage par renforcement consiste à effectuer une série d'essais afin d'apprendre en fonction des succès et des échecs de ces essais. L'objectif pour l'agent qui prend les décisions, est de maximiser ses récompenses dans l'environnement avec lequel il interagit. Il ajuste ses actions en fonction des récompenses positives ou négatives qu'il perçoit, afin d'optimiser ses gains futurs. Les décisions sont prises de manière itérative, permettant à l'agent d'améliorer continuellement ses performances. Ce processus suit un principe similaire à l'apprentissage humain, rendant son fonctionnement intuitif et facile à comprendre.

Zeng, Song et al. détaillent l'exemple d'un bras robotisé qui a à disposition une grande variété d'objets de formes différentes, disposés en vrac dans une caisse, qu'il doit lancer vers un emplacement précis (Zeng, Song et al. 2020). Pour réussir cette tâche, il doit tenir compte

de la façon dont il tient l'objet, ainsi que sa forme et son poids. La position de prise de l'objet, c'est-à-dire s'il est maintenu au centre ou aux extrémités va directement influencer la trajectoire du lancer. Il doit donc adapter la force, l'élan et l'angle de lancer pour atteindre sa cible avec précision. Pour cela, il procède par essais et erreurs en combinant les principes de physique à l'apprentissage par renforcement. Des caméras lui permettent de voir les objets qu'il manipule et les résultats de ses lancers. Avec l'entrainement, il améliore sa précision de lancer et apprend à saisir les objets de manière optimale en fonction de leur forme et de la répartition de leur poids. Il est confronté à beaucoup d'objets et de cibles de lancement différentes, ce qui lui permet d'apprendre plus rapidement lorsqu'il rencontre un nouvel objet ou une nouvelle cible. Après 10 000 lancers d'entrainement, sa précision atteint 85 % avec une fiabilité de préhension de 87 %. Durant l'entrainement, il explore également des vitesses de lancer pour tester différentes approches. Les chercheurs augmentent la complexité des objets. Ils commencent par des formes primitives puis passent par des formes plus complexes comme des fruits et légumes et des objets du quotidien. Les premiers résultats avec des formes complexes ne sont pas très bons, mais le bras robotisé s'améliore rapidement et finit par atteindre des performances similaires aux objets simples. La capacité du robot à s'adapter à de nouveaux objets et à de nouveaux emplacements est une caractéristique intéressante de l'apprentissage par renforcement. Son adaptabilité face à des conditions changeantes et imprévisibles le rend utile pour des systèmes influencés par des éléments extérieurs et incontrôlables.

Ogoke et Farimani explorent l'application de l'apprentissage par renforcement profond à la fabrication additive par fusion laser sur lit de poudre (Ogoke and Farimani 2021). Cette technique de fabrication est utilisée pour produire des pièces trop complexes pour être usinées traditionnellement. Cette méthode consiste à fusionner des couches de poudres métalliques avec un laser. Toutefois, les pièces produites présentent souvent des défauts, l'apprentissage par renforcement est donc utilisé pour ajuster les paramètres de fabrication et minimiser ces imperfections. La stratégie de contrôle appliquée va influer sur la vitesse et la puissance du laser, en augmentant par exemple la vitesse ou en diminuant la puissance dans les zones à forte accumulation de chaleur. D'autres méthodes d'optimisation ont été examinées pour des

applications similaires, comme la soudure laser ou d'autres cas de fusion laser, où une gestion précise des paramètres tels que la puissance du laser et la profondeur du point de fusion est importante. Les algorithmes génétiques ont été utilisés pour développer des stratégies de balayage du laser sur la pièce basées sur une grille, afin de réduire les asymétries thermiques et minimiser les défauts (Mohanty, Tutum et al. 2013). Un algorithme permet d'ajuster la puissance du laser en fonction de la conductance géométrique du matériau autour du bain de fusion (Yeung, Lane et al. 2019). La régulation par rétroaction, utilisant des capteurs pour surveiller la fusion en temps réel et ajuster les paramètres selon les critères de qualité est également mise en œuvre (Craeghs, Bechmann et al. 2010). Des capteurs optiques connectés à un circuit logique programmable permettent de surveiller l'intensité et la taille du bain de fusion pour détecter les défauts potentiels (Clijsters, Craeghs et al. 2014).

Dans le travail de Postma, Aarts et al., le contrôle en boucle fermée utilise également un capteur optique pour assurer la traversée complète du faisceau laser dans le matériau à souder (Postma, Aarts et al. 2002). La puissance du laser est ajustée dynamiquement en fonction des fluctuations détectées par le capteur. Cette régulation permet de maintenir une qualité de soudure constante tout en optimisant la vitesse. Renken, Lübbert et al. combinent les prédictions d'un modèle de simulation thermique par éléments finis avec une régulation par rétroaction basé sur les données en temps réel d'un capteur pyroélectrique (Renken, Lübbert et al. 2018). La régulation réalisée permet de réduire la déviation de la température de 73%. Dans l'étude de Forslund, Snis et al., l'approche « gloutonne » (en anglais, *greedy*) divise le problème de l'instabilité de la profondeur du bain de fusion en sous-problèmes pour ajuster les paramètres et maintenir des températures optimales durant le déplacement du faisceau (Forslund, Snis et al. 2021). Des approches basées sur des données et de la modélisation physique sont utilisées pour prédire les champs thermiques couche par couche (Li, Jin et al. 2018) et les caractéristiques des stratégies de fabrication additive (Schwalbach, Donegan et al. 2019). Enfin, les réseaux CNN sont utilisés pour prédire les caractéristiques de profondeur du bain de fusion à partir d'images (Yang, Lu et al. 2019). Liu, Liu et al. analysent la qualité des pièces avec un modèle de régression à vecteur de support (SVR)

combiné à un modèle physique pour prévoir la porosité des pièces imprimées (Liu, Liu et al. 2021).

L'apprentissage par renforcement est fréquemment utilisé pour commander des systèmes complexes composés de multiples capteurs, comme dans le cas des voitures autonomes. Ces véhicules représentent un enjeu majeur car ils cohabiteront avec des véhicules conduits par des humains sur les routes. Il est donc essentiel de garantir une conduite sécuritaire et similaire à celle d'un conducteur humain. Ainsi, l'apprentissage par renforcement a été appliqué pour permettre aux véhicules autonomes d'apprendre en continu et de s'adapter à diverses situations de conduite (Zhu, Wang et al. 2018). Contrairement aux modèles traditionnels qui se basent sur le comportement d'un conducteur moyen et fixent les paramètres, un algorithme d'apprentissage par renforcement est mis à jour continuellement. De nouvelles données lui sont transmises, permettant au modèle de s'adapter à différents scénarios de conduite.

Un autre exemple intéressant d'application de l'apprentissage par renforcement aux véhicules autonomes concerne l'entrainement d'une voiture à sortir seule d'un fossé (Manring and Mann 2022). Les paramètres pris en compte sont la forme du fossé, le glissement des roues avant ou arrière et la rigidité du châssis. L'approche considère quatre cas de patinage : aucune roue ne patine, les quatre roues patinent, uniquement les deux avants ou les deux arrières. Les techniques employées sont le gradient de politique déterministe profond (en anglais : *Deep Deterministic Policy Gradient*, DDPG) (Lillicrap, Hunt et al. 2015) et l'inférence probabiliste pour l'apprentissage par contrôle (en anglais : *Probabilistic Inference for Learning Control*, PILCO) (Deisenroth and Rasmussen 2011). L'agent a réussi à sortir la voiture des différents types de fossés tout en minimisant le dérapage des roues. Il a appris à donner de l'élan en accélérant avec les quatre roues simultanément ou seulement les roues arrière, puis à freiner en cas de glissement. L'apprentissage par renforcement est une technique d'optimisation efficace pour cette tâche pour plusieurs raisons :

- Il fournit de bons résultats sans nécessiter la connaissance exacte du modèle dynamique du système. Cette capacité est avantageuse lorsque le modèle est complexe ou lorsque le contrôle est basé sur des données plutôt que sur un modèle.

- Il offre la possibilité d'explorer une grande quantité d'états et d'actions.

- Il reste performant même lorsque les actions du système sont limitées.

De plus, cette étude montre les limites des méthodes de contrôle classiques telles que le contrôleur PID (Proportionnel Intégral Dérivé) (Visioli 2006) et le régulateur linéaire quadratique (*Linear Quadratic Regulator*, LQR) (Lehtomaki, Sandell et al. 1981) face à des contraintes de contrôle. Ces techniques mesurent l'erreur entre l'état souhaité et l'état atteint, puis calculent l'effort de contrôle nécessaire pour minimiser cette erreur. Cependant, elles ne sont pas adaptées aux systèmes non linéaires ou contraints. En effet, dans certains cas, l'effort nécessaire pour minimiser l'erreur et atteindre l'état souhaité n'est pas exécutable par le système car il a des contraintes. Un exemple particulier est la saturation des variables contrôlées. Dans le cas du fossé, un effort maximal est appliqué à la roue dirigée vers la sortie du fossé. Bien que cela puisse sembler logique, cela peut provoquer une oscillation de la voiture car l'application d'un effort maximal sur une seule roue ne suffit pas pour sortir la voiture du fossé.

L'aspect physique étant important dans l'industrie agroalimentaire, Phate, Malmathanraj et al. ont permis de développer un modèle capable de classer et de peser des fruits de façon indirecte, en utilisant un système de vision par ordinateur (Phate, Malmathanraj et al. 2021). La classification se fait grâce à un modèle à vecteurs de support (SVM), tandis que la pesée indirecte utilise un système adaptatif de déduction neurofloue (ANFIS) optimisé par des approches métaheuristiques. Ce modèle ANFIS est capable de modéliser des relations non linéaires entre les caractéristiques extraites des images de fruits, telles que la longueur, la largeur moyenne, la profondeur, les aires projetées et les périmètres des fruits avec la variable de sortie qui est leur poids. Le modèle s'améliore progressivement avec les données acquises, ce qui lui permet de faire des prédictions de plus en plus précises.

L’optimisation du modèle ANFIS est réalisée par deux méthodes : les algorithmes génétiques (Goldberg and Holland 1988) et l’optimisation par essaims particulaires. Les algorithmes génétiques simulent la sélection naturelle en combinant des paramètres pour produire des générations successives avec combinaisons améliorées. L’optimisation par essaims particulaires consiste à représenter une particule comme étant une solution potentielle dans l’espace des solutions en fonction des résultats obtenus. Cette particule se déplace dans l’espace des solutions et converge ainsi vers la meilleure solution possible. Ces deux méthodes sont utilisées pour trouver la meilleure configuration de paramètres du modèle ANFIS pour minimiser l’erreur des prédictions des poids. Cette approche de pesée indirecte des fruits présente un intérêt pour l’industrie agroalimentaire, car elle réduit la manipulation des produits et accélère le tri des fruits en nécessitant moins d’étapes. Cependant, elle n’est pas applicable à tous les cas où le pesage dynamique est nécessaire.

En ce qui concerne d’autres techniques de pesage, la masse des objets pesés par des convoyeurs de pesage peut être déterminée par des cellules de charge, mais également par des systèmes de compensation de force électromagnétique (*ElectroMagnetic Force Compensation*, EMFC). Contrairement aux systèmes à cellule de charge qui déduisent la masse grâce à un signal électrique proportionnel à la déformation de la cellule de charge sous le poids de l’objet, les systèmes EMFC mesurent la masse en compensant la force gravitationnelle par une force électromagnétique générée par un électroaimant. La masse est alors déterminée en quantifiant le courant nécessaire pour assurer cette compensation et atteindre l’équilibre. Yamakawa et Yamazaki abordent l’optimisation à l’aide d’un modèle dynamique simplifié d’un convoyeur couplé à un EMFC (Yamakawa and Yamazaki 2015). Cette modélisation approxime le système avec un système masse-ressort-amortisseur (Yamakawa, Yamazaki et al. 2009) et la dérivée de l’équation de mouvement. Les données expérimentales sont ensuite utilisées pour estimer les paramètres. L’étude de l’impact des vibrations du sol sur les mesures est également réalisée afin de compenser leurs effets.

Une approche similaire à celle décrite précédemment a été testée pour des systèmes composés de cellules de charge dans le cadre du pesage dynamique sur convoyeur

(Niedźwiecki, Meller et al. 2016). L'application d'une approche d'identification basée sur un modèle masse-ressort-amortisseur de second ordre n'a pas donné des résultats satisfaisants. Des variantes de cette méthode d'identification ont donc été testées pour essayer d'améliorer les résultats obtenus. La première variante consiste à utiliser l'approche d'identification pour modéliser le signal mesuré comme étant la réponse à une impulsion dans un système dynamique du second ordre. Les valeurs des paramètres sont estimées durant chaque cycle afin de définir le poids de l'objet. Cependant, malgré cette modification, les résultats obtenus n'étaient pas satisfaisants. Une deuxième approche comprend l'utilisation d'un filtre passe-bas à coefficients variables dans le temps pour atténuer les perturbations présentes dans le signal. Les valeurs de poids estimées étaient meilleures que celles de l'approche précédente. Enfin, la dernière approche combine l'identification à une technique de pré-filtrage adaptatif du signal pour améliorer la performance du modèle. Les oscillations et les variations lentes indésirables sont atténuées grâce au pré-filtrage, puis les valeurs de poids sont déterminées grâce à l'approche d'identification. Il a été démontré expérimentalement que cette combinaison permet de répondre aux exigences de précision.

Pour mieux cerner notre cas d'application, il est pertinent de s'intéresser à l'utilisation de l'apprentissage par renforcement dans les systèmes de gestion de flux, dont font partis les systèmes d'ensachage.

Des études sur la gestion de flux d'air ont été menées pour réguler la température dans les locaux commerciaux sans recourir à des modèles préétablis (Liu and Henze 2006). En utilisant l'apprentissage par renforcement, l'agent apprend à réguler les systèmes de refroidissement en s'appuyant sur les conséquences des actions précédentes. Il teste diverses actions pour développer une stratégie optimale de gestion du stockage thermique. Ces systèmes de chauffage, ventilation, et climatisation (CVC) sont couramment étudiés pour tester l'apprentissage par renforcement. Fang, Gong et al. appliquent l'apprentissage par renforcement profond pour ajuster en temps réel les points de consigne d'un système CVC (Fang, Gong et al. 2022). Il adapte les points de consigne de température aux conditions intérieures et extérieures changeantes. Le modèle apprend à gérer les températures de

consigne optimales pour minimiser la consommation d'énergie tout en maintenant le confort à l'intérieur. Un environnement de simulation représentant un bâtiment avec différentes zones de volumes variables est mis en place. Les résultats montrent que la stratégie de contrôle développée est plus économique que celle utilisant des points de consigne fixes.

D'autres méthodes pour gérer l'approvisionnement en énergie des bâtiments industriels ont été explorées par Kohne, Ranzau et al. (Kohne, Ranzau et al. 2020), comparant une approche conventionnelle basée sur des règles, une approche utilisant la programmation linéaire en nombres entiers mixtes (MILP) (Rich and Prokopakis 1986, Kondili, Pantelides et al. 1993) et une basée sur les données qui est l'optimisation de politique proximale (PPO). L'approche basée sur des règles s'appuie sur deux points de consignes, qui sont la température inférieure et supérieure permettant d'indiquer s'il faut activer ou désactiver les sources de chauffage et de refroidissement. La programmation MILP utilise des équations linéaires avec des contraintes de décisions entières. Le PPO est un algorithme d'apprentissage par renforcement profond réputé pour sa stabilité (Schulman, Wolski et al. 2017), qui optimise progressivement la politique en effectuant des mises à jour proches pour éviter les changements brusques et atteindre une politique optimale stable. Pour un système moins complexe, l'approche PPO a réalisé des économies de 50% sur les coûts directs d'approvisionnement en énergie, comparé à 13% pour le MILP. Lors de la prise en compte des coûts indirects, comme les coûts de changement et les pénalités de dépassement de la température, les bénéfices sont de 33% pour le PPO et de 8% pour le MILP. Pour un système plus complexe, les réductions des coûts directs sont modestes (3% pour le MILP et 6% pour le PPO), mais les coûts indirects peuvent augmenter de plus de 50% par rapport à une méthode conventionnelle. Ainsi, pour des systèmes moins complexes, l'apprentissage par renforcement profond est préférable. Toutefois, pour les systèmes plus complexes, une approche conventionnelle peut être plus approprié pour éviter les coûts indirects élevés associés à certains équipements (stockage d'énergie thermique, pompes à chaleur, etc.). Cette étude montre que l'optimisation par apprentissage par renforcement n'est pas toujours la meilleure ou la plus adaptée et peut dépendre du fonctionnement des composants du système étudié.

L’apprentissage par renforcement peut donc être employé dans des systèmes équipés de multiples capteurs pour optimiser certaines fonctionnalités, comme dans le cas des véhicules hybrides rechargeables pour gérer de manière optimale la répartition entre l’utilisation du carburant et de l’électricité (Qi, Luo et al. 2019). Contrairement aux systèmes qui utilisent des règles prédéfinies ou qui se basent sur des prédictions des conditions de circulation, l’apprentissage par renforcement permet une adaptabilité face à des conditions changeantes. Le système apprend ainsi de manière autonome la répartition optimale entre carburant et électricité en se basant sur les interactions du véhicule avec son environnement. Cette approche a permis de réaliser une économie de carburant de 16.3% en moyenne par rapport aux méthodes rapport aux méthodes basées sur des règles prédéfinies lors de tests sur des données réelles.

Au début des années 2000, une application pratique a été testée aux Pays-Bas sur trois systèmes de distribution d’eau (Bhattacharya, Lobbrecht et al. 2003). Le modèle développé combinait les réseaux de neurones artificiels et l’apprentissage par renforcement. Les résultats de cette étude ont encouragé l’utilisation de l’apprentissage par renforcement dans ce domaine, en démontrant que l’erreur entre le niveau d’eau calculé et le niveau souhaité était négligeable.

Des recherches ultérieures ont également exploré le contrôle en temps réel de pompes dans les systèmes de distribution d’eau (Hajgató, Paál et al. 2020). Des méthodes d’optimisation pour déterminer les vitesses optimales des pompes ont été comparées à l’apprentissage par renforcement profond. Dans cette approche, l’agent utilise des données reçues en temps réel pour ajuster les vitesses des pompes. Les résultats de cette étude ont montré que l’apprentissage par renforcement permettait une exécution deux fois plus rapide et une efficacité totale supérieure à 0,98. Il a ainsi été démontré que l’apprentissage par renforcement est efficace pour améliorer la performance de certaines opérations dans la gestion de l’eau.

Le Tableau 5 donne un récapitulatif des travaux discutés dans cette section, en détaillant les domaines abordés, les techniques utilisées, ainsi que les conclusions principales tirées de ces études.

Tableau 5. Aperçu de travaux d'optimisation des régulateurs dans l'industrie

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Schaul, Quan et al. 2015 | Jeu Atari | Apprentissage par renforcement profond et relecture d'expérience priorisée | L'expérience priorisée permet d'atteindre une meilleure performance plus rapidement sur 41 des 49 jeux testés par rapport à une relecture uniforme |
| Zeng, Song et al. 2020 | Bras robotisé pour trier des objets par lancés | Apprentissage par renforcement profond combiné à un modèle physique | Après 10 000 lancers d'entrainement, précision de 85% avec une fiabilité de préhension de 87% et est capable de s'adapter à de nouveaux objets et cibles. |
| Ogoke et Farimani 2021 | Fabrication additive (fusion laser sur lit de poudre) | Apprentissage par renforcement profond | L'apprentissage par renforcement a besoin de moins de puissance de calcul que les autres méthodes |
| Mohanty, Tutum et al. 2013 | Fabrication additive (fusion laser sur lit de poudre) | Algorithme génétique | Meilleure homogénéité thermique obtenue tout en réduisant les zones surchauffées |
| Yeung, Lane et al. 2019 | Fabrication additive (fusion laser sur lit de poudre) | Algorithme de contrôle basé sur le calcul d'un facteur | Amélioration de la finition des surfaces |
| Craeghs, Bechmann et al. 2010 | Fabrication additive (fusion laser sur lit de poudre) | Régulation par rétroaction | Améliore la qualité des pièces, notamment dans des conditions de fabrication non optimales |
| Clijsters, Craeghs et al. 2014 | Fabrication additive (fusion laser sur lit de poudre) | Capteurs optiques connectés à un circuit logique programmable | Surveille la qualité du bain de fusion en temps réel |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Postma, Aarts et al. 2002 | Fabrication additive (fusion laser sur lit de poudre) | Contrôle en boucle fermée et capteurs optiques | Vitesse optimisée en ajustant l'intensité du laser afin de ne pas perdre la qualité |
| Renken, Lübbert et al. 2018 | Fabrication additive (fusion laser sur lit de poudre) | Régulation par rétroaction combiné à une prédiction basée sur un modèle de simulation thermique par éléments finis | Réduction de 73% de la déviation de la température et optimisation en temps réel possible |
| Forslund, Snis et al. 2021 | Fabrication additive (fusion laser sur lit de poudre) | Approche "gloutonne" (en anglais : *greedy*) | Optimise la distribution de chaleur en améliorant l'accumulation de la chaleur et la profondeur de fusion non uniforme |
| Li, Jin et al. 2018 | Fabrication additive (fusion laser sur lit de poudre) | Combine modélisation basée sur des données et sur des principes physiques | Prédiction précise du champ thermique pour des conditions variables de fabrication |
| Schwalbach, Donegan et al. 2019 | Fabrication additive (fusion laser sur lit de poudre) | Modélisation et simplifications physiques et numériques | Optimisation des paramètres de fabrication |
| Yang, Lu et al. 2019 | Fabrication additive (fusion laser sur lit de poudre) | Classificateur basé sur un réseau de neurones convolutifs (CNN) | Le temps de traitement des images par rapport à une méthode traditionnelle est réduite de 90% et la précision de la prédiction est de 91%. |
| Liu, Liu et al. 2021 | Fabrication additive (fusion laser sur lit de poudre) | Modèles de régression à vecteur de support (SVR) testées avec différentes données (paramètres de réglages, effets physiques et combinaison entre les deux) | Le modèle comprenant les paramètres machine et les effets physiques associés a permis d'obtenir les meilleures performances |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Zhu, Wang et al. 2018 | Véhicules autonomes | Gradient de politique déterministe profond (DDPG) | Erreurs de vitesse de 5% et de distance de 18%, ce qui est plus faible que des modèles de régression ou conventionnels |
| Manring and Mann 2022 | Véhicules autonomes, sortir d'un fossé | Gradient de politique déterministe profond (DDPG), inférence probabiliste pour l'apprentissage par contrôle (PILCO) | L'agent réussit à sortir la voiture en minimisant le dérapage. |
| Phate, Malmathanraj et al. 2021 | Industrie alimentaire pesage indirect de fruits | Vision par ordinateur, modèles SVM, ANFIS | Cette approche permet de réduire la manipulation des fruits et le temps nécessaire pour le tri. |
| Yamakawa et Yamazaki 2015 | Pesage sur convoyeur EMFC | Modélisation masse-ressort-amortisseur et dérivée de l'équation de mouvement | Le modèle s'adapte afin de compenser l'impact des vibrations sur les mesures. |
| Niedźwiecki, Meller et al. 2016 | Pesage dynamique sur convoyeur à cellules de charge | Approche d'identification pour modéliser le signal, filtre passe-bas à coefficients variables, combinaison d'identification et de pré-filtrage | L'approche qui combine l'identification et le pré-filtrage a permis de répondre aux exigences de précision, les autres non. |
| Liu and Henze 2006 | Gestion thermique des bâtiments | *Q-Learning* | Algorithme de contrôle efficace dans ce milieu |
| Fang, Gong et al. 2022 | Gestion des systèmes de chauffage, ventilation et climatisation (CVC) | Approche *Deep Q-Network* | Trouve une stratégie de contrôle plus économe que lorsque des points de consignes sont utilisés |

| Auteur | Domaine | Techniques utilisées | Conclusions |
|---|---|---|---|
| Kohne, Ranzau et al. 2020 | Approvisionnement en énergie des bâtiments industriels | Comparaison : apprentissage par renforcement profond (optimisation de politique proximale, PPO), programmation linéaire en nombres entiers mixtes (MILP) et approche avec points de consignes | Système moins complexe : apprentissage par renforcement est le meilleur. Pour un système plus complexe il vaut mieux préférer l'approche avec points de consignes pour éviter les surcouts |
| Qi, Luo et al. 2019 | Véhicules hybrides rechargeables | Deep Q-Network comparé avec des méthodes basées sur des règles et sur de la prédiction | Économie de carburant de 16,3% par rapport aux méthodes conventionnelles |
| Bhattacharya, Lobbrecht et al. 2003 | Systèmes de distribution d'eau | Apprentissage par renforcement pour réduire l'erreur de réseaux de neurones artificiels | Stratégie de contrôle optimale trouvée |
| Hajgató, Paál et al. 2020 | Systèmes de distribution d'eau | Apprentissage par renforcement profond, Dueling DQN (variante du modèle Deep Q-Network) | Apprentissage par renforcement : exécution deux fois plus rapide et efficacité totale de plus de 0,98 |

## 2.4 SYNTHÈSE DU CHAPITRE

Le filtrage des données issues des cellules de charge est essentiel au pesage dynamique. Les diverses études présentées dans ce chapitre illustrent la possibilité d'appliquer l'IA à différents problèmes de filtrage dans de multiples domaines. Malgré cette diversité, des problématiques similaires émergent souvent. Ces exemples montrent une bonne efficacité des modèles d'apprentissage supervisé dans le traitement des données brutes pour

éliminer le bruit et améliorer les performances des tâches gérées par l'API du système. Bien que les filtres traditionnels apportent des améliorations dans de nombreuses situations, leur efficacité reste limitée face à des signaux plus complexes, où l'utilisation de filtres basés sur l'IA a démontré une précision accrue.

Les recherches dans les secteurs industriel et de gestion de flux ont révélé des résultats prometteurs quant à l'application de l'apprentissage par renforcement pour la commande de systèmes complexes. La grande variété d'applications montre que l'apprentissage par renforcement pourraient être intégrée dans un système d'ensachage-pesage.

Dans le chapitre suivant, nous présentons le fonctionnement du système étudié et décrivons les données collectées, ainsi que la problématique que nous souhaitons résoudre. Nous aborderons également les différentes approches envisagées pour la résoudre.

# CHAPITRE 3
# DESCRIPTION DE L'APPROCHE DE RÉSOLUTION

Ce chapitre commence par un aperçu du contexte du pesage industriel. Il poursuit en décrivant en détail le fonctionnement de le balance industrielle, en se concentrant sur le déroulement des cycles de pesage. Enfin, il aborde la mise en place de la collecte de données et présente la méthodologie utilisée pour aborder et résoudre les divers aspects du problème.

## 3.1 CONTEXTE

### 3.1.1 L'entreprise partenaire

L'entreprise partenaire comprend une filière « Systèmes Automatisés » qui a pour objectif de fournir des solutions de dosage, d'ensachage et de palettisation pour différents types de produits. Il s'agit de l'un des plus importants fournisseurs d'équipements d'emballage et de traitement des produits en vrac. Les principales catégories d'équipement qu'elle fabrique sont :

- Des peseuses-ensacheuses industrielles qui permettent de traiter une large gamme de matière, que ce soit en poudre ou sous forme de granulés afin de fournir différents systèmes de remplissage de sacs sous différents formats. En fonction de la valeur des produits à ensacher et donc de ce qui est privilégié entre la précision ou la vitesse, le dosage est réalisé de façon massique ou volumique.
- Des équipements de palettisation,

- Des emballeuses de charges palettisées

Parmi ses différentes catégories, elle se distingue dans l'industrie des peseuses-ensacheuses, avec notamment plusieurs milliers d'unités vendues dans ce domaine. Un de ses arguments est qu'elle propose des solutions permettant de traiter une grande gamme de types de produits différents comme des granulés, des poudres, des flocons, des matières fibreuses ou ayant de plus grandes dimensions. Son objectif étant de fournir des machines rapides et précises pour chaque application.

### 3.1.2 Fonctionnement de la balance industrielle

La Figure 3 offre un aperçu du système et des différents éléments qui le compose.

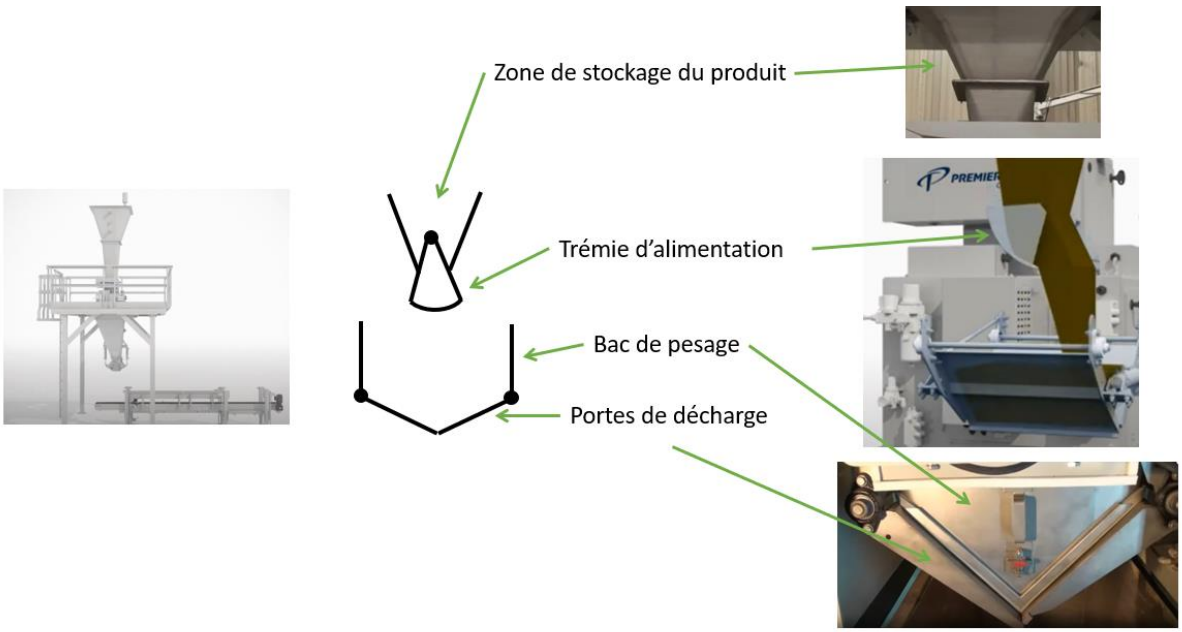



Figure 3. Schéma légendé du système

Avant le pesage, le produit est acheminé dans la zone de stockage du produit. Son écoulement est bloqué par la trémie d'alimentation qui est fermée. Lorsque le cycle de pesage

est lancé, la trémie s'ouvre et le produit s'écoule dans le bac de pesage. Les portes de décharge sont fermées, le produit s'accumule donc dans le bac jusqu'à atteindre le poids souhaité. Une fois le poids cible atteint, les portes de décharge s'ouvrent afin que le produit soit mis en sac. Une explication plus détaillée du système est réalisée par la suite à l'aide du langage de modélisation SysML.

Le langage de modélisation graphique SysML, qui signifie « Systems Modeling Langage », permet de modéliser des systèmes afin de faciliter leur compréhension et la représentation des interactions entre les différents éléments qui les composent. Ce langage est basé sur le langage UML (Unified Modeling Langage) (Booch, Rumbaugh et al. 2005) et comporte différents diagrammes qui permettent de modéliser différents aspects des systèmes. Les éléments théoriques utilisés dans cette partie sont issus en grande majorité du livre de Roques (Roques 2009). Le langage SysML est donc appliqué pour représenter le fonctionnement de la balance industrielle. Parmi les différents diagrammes, ceux qui seront utilisés sont les diagrammes de cas d'utilisation, le diagramme de définition de blocs et le diagramme d'activité pour contrôler le pesage.

- **Le diagramme de cas d'utilisation :**

Le diagramme de cas d'utilisation permet de résumer les différents cas d'utilisation du système d'ensachage industriel et les interactions qu'il peut avoir avec ses utilisateurs et d'autres systèmes. Cela comprend également les exceptions qui peuvent se produire en dehors de son fonctionnement normal.

Ainsi ce diagramme décrit les utilisations pour lesquelles le système est employé, mais il ne précise pas comment il les effectue. En termes de schématisation, les formes ovales représentent les cas d'utilisation, les dessins de personnage en bâton désignent les acteurs et les lignes permettent d'associer ces éléments pour montrer la façon dont l'acteur participe à cette utilisation. Le rectangle englobant les différentes utilisations représente le système. Le terme d'acteur peut également faire référence à un système connexe, qui est alors représenté

par un rectangle. Les différentes utilisations peuvent être reliées entre elles suivant trois relations distinguables : la relation d'inclusion, la relation d'extension et la relation de généralisation.

La relation d'inclusion se représente par une flèche pointillée sur laquelle apparait le mot « *include* ». Cette relation permet de montrer que l'utilisation écrite à la base de la flèche comprend toujours le comportement décrit par le cas d'utilisation qui est pointé par cette flèche.

La relation d'extension est représentée graphiquement de la même façon que la précédente, mais avec le mot « *extend* ». Elle a pour rôle de montrer que le cas d'utilisation peut avoir son comportement étendu par le cas d'extension, désigné par la base de la flèche, sous certaines conditions.

Quant à la relation de généralisation, elle est symbolisée par une flèche en trait continu avec une pointe blanche. Une relation de généralisation entre deux cas d'utilisation signifie que l'un est un cas spécial de l'autre.

Dans notre système d'ensachage industriel, l'opérateur est le seul intervenant avec le système, particulièrement dans le cas d'utilisation concernant la gestion du processus d'ensachage (voir Figure 4). Ce cas d'utilisation englobe les cas d'utilisation suivants : le paramétrage lié à un produit, le lancement et l'arrêt du processus d'ensachage, ainsi que la résolution de problèmes éventuels survenant en cours d'opération. Le système comprend donc également le pesage du produit à ensacher comme cas d'utilisation qui inclut nécessairement un tarage automatique. En cas de dysfonctionnement, une alarme est déclenchée pour signaler un problème lors de l'action de pesage. Il s'agit d'une relation d'extension entre le pesage du produit et le déclenchement d'une alarme.

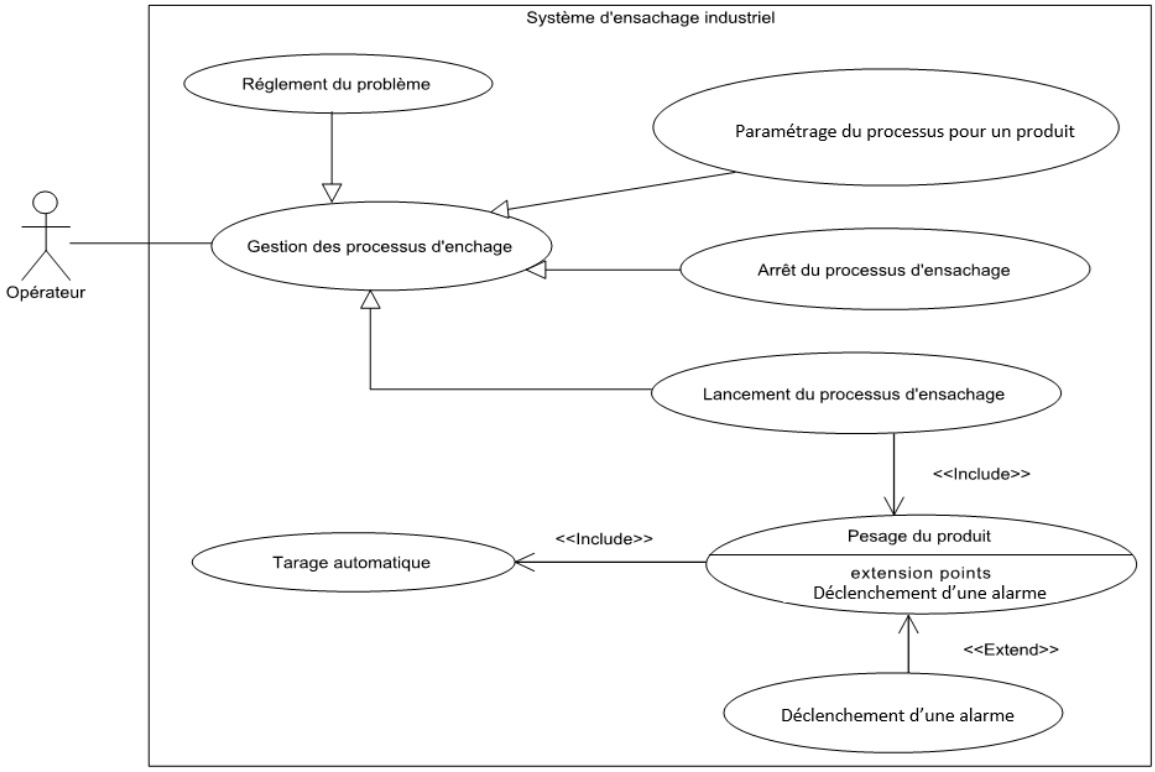



Figure 4. Diagramme de cas d'utilisation

- **Le diagramme de définition de blocs :**

Le diagramme de définition de blocs est un diagramme qui permet de décrire les sous-ensembles du système et les relations qui lient ces sous-ensembles entre eux. Il est composé de blocs qui peuvent représenter un système, un sous-système ou un composant, dont les propriétés et les relations entre blocs sont détaillées. Les propriétés qui sont précisées peuvent être des propriétés de valeurs ou de parties. Les valeurs (en anglais, *value properties*) sont les caractéristiques quantifiables telles que des dimensions, des unités, des paramètres ou des propriétés géométriques. Les parties (en anglais, *part properties*) permettent de décomposer les blocs en parties plus petites.

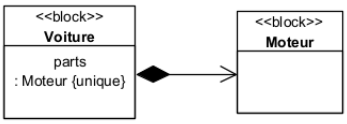
Graphiquement, les blocs sont symbolisés par un rectangle souvent scindé en deux ou plus. La partie du haut est composée du nom du bloc et est obligatoire. Les parties en dessous

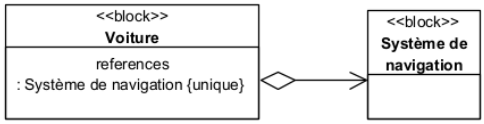
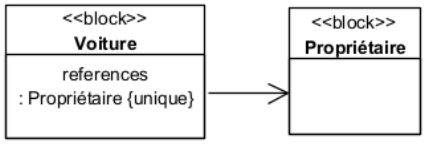
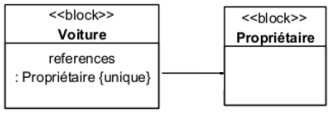
du titre sont facultatives et comportent toutes les autres informations. Parmi ces informations, on retrouve : les propriétés de valeur et les parties. Il est possible de distinguer différents types de relations entre un bloc et ses parties : composition, agrégation et association. La description de ces trois éléments est regroupée dans le Tableau 6.

Avant de définir les différentes relations, le principe de multiplicité est détaillé. La multiplicité permet de définir les contraintes concernant le nombre d'éléments qui peuvent être associés à un autre élément dans une relation. Les valeurs de multiplicités possibles sont :

- « 1 » : Il n'y a qu'un seul élément qui peut être associé.
- « 0..1 » : Soit il n'y a pas d'éléments, soit il y en a un qui peut l'être.
- « 0..* » : Le nombre d'éléments pouvant être associés va de zéro à plusieurs possibles.
- « 1..* » : Il peut y en avoir un ou plusieurs.

Tableau 6. Éléments du diagramme de définition de blocs

| **Relation** | **Description et exemple** | **Multiplicité** | **Représentation** |
|---|---|---|---|
| Composition | La relation de composition est une relation forte où la partie concernée est une partie intégrante du bloc auquel elle est rattachée. Ils ne peuvent exister indépendamment. Exemple : La voiture et son moteur qui ont une relation de composition, car il ne peut pas être partagé par une autre voiture. | Il ne peut y avoir que « 1 » ou « 0..1 » du côté du tout et « 1 » du côté de la partie. |  Un losange noir est du côté du tout et il n'y a rien du côté de ses parties. |

| Relation | Description et exemple | Multiplicité | Représentation |
|---|---|---|---|
| Agrégation | La relation d'agrégation est une relation plus faible où l'élément du bloc et ses parties peuvent exister indépendamment. De plus, les parties peuvent être partagées. Exemple : La voiture et un système de navigation, le bon fonctionnement de la voiture n'en dépend pas et ils peuvent exister séparément. Le même système de navigation peut être utilisé sur une autre voiture. | Il n'y a pas de contrainte de multiplicité pour cette relation. |  Un losange vide est positionné du côté du bloc qui reçoit la partie. |
| Association | La relation d'association permet de définir une relation d'égal à égal entre blocs. Exemple : La voiture et son propriétaire. | Dans ce cas, la multiplicité est utile à titre informatif. Dans l'exemple de la voiture, on pourrait montrer que la voiture a un seul propriétaire qui peut posséder plusieurs voitures. |  Un trait simple relie les deux blocs.  Si le trait n'a pas de flèche et seulement un trait alors l'association est bidirectionnelle ou non spécifiée. |

Le diagramme de définition de blocs de notre système est présenté Figure 5. Dans le diagramme de définition de blocs, la balance industrielle est représentée avec ses sous-systèmes et leurs composants respectifs. Tous ces sous-ensembles sont essentiels au fonctionnement de la balance et sont interdépendants. Ils sont donc reliés par une relation de composition. Les liens entre les différents composants des sous-ensembles sont liés à l'aide de relations d'association. Les opérations ne sont pas représentées dans ce diagramme.

Le module d'alimentation de la matière comprend donc le contenant de stockage amont, dans lequel le produit à stocker est retenu avant d'être pesé. L'écoulement du produit dans le bac de pesage est contrôlé par la porte de la trémie d'alimentation qui est actionnée par un servomoteur. Selon les modèles du système, il peut y avoir une ou deux portes. Une fois que le produit s'écoule par la trémie, il tombe dans le bac de pesage. Ensuite, les trois cellules de charge mesurent la quantité de produit et lorsque le poids cible est atteint, le produit est évacué pour être ensaché. Pour finir, les portes de libération du produit s'ouvrent grâce au vérin pneumatique. Tous ces composants sont dirigés par l'API de pesage qui fait partie de la commande centrale.

La commande centrale comprend également l'écran et le clavier, permettant ainsi la communication entre l'opérateur et la balance. Pour garantir la sécurité et la conformité, des dispositifs tels que le disjoncteur et l'alarme sont également présents sur le système.

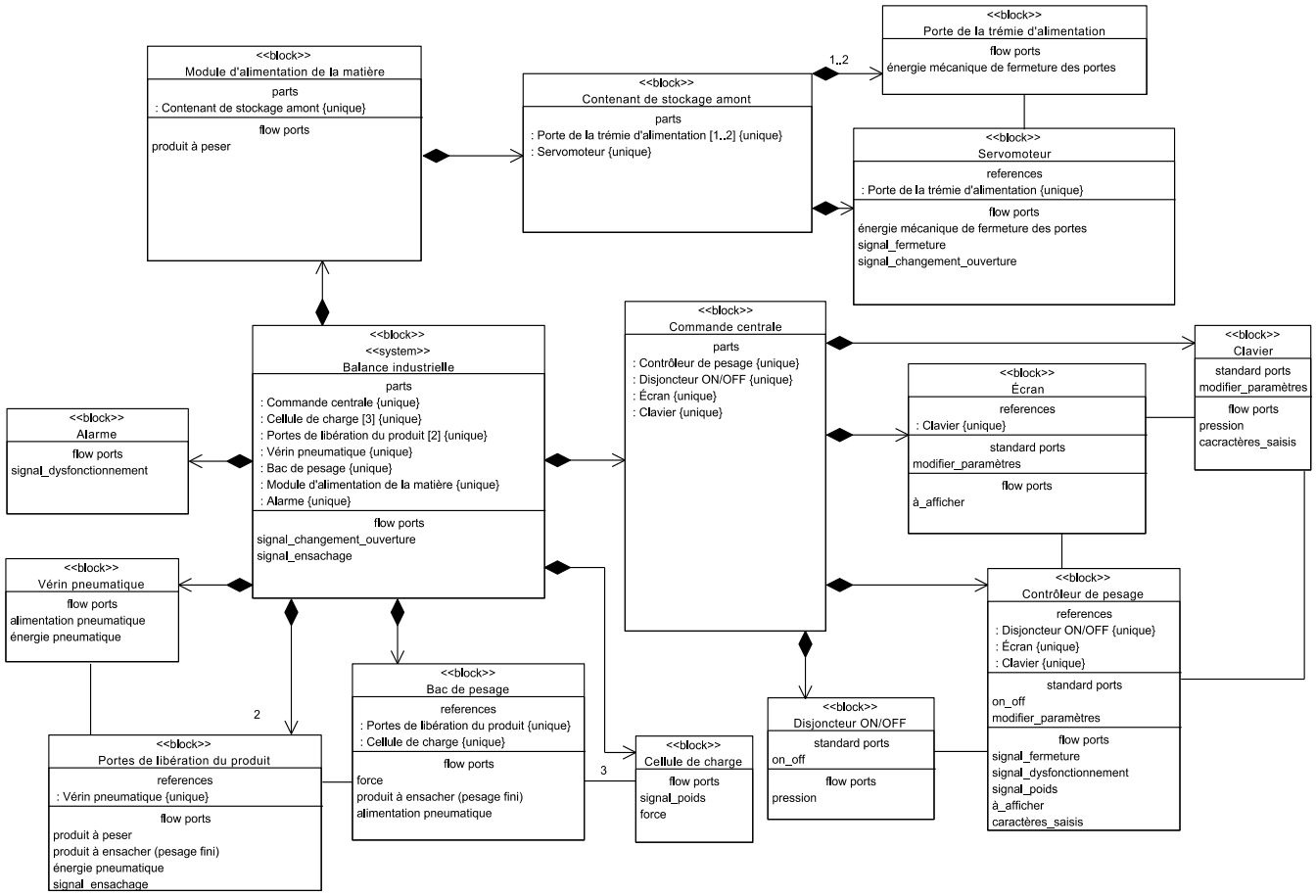



Figure 5. Diagramme de définition de blocs de la balance industrielle

- **Le diagramme d'activités :**

Le diagramme d'activités regroupe les éléments suivants : des activités, des actions et des flux entre ces actions. Les activités fournissent un contexte aux actions et aux flux. Parmi les éléments qui transitent entre les actions, il peut y avoir des données ou des éléments physiques. La description du diagramme d'activité pour contrôler le pesage sur la balance industrielle permettra de présenter les différents composants de ce diagramme.

Au début du diagramme, le branchement parallèle représenté par un rectangle vertical noir, appelé nœud de jonction ou « fork », permet de réaliser simultanément les actions de mise en mémoire tampon du signal et d'extraction de segments des signaux. Pour l'action de mise en mémoire tampon du signal, celle-ci perçoit de façon continue le signal qui provient des cellules de charge. Le paramètre « *stream* », est représenté par un petit carré présent sur l'activité « mettre en mémoire tampon le signal », indique une entrée de flux de données et permet au port de recevoir les données de façon continue tant que ce paramètre est activé.

En parallèle, la seconde action réalisée a pour but d'extraire des segments de données qui sont ensuite filtrés. Une fois les segments de poids filtrés, le système lit le poids capté sur le moment et décide ainsi de l'action suivante à effectuer. Le nœud de décision qui suit permet d'adapter l'action à réaliser en fonction de l'état dans lequel le cycle de pesage se trouve. Dans le cas où le cycle de pesage est à la fin du grand débit, l'action de changement d'ouverture est exécutée. Pour ce faire, un signal est envoyé par l'API au servomoteur qui gère les portes de la trémie pour ajuster l'ouverture. Une fois ce changement effectué, l'action « passer en petit débit » est exécutée. Cela est visible sur le diagramme d'activités grâce à l'élément de la Figure 7.

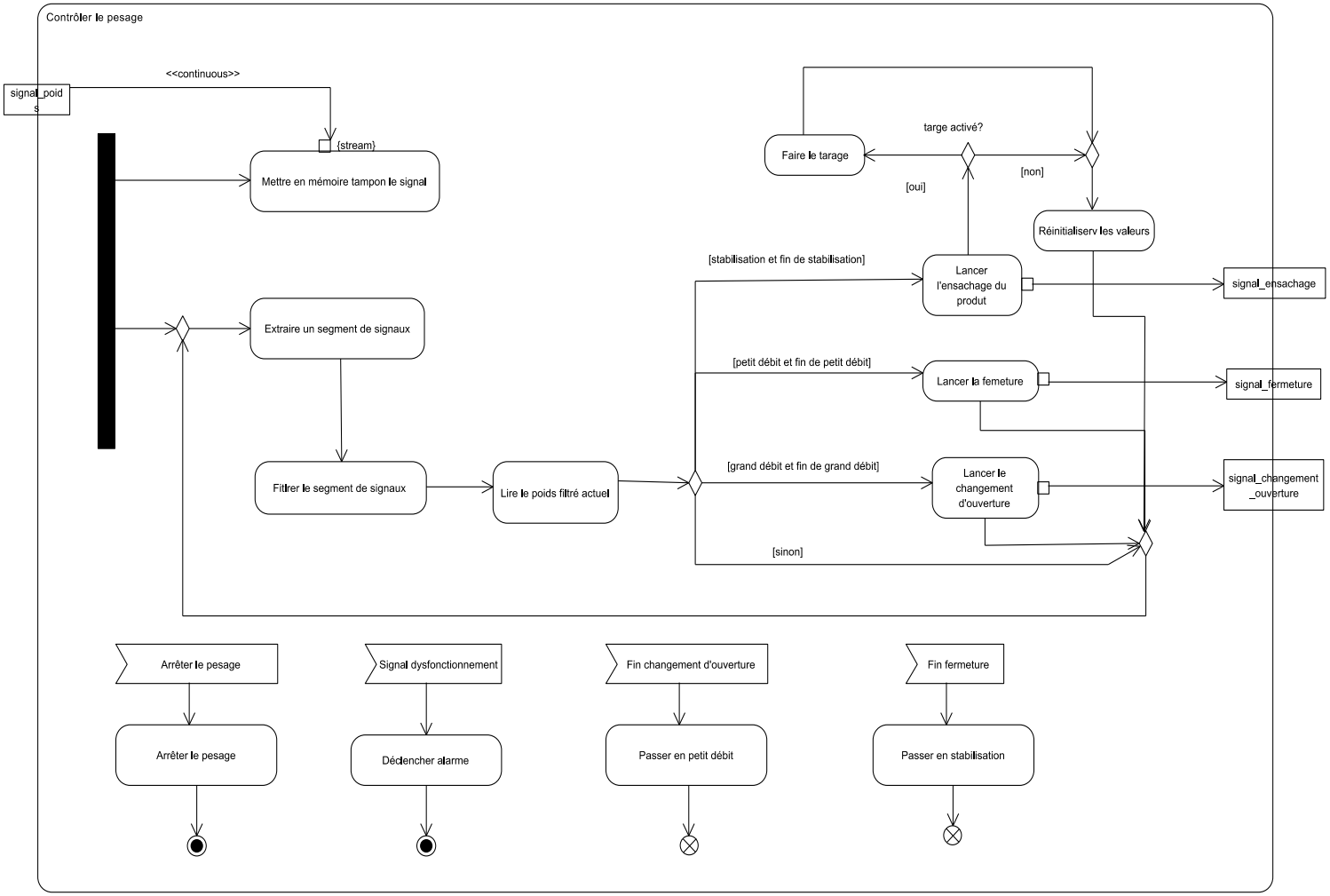



Figure 6. Diagramme de l’activité contrôler le pesage

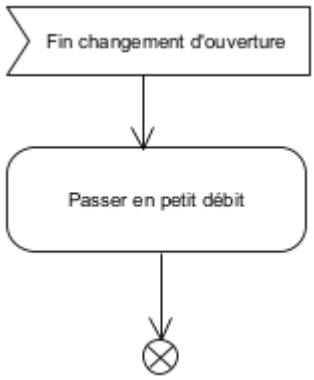



Figure 7. Extrait du diagramme d'activités

Il en est de même pour la fin de petit débit qui amène à la fermeture et à l'exécution de la stabilisation. À la fin de la stabilisation, si la tare est activée alors elle est réalisée, puis l'ensachage est déclenché grâce à un signal envoyé pour l'ouverture des portes de décharge. La tare n'est pas activée à chaque cycle. La fréquence à laquelle elle est effectuée est adaptée en fonction du produit conditionné. Il s'agit d'un choix réalisé par l'entreprise afin d'avoir une cadence d'ensachage plus élevée. Enfin, les valeurs sont réinitialisées pour permettre le démarrage d'un nouveau cycle. L'exécution de toutes ces actions détaillées est réalisée en fonction des segments de poids qui sont transmis à chaque instant. Dans le cas où l'arrêt du pesage est activé, alors le cycle est arrêté. De plus, une alarme est déclenchée si un dysfonctionnement est détecté. Dans les deux cas, cela marque l'arrêt de l'activité « contrôler le pesage ».

### 3.1.3 Description du cycle de pesage de la balance

La machine d'ensachage utilise les cycles de remplissage « vrac » et « goutte-à-goutte» (*Bulk & Dribble Filling*). Le cycle de remplissage en vrac est la vitesse rapide et le cycle de remplissage goutte à goutte est la vitesse lente. On parle aussi du concept de point

de consigne double (Dual Set-point). Le premier point de consigne marque la fin de la partie rapide du cycle de remplissage (*Bulk Cut Off*) et le second point de consigne marque la fin de la partie lente du cycle de remplissage (*Dribble Cut Off*). La machine d'ensachage peut être configurée par deux catégories de paramètres : physique et logique (2020). Les paramètres physiques sont :

- La matière première.
- Le type de trémie d'alimentation (une porte ou deux portes).
- Le type de sac, c'est-à-dire le poids visé.

Les paramètres logiques sont divisés en deux sous-catégories : ceux configurables par l'opérateur de la machine à travers une interface utilisateur et ceux définis par l'API de la machine. La première sous-catégorie contient les paramètres suivants :

- Le poids nominal qui est aussi désigné comme étant le poids de déchargement (*Discharge weight*)
- L'intervalle de poids accepté (*Overweight* et *Underweight*) : Limite basse et limite haute de produit pesé. Si ces limites sont atteintes, cela fera apparaître un dysfonctionnement sur la machine afin que l'opérateur puisse intervenir en mettant de côté le sac non conforme et en réajustant certains paramètres si cela est nécessaire.
- Le temps de déchargement (*Discharge time*) : temps requis pour vider le bac de pesage vers l'ensacheur. Étant donné qu'à ce moment-là les portes de décharge sont ouvertes sur le sac, cela permet d'être certain que la machine ne va pas ouvrir les portes de la trémie d'alimentation et rendre la pesée non conforme.
- Le pourcentage d'ouverture de la trémie d'alimentation lors de l'écoulement en grand débit (*Maximum feeding*), qui va agir sur la vitesse de l'écoulement en grand débit.

- Le pourcentage d'ouverture de la trémie d'alimentation lors de l'écoulement en goutte-à-goutte (*Minimum feeding*), qui va agir sur la vitesse d'écoulement en goutte-à-goutte.

La deuxième sous-catégorie contient les paramètres suivants :

- L'arrêt (ou point de consigne) du grand débit (*Bulk feed cut off*) : Poids où l'écoulement du produit dans la balance avec le plus grand débit s'arrête et passe ensuite à un débit plus petit.

- L'arrêt (ou point de consigne) du petit débit (*Dribble feed cut off*) : Poids d'arrêt de l'écoulement en goutte-à-goutte, ce qui implique l'arrêt total de l'écoulement du produit pour le remplissage en cours.

L'analyse du cycle de pesage a permis de distinguer huit phases suivies par l'algorithme dans le cycle d'une pesée. La Figure 8 permet de représenter l'évolution du poids filtré, à partir duquel le système prend ses décisions, et les différentes phases correspondantes. Le cycle se déroule de la façon suivante :

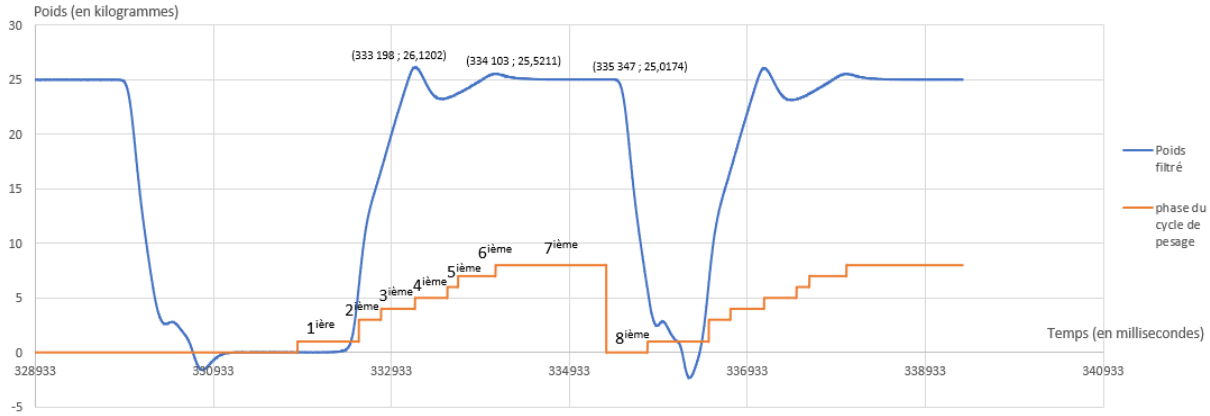



Figure 8. Graphe regroupant les valeurs de poids filtrés et les valeurs des phases correspondantes de cycles de pesage

Première phase : Une pesée débute par l'ouverture en grand débit de la trémie d'alimentation et la matière commence alors à s'écouler dans le bac de pesage.

Deuxième phase : La phase qui suit permet à la balance de déterminer la valeur de poids à laquelle la machine va changer l'ouverture de la trémie d'alimentation en petit débit. La détermination de cette étape se fait en fonction du débit.

Troisième phase : Lorsque la valeur de poids marquant la valeur de poids déterminée pour la fin du grand débit est atteinte par le poids filtré, alors le changement d'ouverture de la trémie d'alimentation est décidé.

Quatrième phase : Dans cette quatrième phase, le changement d'ouverture se fait.

Cinquième phase : Lorsque l'écoulement en petit débit est atteint, une analyse du débit est alors réalisée dans cette étape afin que la décision de fermeture de la trémie d'alimentation soit prise au bon moment.

Sixième phase : Lors de cette sixième étape, la fermeture de la trémie d'alimentation est opérée et dure le temps que le produit continue de s'écouler entre le moment où la décision est prise par la machine et la fin de l'écoulement de matière dans le bac de pesage.

Septième phase : Dans cette étape, un temps de stabilisation est marqué par la machine afin de vérifier le poids final de la pesée avant d'ouvrir les portes de décharge pour la mise en sac.

Huitième phase : La dernière étape de pesée consiste en l'ouverture des portes de déchargement du produit dans le sac puis la fermeture de celles-ci. Il est possible d'activer un paramètre permettant d'effectuer une tare de façon récurrente tous les dix cycles par exemple, la fréquence étant à définir lors du paramétrage. Dans les cycles tracés sur la Figure 8, une tare est effectuée avant de commencer la première phase, cela est remarquable par la durée de phase présente avant le lancement de la nouvelle pesée. Pour cela, le système attend que le poids à vide se stabilise et une rectification de la tare est effectuée si cela est nécessaire.

La tare n’est pas réalisée à la suite de la nouvelle pesée effectuée. En effet, aucune attente n’est remarquable, la pesée suivante est donc lancée directement à la suite.

Ainsi, le système prend ses décisions en s’appuyant sur les valeurs de poids filtré, mais il est intéressant de faire le parallèle entre les valeurs de poids filtrés et les valeurs de poids bruts. Le poids brut correspond aux valeurs perçues par les cellules de charge avant que les données ne soient lissées pour les rendre plus exploitables par le système. Cela permet donc de se rendre compte des vibrations induites et captées par le système. La Figure 9 regroupe ces deux courbes de valeurs pour les mêmes cycles que ceux présents dans la Figure 8.

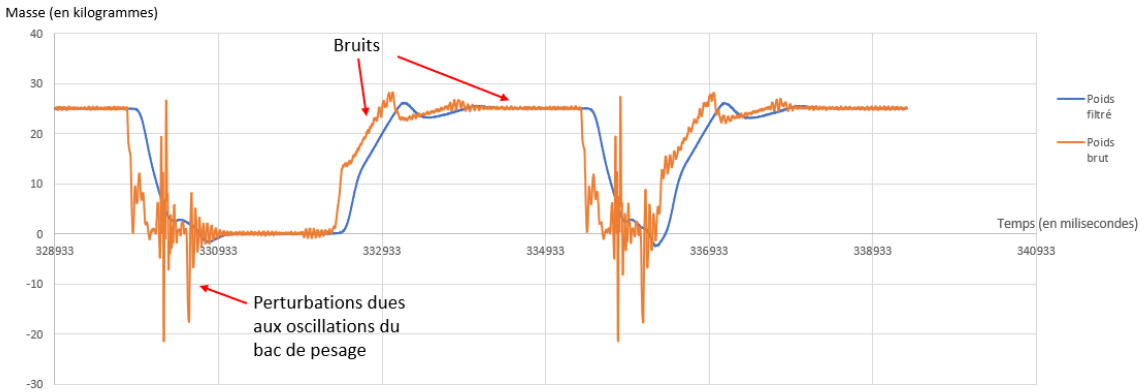



Figure 9. Tracés des valeurs de poids bruts et de poids filtrés de cycles de pesage

Figure 9 permet donc de visualiser les perturbations dues aux oscillations naturelles du bac et les bruits captés par le système. Ces éléments sont traités par le filtre présent dans l’API. Cependant, le programme présent dans l’API ayant été conçu par une partie prenante extérieure au projet, il n’est pas possible de connaitre la méthode de filtrage employée. Il est à noter que tous les cycles de pesage sont uniques. De plus, il est possible d’observer un « décalage » entre le signal brut et le signal filtré, il s’agit de la phase du filtre.

L’impact des mouvements de la machine sur les poids perçus et enregistrés par les cellules de charge est également très discernable grâce au tracé du poids brut. Dès que le système effectue un mouvement, cela crée des perturbations dans le signal. Lors de

l’ouverture de la trémie d’alimentation, l’impact causé par le produit tombant sur la balance est remarquable par la croissance exponentielle du signal en début de cycle et par les grandes oscillations présentes jusqu’au changement de débit. Le débit rapide utilisé en début de cycle cause également de grandes oscillations du bac. Une valeur de poids supérieure à ce qu’il y a réellement comme produit dans la balance est alors visible. La diminution du débit permet donc d’atténuer ces fortes oscillations et de redescendre vers des valeurs de poids plus réelles. La même réaction est observable à la fin du petit débit, mais dans des proportions plus faibles. Le filtrage permet d’atténuer ces effets, mais ne les supprime pas, car ces réactions sont également observables sur le tracé du poids. L’API prend donc en compte ces éléments et l’étape de stabilisation permet de vérifier que cela est correctement fait. Ainsi, malgré la phase qu’implique le filtrage, il permet de faciliter la prise de décision pour l’API.

En ce qui concerne l’évacuation du produit une fois qu’il a été pesé, cela se fait en ouvrant les portes de décharge, ce qui cause des vibrations assez conséquentes. Elles sont observables par les grandes oscillations présentes à la fin du cycle. Si une tare est programmée, alors une stabilisation du système est réalisée pour éviter de fausser cette étape. Si aucune tare n’est programmée entre deux pesées alors une partie des vibrations causées par la fermeture des portes de décharge sont encore présentes lorsqu’une nouvelle pesée est lancée. La Figure 9 montre ces deux éléments et l’on remarque que le filtre permet tout de même de lisser ces vibrations également. Il est à noter qu’il ne serait pas envisageable de réaliser des tares après chaque pesée, car cela ferait perdre un temps non négligeable lors de l’ensachage. Les tares sont donc réalisées de façon séquentielle.

### 3.1.4 Problèmes rencontrés

Les erreurs de pesage font partie des principaux problèmes rencontrés dans ce secteur. En effet, la précision de la mesure est très importante et peut être grandement affectée par l’environnement industriel dans lequel se trouvent les systèmes de pesage. Cet environnement génère beaucoup de bruit dans les signaux des cellules de charge, ce qui nuit

à la performance et entraine des erreurs de dosage. Les erreurs ont des causes multiples, dont voici les principales.

- Les chocs et les mouvements du bac de pesage qui peuvent être causés par la chute du produit. Le système étant vertical et par alimentation gravimétrique, plus la hauteur de la trémie sera grande, plus l'impact du produit sera conséquent. Ce paramètre est fixe, mais il est lié à la structure du système. Il en est de même pour la forme de la trémie ou la rigidité de la structure par exemple. Cela implique donc qu'il existe autant de profils de signaux de pesage que de modèles du système existants.
- Les mouvements environnants qui créent des vibrations perçues par les cellules de charge vont créer des erreurs de mesure. Il est possible de trouver des solutions pour minimiser leurs impacts sur les mesures, mais cela va dépendre de leur périodicité. S'il s'agit d'un bruit permanent ou d'un bruit ponctuel, chacun a un impact sur la pesée. Cependant, il est important d'identifier la bonne catégorie pour utiliser des mesures correctives adaptées ou au contraire, identifier le fait que c'est ponctuel afin de ne pas rectifier des paramètres en fonction d'un évènement qui ne se reproduira pas lors d'une autre pesée.
- Le type de produit pesé peut agir de façon à fausser des mesures. Il peut y avoir des produits qui ne sont pas homogènes en poids, en forme, en densité ou qui adhèrent aux parois. Par exemple, s'il s'agit d'un produit sous forme de poudre alors il pourrait être plus compact ou plus volatile en fonction du taux d'humidité présent dans l'environnement de pesage.

Ces erreurs de pesage vont amener à produire des sacs pouvant être trop ou pas assez remplis, ce qui peut entrainer différents problèmes sur la suite de la ligne de production. Un sac trop rempli pourrait entrainer des problèmes de qualité, car cela pourrait l'endommager en le déformant ou en le fissurant et provoquer une fuite de produit. Dans les deux cas de défauts de remplissage, cela peut entrainer des coûts de production plus élevés, des problèmes de conformité, des pertes de produits et une insatisfaction du client.

Par ailleurs, le pesage ne doit pas constituer un élément qui ralentit la chaine de production et la vitesse de production ne doit pas nuire à la précision de la mesure. Ainsi il est essentiel de trouver un équilibre entre la vitesse de production et la précision de la mesure pour maximiser l'efficacité de la ligne de production et ne pas affecter l'entreprise dans sa capacité à répondre à la demande.

## 3.2 OBJECTIFS ET CONTRAINTES

### 3.2.1 Problématique et objectifs

La compréhension du fonctionnement du système telle que décrite dans ce chapitre a donc été une partie essentielle de cette problématique à traiter, car elle a permis d'identifier les parties du signal sur lesquelles il sera possible d'agir pour réduire le temps de cycle. Ainsi, l'objectif de ce projet est donc d'améliorer le cycle de pesage en agissant sur la rapidité afin de pouvoir réaliser plus de sacs en un temps donné sans perdre en précision de poids. Pour réaliser cet objectif, les sous-objectifs suivants ont été identifiés :

1. Constitution de la base de données
2. Optimisation du cycle de pesage avec précision

La première composante du signal de pesage sur laquelle agir est la diminution du temps de débit en goutte-à-goutte. La réduction de cette période se fera en augmentant celle de débit rapide qui se produit au début du cycle de pesage. De plus, les autres éléments sur lesquels agir seront notamment les périodes de stabilisation lors de la vérification du poids final avant l'évacuation pour la mise en sac et la vérification du zéro fait par la balance en fin de cycle. La durée de ces deux périodes est dépendante des perturbations environnantes de la balance et l'IA pourrait donc aider à améliorer ces parties du signal. De plus, lors de l'analyse des données, il a été constaté que les durées des phases entre différents cycles sont très variables. Ainsi, une réduction du temps de cycle est possible.

### 3.2.2 Contraintes

L’entreprise a sous-traité la réalisation de l’API. Par conséquent, hormis les données de sortie qu’il est possible de récupérer lors des pesées, notre connaissance sur l’algorithme utilisé par l’API se limite au fait que l’algorithme permet une certaine optimisation entre les cycles de pesage et qu’il se base principalement sur le poids filtré. Le détail de cette optimisation nous est inconnu.

De plus, il sera nécessaire d’obtenir une quantité de données suffisante afin de pouvoir entrainer des modèles d’IA. Cependant il ne serait pas possible de récupérer des données issues de lignes présentes chez les clients de l’entreprise partenaire, car il y aurait trop de variabilités dans les données. En effet, dans les lignes de production, un seul produit avec une seule configuration de paramètres n’est pas dédié en permanence à une seule balance industrielle dans une production. Or, comme cela a été évoqué précédemment, chaque cycle est unique. Ainsi, ajouter une variabilité liée aux paramètres de configuration pourrait nous empêcher d’optimiser le cycle en lui-même. Le but du projet n’est pas de trouver la bonne combinaison de paramètres lors du paramétrage de la balance industrielle. Ainsi, les prises de données seront réalisées avec le même produit et autant que possible sur une balance paramétrée de la même manière. Concernant le produit, il faudra qu’il ne soit pas périssable, qu’il ne s’altère pas à cause de l’humidité et qu’il ait un comportement qui soit constant.

Enfin, une dernière contrainte émise par l’entreprise partenaire serait que les améliorations apportées au système soient réalisées sans modification structurelle du système de pesage-emballage. Cela concerne la mécanique et les capteurs du système. Seul l’API peut être modifié, notamment au niveau logiciel.

## 3.3 MÉTHODOLOGIE

La section méthodologie a pour but d'expliquer la stratégie qui a été mise en place pour résoudre le problème et qui a permis d'obtenir les résultats qui seront décrits dans les prochains chapitres de ce mémoire. La méthode de constitution de la base de données est donc expliquée dans la première partie de cette section. Par la suite, les démarches appliquées pour le filtrage du bruit et l'optimisation du cycle sont détaillées.

### 3.3.1 Constitution de la base de données

- **Enjeux de la prise de données :**

Dans le cadre de ce projet, la constitution de la base de données a été une partie essentielle pour comprendre le fonctionnement du système étudié. L'étude de son fonctionnement a permis d'identifier les éléments du signal sur lesquels agir pour répondre à la problématique. De plus, l'étape de prise en main du système a permis de comprendre les effets des différents paramètres configurables sur le signal de poids. Cela a également contribué à comprendre partiellement la manière dont l'API optimise les cycles de pesées en s'appuyant sur le signal filtré, tout en tenant compte du fait que ce signal présente une phase.

Ainsi, la prise de données s'est faite en plusieurs étapes. D'abord les premières prises de données ont permis de travailler sur les différents éléments de compréhension cités. Puis une étape dédiée à la prise de données en masse afin d'avoir des données pour entrainer les différents modèles. Lors de cette étape, des paramètres fixes ont été appliqués et qui seront détaillés par la suite.

- **Particularités du système de prise de données :**

Les données ont été prises sur un système de test présent dans l'entreprise afin de réaliser des essais pour différents projets. La particularité de ce système est qu'il ne permet pas d'ensacher la matière qui est pesée dans la balance. À la place, un bac de récupération a été positionné en dessous des portes de décharge. Le produit y est donc déversé puis un système de vis sans fin permet de le faire remonter jusque dans le bac de stockage du produit. Le désavantage de ce système est qu'il n'est pas possible de revérifier le poids à la fin du cycle. Il faudra donc considérer le dernier poids transmis par la balance comme étant le poids du sac qui aurait été ensaché. Néanmoins le système a été initialement conçu pour fournir des résultats précis et l'objectif du projet étant de le rendre plus rapide, nous pouvons donc considérer les résultats fournis par la balance comme étant fiables. L'avantage majeur de ce système est qu'il permet de pouvoir tester très facilement différents réglages du système.

L'impact de cette modification sur les tracés des pesées est qu'un temps d'évacuation du produit est nécessaire afin que le bac de récupération ne déborde pas. La Figure 10 ci-dessous montre les graphes d'un cycle de pesage sur un système standard et le graphe d'un cycle sur le système de prise de données.

Comme nous pouvons le voir sur la Figure 10, la phase de stabilisation est prolongée dans le cycle du système avec récupération. Cependant, l'API distingue bien la différence entre ces éléments. Ainsi, la phase de stabilisation dure le temps nécessaire, comme dans un système non modifié pour des essais. La prolongation due à la récupération est placée dans la phase d'évacuation et d'initialisation d'un nouveau cycle. Cette partie du signal n'est donc pas présente dans les cycles de pesage.


Figure 10. Graphes montrant la différence entre des cycles de pesage sur les systèmes standard et de prise de données

Concernant le produit sélectionné et fourni par l'entreprise afin de récolter des données pour le projet, il s'agit de billes de plastiques. Ce matériau a été choisi, car il est durable dans le temps, non volatile et sa densité n'est pas altérée par les conditions environnantes. Cependant, les cycles répétés ont entrainé une dégradation du matériau en usant les billes de plastique à cause du frottement contre les parois de la balance et de la vis sans fin. Cela a donc contribué à ajouter une certaine variabilité tout au long des prises de données, notamment car le système n'est pas uniquement utilisé pour ce projet.

- **Les données récupérées :**

Le système fournit douze colonnes de valeurs. Ces données de sorties transmises par la machine sont :

- Le flux de matière
- La valeur de poids de fin du débit rapide : Il s'agit de la valeur de poids filtré choisie pour le changement de débit dans le cycle en cours. Cette valeur est définie dans les premières phases du cycle en fonction du poids cible et du débit en cours. Elle est variable pour chaque cycle et est remise à zéro à chaque début de nouveau cycle, puis redéfinie en cours.
- La valeur de poids de fin du débit lent : Il s'agit de la valeur de poids filtré pour laquelle la trémie d'alimentation sera fermée afin que plus aucun produit ne s'écoule. Cette valeur est définie durant les phases qui suivent le changement de débit. Elle est également remise à zéro à chaque nouveau cycle.
- Le temps : Cette colonne de valeur donne le temps écoulé depuis le début du lancement des cycles. Il est donné en millisecondes, en sachant que la fréquence d'échantillonnage du système est de 627 Hertz.
- Les étapes du cycle de pesage : Il s'agit de valeurs entières qui indiquent l'action dans laquelle se trouve le système. Durant les cycles de pesage, ces valeurs vont de 1 à 8, mais il existe d'autres valeurs pour des actions utiles au fonctionnement du système. Ces différentes phases seront détaillées par la suite.
- La colonne de données qui suit celle des étapes du cycle de pesage contient des valeurs toujours nulles.
- Activation de l'écoulement : Cette prend la valeur « 1 » lorsque la trémie est ouverte et que l'écoulement de produit dans la balance est activé. S'il n'y a pas de produit qui s'écoule dans le bac de pesage, cette valeur est nulle.
- Le poids filtré : Il s'agit des valeurs de poids filtré par l'API.

- Pourcentage d'ouverture de la trémie d'alimentation.
- Activation du déchargement du bac de pesage : Cette colonne prend la valeur « 1 » lorsque le pesage est terminé et que le produit est prêt à être évacué, sinon la valeur est nulle.
- Le poids brut : Il s'agit des valeurs de poids telles qu'elles sont perçues par les cellules de charge.
- La valeur de la tare : Cette colonne donne la valeur de la tare qui est réalisée lorsque le pesage à vide a été activé. Lorsque le pesage à vide est activé, il est possible de le remarquer sur le signal lorsqu'un temps d'attente est observé avant de commencer un nouveau cycle (voir Figure 8). Cette étape permet de s'assurer que la valeur de poids net est mesurée durant la pesée. Cependant, cette étape ralentit le processus donc elle n'est pas effectuée pour chaque cycle mais de façon périodique, c'est-à-dire une fois toutes les 25 pesées dans notre cas.
- **Détails des différentes phases :**

Concernant la numérotation des différentes actions prises par l'API, le Tableau 7 résume les phases qui se produisent lors des cycles de pesage.

Tableau 7. Description des phases du cycle

| Numéro de la phase pour l'API | Action de cette phase |
|---|---|
| 1 | Début d'une nouvelle pesée avec l'ouverture de la trémie pour un débit rapide |
| 3 | Détermination de la valeur de changement débit |
| 4 | Changement d'ouverture de la trémie pour le passage en petit débit |
| 5 | Écoulement en petit débit |
| 6 | Détermination de la valeur de poids à atteindre pour lancer la fermeture de la trémie |
| 7 | Fermeture de la trémie |
| 8 | Phase de stabilisation afin de vérifier le poids dans la balance |
| 0 | Évacuation du produit et remise à l'état de début de cycle |

Ainsi, lorsque le pesage est lancé, ces différentes phases se succèdent de façon itérative. Il est possible de constater qu'il n'y a pas de phase numérotée « 2 ». En effet, celle-ci existe, mais se produit uniquement lorsqu'il est demandé au système de ne pas surveiller le signal tant qu'une certaine durée n'est pas écoulée. Dans notre cas, nous souhaitons connaitre l'évolution du signal en tout temps et qu'il soit surveillé par l'API, ainsi il est normal que cette phase n'apparaisse pas dans notre signal et que notre deuxième phase corresponde à l'indication « 3 » pour l'API.

- **Les différents paramètres :**

Il est possible de distinguer différents types de paramètres. Parmi tous les paramètres réglables sur le système, nous ne détaillerons que ceux qui nous intéressent, car ils ont un impact sur les cycles de pesage. Ces paramètres sont séparés en deux catégories lors de leur paramétrage sur l'interface personne-machine : les paramètres liés à la commande de l'API, qui concernent ce qui se passe dans un cycle, et ceux liés au système, qui concernent les paramètres des lots de cycles.

Les paramètres qui concernent le programme de pesage sont les paramètres suivants :

- Le poids nominal
- Le pourcentage de sur-remplissage qui permet d'ajouter entre 0% et 2% du poids nominal à chaque pesée afin de limiter les sacs non conformes en étant certain qu'aucun sac n'ait un poids inférieur au poids nominal.
- La limite basse et la limite haute de poids admissible dans les sacs.
- Le pourcentage d'ouverture de la trémie d'alimentation pour le débit rapide.
- Le pourcentage d'ouverture de la trémie pour le débit lent.

- Le type d'algorithme, qui peut être standard ou intelligent. Dans le cas du programme standard, l'API ajuste les vitesses de remplissage en fonction de poids intermédiaires qu'il doit atteindre, aussi appelés points de consignes. Pour l'algorithme dit « intelligent », il surveille le flux et adapte les points de consignes en fonction de celui-ci. Pour cet algorithme, il est possible de décider si sa priorité doit être d'obtenir des pesées très précises ou très rapides ou un équilibre entre les deux.

D'autres paramètres comme l'activation de la vibration des portes de décharge qui est utilisée en cas de produits agglomérants existent, mais ils ne sont pas utiles à ce projet donc ils ne seront pas décrits dans ce mémoire.

Les paramètres du système sont :

- La taille des lots (*sample frequency*) est le nombre de cycles de pesage dans un lot. Les lots sont principalement utiles pour indiquer à quelle fréquence faire la tare.
- La fréquence de la tare (*zero frequency*) indique après combien de lots doit être réalisée la tare.
- Le nombre de pesées vérifiées (*sample size*) correspond au nombre de cycles de pesage par lot où le cycle comprend la phase de stabilisation afin de vérifier le poids dans la balance. Si cette valeur vaut « 1 » alors la vérification du poids se fait à chaque cycle. Si la valeur est différente de « 1 » et vaut par exemple « 4 » alors seule quatre pesées seront vérifiées dans chaque lot. La taille du lot devant forcément être supérieure à « 4 ».

Lorsque le programme est lancé, l'interface personne-machine affiche différentes données et statistiques sur la production en cours (Figure 11).


Figure 11. Photographie de l'interface durant les cycles

Comme le montre la capture d'écran (voir Figure 11), le poids capté par la balance à chaque instant est affiché. Le poids cible est rappelé et la moyenne des poids des sacs réalisés ainsi que l'écart-type sont également donnés avec le poids du dernier sac fait. Cela indique également le nombre de cycles acceptés et des informations sur les cycles comme le nombre d'unités réalisées par minute et les durées de débit rapide, lent et du cycle entier.

- **Protocole de la prise de données de masse :**

L'ensemble de la récolte des données se fait sur le même système et avec le même matériau. La plupart des paramètres sont fixes entre chaque récolte de données. Le choix de garder une majorité de paramètres fixes est de limiter les facteurs afin de ne pas augmenter la variabilité du signal déjà présente dans les données. De plus, dans une utilisation industrielle, lors de l'ensachage d'un produit, une fois le processus lancé, les paramètres sont souvent fixes pendant un très grand nombre de pesées étant donné que les machines fonctionnent autant que possible en continu.

Les paramètres et leurs valeurs sont regroupés dans le Tableau 8 :

Tableau 8. Valeurs des paramètres de l'API

| **Paramètres** | **Valeurs** |
|---|---|
| Poids nominal | 25,0 kg |
| Pourcentage de sur-remplissage | 0% |
| Limite basse de poids admissible | 24 |
| Limite haute de poids admissible | 26 |
| Pourcentage d'ouverture de la trémie pour le débit rapide | 65% |
| Pourcentage d'ouverture de la trémie pour le débit lent | 25% |
| Type d'algorithme | Intelligent et rapide |
| Taille du lot | 1 |
| Fréquence de la tare | 25 |
| Nombre de pesées vérifiées par lot | 1 |

Concernant le paramètre indiquant le nombre de cycles de pesage où le poids est vérifié, dans notre cas, nous l'activons pour chaque cycle. Toutefois, dans l'industrie, cette vérification peut être moins fréquente afin de maximiser le nombre de sacs produits par minute. Il s'agit de notre seule méthode pour vérifier que la balance a correctement effectué son cycle, étant donné que dans notre modèle du système n'inclut pas la mise en sac.

Dans l'industrie, la tare n'est pas effectuée après chaque sac pour optimiser la production en un temps donné. Pour la majorité de nos prises de données, nous avons appliqué ce même principe en effectuant la tare toutes les 25 pesées.

Un total de 10 041 cycles de pesage a été enregistré lors des différentes sessions de prises de données. Comme cela a pu être évoqué précédemment, chaque cycle de pesage est unique. Cette variabilité est causée par différents facteurs et notamment le bruit environnant qui crée des vibrations captées par les cellules de charge.

Une information intéressante à observer est la durée des différentes phases de chaque cycle. Le Tableau 9 regroupe quelques statistiques concernant la durée de ces phases.

Tableau 9. Statistiques sur la durée des phases des cycles de pesage

| **Phase** | **1** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|
| Durée minimum (millisecondes) | 655 | 24 | 369 | 237 | 21 | 335 | 905 |
| Durée maximum (millisecondes) | 780 | 593 | 524 | 526 | 1666 | 555 | 11730 |
| Médiane (millisecondes) | 686 | 243 | 379 | 374 | 144 | 417 | 1263 |
| Moyenne (millisecondes) | 684,51 | 245,38 | 381,55 | 378,35 | 193,41 | 413,75 | 1319,38 |
| Ecart-type (millisecondes) | 10,72 | 55,42 | 11,93 | 61,52 | 133,33 | 24,12 | 371,10 |

Les valeurs présentées dans le Tableau 9 sont en millisecondes. La numérotation des phases correspond à celle donnée par l'API, c'est pour cette raison qu'il n'y a pas de phase portant le chiffre « 2 » dans le tableau.

Ainsi, il est possible d'observer une variabilité dans les durées de phases. Cette variabilité permet de confirmer qu'il est possible de diminuer la durée de chaque cycle, tout en gardant la même précision de pesage. Pour réaliser cette optimisation, il faut donc faire en sorte de faire durer le plus longtemps possible, les phases 1 et 3. Elles correspondent au débit rapide, il sera donc avantageux de le maintenir le plus longtemps possible afin d'améliorer le remplissage, ce qui aura pour impact de réduire le petit débit. Concernant le petit débit, qui correspond aux phases 5 et 6, il ne doit pas être remplacé entièrement par du débit rapide afin de garder une précision permettant d'atteindre le poids cible. Cependant, réduire sa durée permettra de réaliser plus de cycles en un temps donné.

- **Prétraitement :**

Les fichiers de données se présentent sous forme de fichiers textes dans lesquelles différentes informations sur l'exécution des pesées sont données. Puis une ligne est dédiée à

chaque pas de temps des cycles. Dans chaque ligne, les différentes informations (flux, poids de consignes, temps …) sont séparées par des points-virgules. Ainsi la première étape consiste à enlever les informations inutiles et à identifier chaque colonne de données.

Avant d'utiliser les données récoltées dans un programme, les trois premières pesées et la dernière sont retirées du fichier de données. Les trois premières sont retirées, car elles servent à l'API pour se paramétrer donc ces premières pesées sont composées d'autres phases que celles standards. La dernière est supprimée, car elle peut ne pas être entière, cela peut être dû au moment d'arrêt de l'enregistrement lors de la prise de données. Puis les différents fichiers de données sont concaténés en ne gardant que les colonnes utiles pour le programme dans lequel ils vont être utilisés afin de limiter la taille du fichier. Toutes ces étapes sont réalisées à l'aide d'un programme Python.

### 3.3.2 Optimisation en une étape

Initialement, l'entreprise partenaire souhaitait que l'optimisation soit faite en une étape, c'est-à-dire réaliser l'optimisation en se basant uniquement sur le poids brut. Ce souhait était lié au fait qu'ils avaient constaté que le filtrage crée un retard. Ainsi, le but de cette approche est de réaliser des prédictions puis d'appliquer des mesures préventives ou correctives en fonction des prédictions pour la suite du cycle ou des pesées à venir. Cela revient donc à extrapoler puisqu'il s'agit de prédire des valeurs en dehors des données existantes. Pour cela, les objectifs suivants ont été définis afin de déterminer les éléments sur lesquels agir :

- Prédire le poids final rapidement,
- Réduire la durée du cycle de pesage, en maximisant la période de grand débit par rapport à celle de petit débit.

Ces deux objectifs doivent être intégrés dans un modèle global, qui doit ensuite être généralisable à plusieurs poids cibles et types de produits pesés. Ainsi, dans le quatrième

chapitre, nous détaillerons les résultats et les réflexions qui nous ont amenés à appliquer ces méthodes.

### 3.3.3 Optimisation en deux étapes

Suite aux résultats obtenus en mettant en place l'approche en une étape, nous avons été amenés à reconsidérer la méthodologie appliquée. Ainsi cette nouvelle approche repensée, comporte deux étapes : une étape de filtrage des données brutes et une étape d'optimisation des décisions prises par l'API. L'objectif est donc de fournir un modèle global qui puisse être généralisable. Le modèle obtenu devra se suffire à lui-même pour réaliser une optimisation complète du cycle en sachant quelles actions prendre en fonction des cycles observés précédemment et du déroulement du cycle en cours.

En ce qui concerne le filtrage des bruits, l'objectif est de trouver un moyen de filtrer les bruits présents dans les données. Il doit donc être efficace pour lisser les données sans perdre d'informations importantes et être utilisable en temps réel. En effet, il doit nécessiter un temps de traitement aussi faible que possible. Pour répondre à cet objectif, différentes méthodes de filtrage vont être testées. Il s'agira de filtres numériques, des modèles d'apprentissage automatique et des modèles d'apprentissage profond. Ces méthodes seront comparées afin de sélectionner la meilleure pour notre utilisation.

Après les premiers résultats obtenus pour l'approche en une étape, nous avons cherché la façon dont les différents modèles devraient être combinés afin d'indiquer à l'API les meilleures décisions à prendre. Après quelques recherches de l'existant, nous avons envisagé d'utiliser l'apprentissage par renforcement pour répondre à cet objectif. En effet, l'apprentissage par renforcement permet de définir la meilleure action à réaliser en fonction d'un but recherché. Ce type d'algorithmes a notamment déjà été appliqué dans de nombreuses utilisations industrielles, comme nous l'avons déjà mentionné dans le chapitre précédent.

Pour intégrer l'apprentissage par renforcement à notre problème, nous avons commencé par simplifier au maximum le problème. Cela nous a permis de sélectionner des modèles adaptés et de définir les bonnes stratégies ou formules de récompenses à employer. Nous avons ensuite rajouté des difficultés pour nous rapprocher de plus en plus de notre problème.

L'ensemble de ce travail est développé dans le cinquième chapitre de ce mémoire.

## 3.4 SYNTHÈSE DU CHAPITRE

Ce chapitre a permis d'expliquer plus en détail le contexte du problème en détaillant le fonctionnement du système de pesage. Il a également permis de présenter en détail les données du système, dont l'analyse à constituer une part non négligeable du travail réalisé. Cette analyse a contribué à définir les zones du signal qui peuvent être optimisées. La Figure 12 reprend les éléments présentés dans l'introduction du mémoire, complétés avec les informations détaillées précédemment. Il a pour objectif de résumer la méthodologie adoptée pour résoudre le problème. Les éléments concernant la récolte des données et la compréhension du système ont été détaillées précédemment. Les deux chapitres suivants vont donc permettre de détailler les différentes approches employées, ainsi que les résultats obtenus lors de l'application de ces approches.

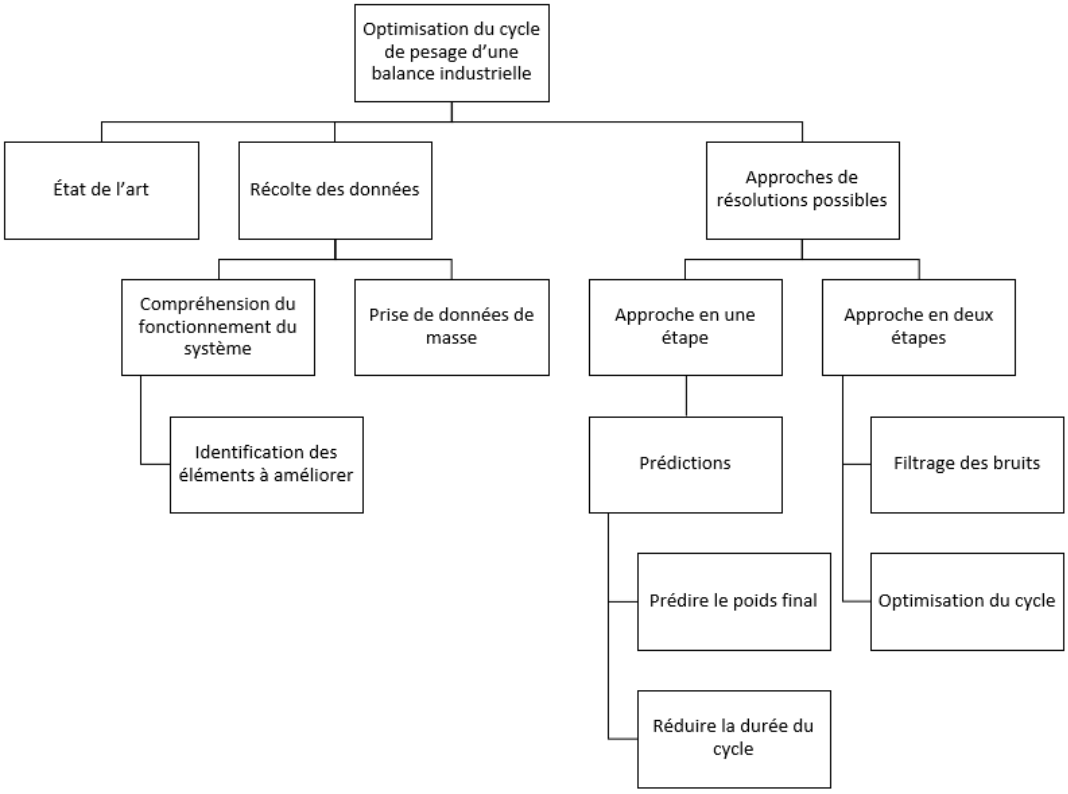



Figure 12. Graphique résumant la méthodologie générale

# CHAPITRE 4
# OPTIMISATION EN UNE ÉTAPE

L'approche développée pour résoudre le problème étudié a été amenée à évoluer durant le projet. Ce changement est notamment dû à la meilleure compréhension du système suite aux différentes prises de données réalisées et analyses de celles-ci, mais également après l'obtention des premiers résultats. Les différents résultats obtenus tout au long du projet nous ont amené à repenser notre approche et à la faire évoluer. Ce chapitre présente le cheminement réalisé pour la première approche développée et qui correspondait aux premières attentes de l'entreprise partenaire. Cette approche est donc détaillée dans un premier temps. Puis les résultats obtenus seront présentés et les justifications ayant amenées à l'évolution de l'approche également. Pour finir une discussion autour de ces différents éléments est donnée.

## 4.1 L'APPROCHE

L'objectif de cette approche est donc de prédire la continuité des cycles de pesage en cours et à venir, puis d'ajuster les décisions de l'API en se basant sur les résultats des prédictions réalisées. Pour cela, un modèle intégrant ce fonctionnement a été pensé. Ce modèle se décompose en quatre sous-modèles suivants (voir Figure 13) :

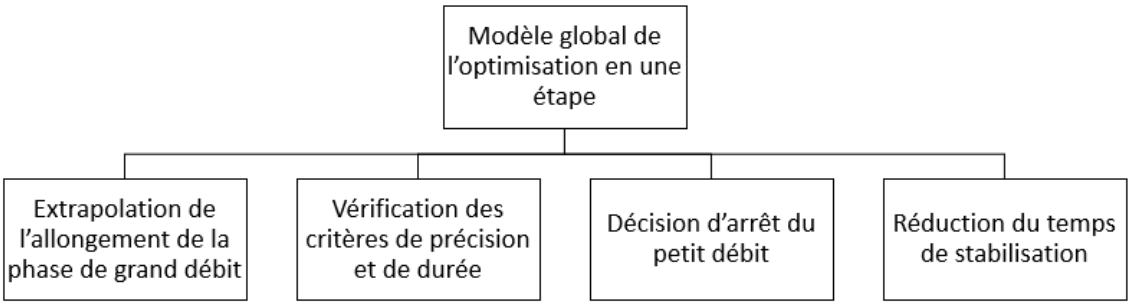



Figure 13. Résumé de la première approche proposée

- Premier sous-modèle : Il a pour but de prédire et d'ainsi extrapoler un allongement de la phase de grand débit afin de réduire la durée du cycle.

- Deuxième sous-modèle : Les résultats de cette prédiction seront utilisés afin de vérifier que l'on répond aux critères de précision et de durée. Pour cela, les prédictions du temps restant du cycle et du poids final seront réalisées.

- Troisième sous-modèle : Ce sous-modèle a pour objectif de décider quand arrêter le petit débit afin de procéder à l'arrêt de l'écoulement du produit. Cette décision serait prise pour respecter le poids cible.

- Quatrième sous-modèle : Pour terminer, la réduction du temps de stabilisation sera réalisée.

Ainsi, ce modèle général permettrait de réduire la durée de cycle de pesage en augmentant par extrapolation celle de la phase de grand débit puis en raccourcissant le temps nécessaire à la stabilisation. Pour la prédiction, des modèles d'apprentissage supervisé et des méthodes basées sur les statistiques sont employées. Les méthodes statistiques utilisées pour faire de la prédiction sont le modèle ARIMA, pour moyenne mobile autorégressive intégrée, et ses variantes. Concernant les modèles d'apprentissage profond, les modèles de réseaux récurrents à mémoire courte et long terme et les réseaux de convolution, soit respectivement

LSTM et CNN, sont testés. Le recours à des modèles statistiques a pour but de permettre de justifier, ou non, l'utilisation de l'IA. Les premiers résultats obtenus avec cette approche nous ont amené à envisager une approche différente. Ils seront décrits dans la section dédiée aux résultats.

Ces modèles comportent différents hyperparamètres, dont plusieurs valeurs ont été testées lors de la mise en application pour chercher les configurations apportant des résultats intéressants.

Les hyperparamètres testés pour les modèles ARIMA sont les suivants :

- Paramètre *p* (ordre d'autocorrélation) : 0 ;1 ; 2 ; 3 ; 4 ; 5

- Paramètre *d* (ordre de différenciation) : 0 ; 1 ; 2

- Paramètre *q* (ordre de moyenne mobile) : 0 ;1 ; 2 ; 3 ; 4 ; 5

Les valeurs de paramètres testées ne sont pas plus grandes car cela amènerait à un modèle complexe et très ajusté aux données d'entrainements, ce qui le rendrait donc moins généralisable à d'autres données. En effet, des valeurs élevées pour ces paramètres peuvent entrainer un surajustement, où le modèle capture le bruit plutôt que le signal réel. Les valeurs du paramètre de différenciation sont plus limitées que les autres car une différenciation supérieure peut rendre le modèle instable et difficile à interpréter. Un équilibre doit donc être trouvé entre la complexité du modèle et sa capacité à se généraliser à de nouvelles données.

De plus, trois tailles de base de données d'entrainement sont testées pour vérifier les capacités des modèles sur différentes quantités de données. Le modèle peut avoir des performances variées selon la taille de la base de données, et il est intéressant d'estimer une quantité minimale de données nécessaires pour faire des prédictions fiables.

Les hyperparamètres communs aux différents modèles d'apprentissage supervisé :

- Taux d'apprentissage : Un taux d'apprentissage faible permet d'obtenir une convergence plus stable mais plus lente, nécessitant un plus grand nombre d'époques

d’entrainements. Une valeur plus élevée amène à une convergence plus rapide, mais peut créer une instabilité dans l’entrainement, notamment avec des données bruitées. Ainsi, une valeur intermédiaire est également testée pour avoir un compromis entre la vitesse et la stabilité.

- Taille de lot : Afin d’accélérer l’entrainement, une taille de lot suffisamment grande a été utilisée. Les modèles présentés qui vont être mis en place n’utilisent pas tous la même quantité de données, ainsi nous testé les valeurs 128 et 256, qui sont des valeurs couramment utilisées pour ce paramètre.

- Nombre d’époques : Une époque correspond à un passage complet sur l’ensemble du jeu de données pendant l’entrainement du modèle. Pour commencer, un faible nombre d’époques est testé pour obtenir des résultats rapidement. Ensuite, un nombre plus grand et adapté aux modèles plus complexes a été appliqué.

- Taille de la fenêtre de données : La valeur de ce paramètre dépend de l’application et des caractéristiques que nous souhaitons capturer. Une petite fenêtre de données permet de capturer des variations rapides du signal et des motifs présents à court terme. À l’inverse, une grande taille de fenêtre est dédiée pour capturer les tendances du signal. Pour l’application de prolongement des données, une grande fenêtre de données est privilégiée car le réseau doit comprendre l’évolution du signal. Dans le cas de vérifications des critères, une plus petite fenêtre de données peut être suffisante pour observer les changements fréquents et les caractéristiques concernées. Cela dépend donc du type de prévisions souhaitées, à long ou court terme. Enfin, les modèles LSTM, capables de capturer des dépendances à long terme, peuvent bénéficier une fenêtre plus longue pour obtenir des résultats intéressants.

Les hyperparamètres propres aux modèles :

- Nombre de nœuds LSTM : Un modèle avec un faible nombre de nœuds, c’est-à-dire 16 ou 32 nœuds, est dédié à des modèles simples ayant des motifs et des données peu complexes. Un nombre de nœuds plus élevé (64, 128, 256) est plus adapté pour des données

complexes. Ainsi, la valeur choisie est un compromis entre modèles simples et complexes, souvent employé comme point de départ pour tester les modèles.

- Taille du noyau dans un réseau CNN : Une petite valeur a pour rôle de capturer des motifs locaux. Plus la valeur est grande, plus le modèle peut capturer des motifs plus larges en moins de couches. Ainsi, la valeur sélectionnée dépend des dépendances à court, moyen et long terme présents dans les données. Nous avons donc choisi trois valeurs afin de comparer les résultats obtenus lors du prolongement des données de la pesée.

- Nombre de filtres d'un réseau CNN : Tout comme les autres paramètres, les valeurs choisies constituent un compromis entre un modèle capturant des caractéristiques basiques ou plus complexes. Néanmoins, les modèles plus complexes ont un fort impact sur le temps de calcul et ont des besoins en mémoire plus importants.

- Taille de mise en commun d'un réseau CNN : La valeur « 2 » est standard pour des données d'une dimension, car elle réduit la dimensionnalité tout en conservant l'information essentielle. Plus cette valeur est élevée, plus la réduction sera importante, ce qui peut entrainer une perte d'informations. Ainsi, cette valeur doit être adaptée en fonction de la longueur des séquences de données fournies au réseau.

## 4.2 LES RÉSULTATS

Dans cette section, nous présentons les premiers résultats obtenus en appliquant les méthodes statistiques présentées dans les premiers chapitres. Leurs applications ont été réalisées sur des étapes spécifiques du signal pour nous permettre de réaliser une première évaluation de cette technique. Ensuite, nous décrivons la mise en place de l'approche présentée et analysons les résultats obtenus. Les codes de ces méthodes statistiques sont présents à l'Annexe A.

### 4.2.1 Méthodes statistiques : prédictions des valeurs suivantes dans la phase de stabilisation

La première étape consiste à identifier la phase de stabilisation dans chaque cycle de pesage. Ensuite, certains cycles de pesage sont choisis aléatoirement pour permettre d'appliquer la boucle de paramètres et ainsi identifier la meilleure configuration. Une fois les meilleures valeurs de paramètres trouvées, le modèle est appliqué à de nouvelles pesées de la base de données.

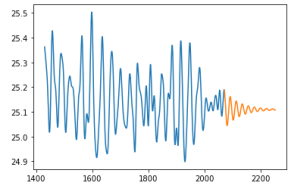
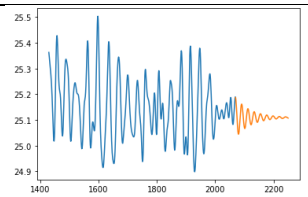
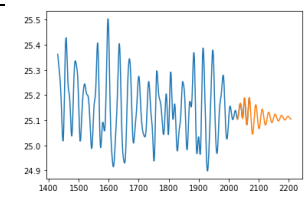
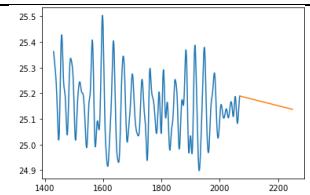
Les modèles ARIMA et ARIMAX ont été testées. Le modèle ARIMA permet de modéliser des séries temporelles en se basant sur des composantes de moyenne mobile et autorégressives. La différenciation permet de rendre les données stationnaires. Le modèle ARIMAX tient compte de variables exogènes afin de pouvoir prendre en compte des facteurs externes qui pourraient affecter la tendance des données. Les valeurs exogènes qui sont prises en compte dans notre cas sont : le point de consigne correspondant au poids de changement de débit, celui de fermeture de la trémie, et la précision du poids final.

Concernant les valeurs des hyperparamètres, nous pouvons observer dans le Tableau 10 que les meilleurs paramètres ne correspondent pas aux valeurs maximales proposées. Ainsi cela montre que pour nos données, il n'y a pas besoin de modèles plus complexes. La valeur RMSE, qui correspond à la racine de l'erreur quadratique moyenne, permet d'évaluer la précision des valeurs prédites par rapport à celles réelles.

La recherche des meilleurs paramètres est réalisée sur les mêmes données pour chaque modèle. Par conséquent, le fait que les valeurs du modèle ARIMA soient identiques aux valeurs des modèles ARIMAX pour les poids d'arrêt est dû au fait que ces valeurs exogènes sont fixes dans une même pesée. Une fois que les valeurs des points de consigne ont été fixées, elles ne changent plus. C'est donc pour cette raison que l'utilisation de ces données en variables exogènes n'a pas eu d'impact sur la prédiction dans ces cas-là. Concernant le résultat obtenu en tenant compte de la précision du poids final, la valeur de

RMSE est plutôt faible même si le modèle d'ajustement est très simple. En effet, seule une autorégression est appliquée sur les données.

Tableau 10. Résultats des modèles ARIMA et ARIMAX pour la prédiction des valeurs suivantes de stabilisation (les signaux prédits en orange)

| **Modèle** | | **Résultats sur une pesée** | | | **Statistiques sur toutes les pesées** | |
|---|---|---|---|---|---|---|
| | | **Meilleurs paramètres trouvés** | **RMSE** | **Tracé des valeurs prédites** | **Moyenne des RMSE** | **Écart-types** |
| ARIMA | | (2,1,0) | 0,107 |  | 0,143 | 0,035 |
| ARIMAX | Poids de changement de débit | (2,1,0) | 0,107 |  | 0,143 | 0,035 |
| | Poids de fermeture de la trémie | (2,1,0) | 0,107 |  | 0,143 | 0,035 |
| | Précision du poids final | (1,0,0) | 0,048 |  | 0,100 | 0,063 |

Ainsi, dans le cas de l'utilisation du modèle ARIMA, les prédictions obtenues oscillent autour de la valeur moyenne des données et convergent vers celle-ci. Quant aux résultats de prédiction pour le modèle ARIMAX qui prend en compte la précision du poids final, le modèle ne permet pas non plus d'obtenir de bons résultats. Des modèles plus complexes seraient nécessaires. Étant donné la grande variabilité des données et le temps de calcul conséquent que ces modèles requièrent, il ne serait pas envisageable de les utiliser en temps réel pour prédire la suite d'une phase de stabilisation, car cela nécessiterait un recalcul régulier du modèle. La valeur moyenne pourrait fournir une valeur fiable sur laquelle s'appuyer.

### 4.2.2 Méthodes statistiques : prédictions des poids finaux des pesées suivantes

Étant donné qu'il n'est pas envisageable d'utiliser de modèle ARIMA pour réaliser une prédiction en temps réel des données suivantes de la pesée en cours, une autre alternative a été envisagée. La méthode ARIMA a été utilisée afin de prédire les poids finaux des pesées suivantes, ce qui permet de réaliser les calculs lors de l'exécution d'un nouveau cycle, en tenant compte de la valeur du cycle précédent pour déterminer celles des suivants. Les mêmes valeurs exogènes que précédemment ont également été prises en compte car, cette fois-ci, chaque valeur représente un cycle de pesage, ce qui entraine une évolution dans les données exogènes. En effet, le point de consigne de fermeture de la trémie d'écoulement a un impact sur la valeur du poids fial de la même pesée. Il en est de même pour les autres valeurs exogènes prises en compte.

Les données utilisées pour réaliser cet objectif sont les derniers poids bruts enregistrés par la balance à la fin de chaque pesée avant d'évacuer le produit qui est ensaché, c'est-à-dire à la fin de la phase de stabilisation qui permet de vérifier le poids du sac. Nous isolons donc ces derniers poids filtrés de plusieurs pesées consécutives, ce qui permet de constituer

une nouvelle série temporelle où prédire la valeur consécutive reviendra à prédire le poids final du cycle de pesée suivant.

La Figure 14 regroupe ces valeurs de poids finaux pour une prise de données de 77 cycles de pesage consécutifs.

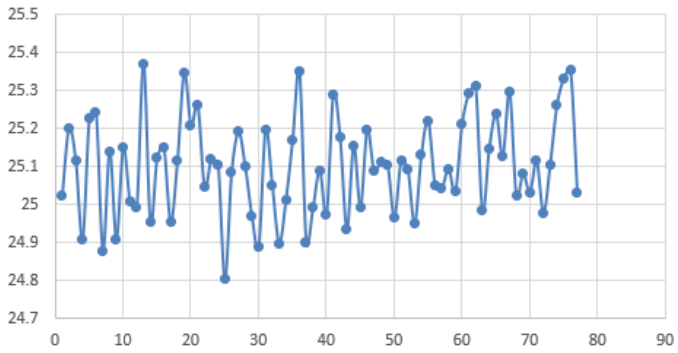



Figure 14. Tracé des derniers poids enregistrés pour 77 cycles consécutifs

Les valeurs oscillent entre 24,806 et 25.372. La moyenne de ces valeurs est de 25,103 pour un écart-type de 0,1298.

Une grille de paramètres a été utilisée pour trouver la meilleure combinaison de paramètres pour les données concernées. Pour rappel, la combinaison de paramètres correspond aux paramètres suivants : nombre de termes autorégressifs, nombre de différenciations appliquées, nombre de moyennes mobiles. Étant donné qu'il s'agit de prédire assez précisément les valeurs successives de cette série de données, des plus grands paramètres ont été testés pour l'autorégression et la moyenne mobile. Ces valeurs vont donc de 0 à 10. Le nombre de données dans la base de données d'entrainement a également varié afin de pouvoir jauger la quantité nécessaire pour obtenir des résultats satisfaisants. Cela correspond à 25%, 50% et 75% du fichier de données. Le Tableau 11 regroupe la meilleure configuration de paramètres trouvées sur les différentes tailles de bases de données d'entrainement et les résultats obtenus.

Tableau 11. Résultats des modèles ARIMA et ARIMAX pour la prédiction des poids finaux

| **Modèle** | | **Nombre de pesées d'entrainement** | **Paramètres** | **RMSE** |
|---|---|---|---|---|
| Modèle ARIMA | | 19 | (1,0,2) | 0,125 |
| | | 38 | (4,2,2) | 0,113 |
| | | 57 | (8,2,0) | 0,119 |
| Modèle ARIMAX | Valeur du poids d'arrêt de l'écoulement en grand débit | 19 | (2,1,0) | 0,124 |
| | | 38 | (4,1,1) | 0,105 |
| | | 57 | (10,1,0) | 0,111 |
| | Valeur du poids d'arrêt de l'écoulement en petit débit | 19 | (4,0,2) | 0,127 |
| | | 38 | (5,0,2) | 0,109 |
| | | 57 | (7,2,0) | 0,130 |
| | Précision du poids final | 19 | (5,1,1) | 0,133 |
| | | 38 | (0,2,1) | 0,188 |
| | | 57 | (4,1,0) | 0,140 |

D'après le Tableau 11, les valeurs de RMSE observées pour les différents modèles identifiés ne montrent pas de variations importantes entre les tailles de base de données d'entraînement utilisées. Cette conclusion se base directement sur l'examen des résultats obtenus, sans procéder à une évaluation statistique formelle pour déterminer la signification ou la systématicité de ces différences. La prise en compte de variables exogènes n'est que faiblement meilleure lorsque nous utilisons le poids de changement de débit. Pour les autres variables exogènes, les résultats ne sont pas meilleurs. De plus, nous pouvons observer que les valeurs RMSE des valeurs prédites sont du même ordre de grandeur que l'écart-type. Comme évoqué dans le chapitre précédent, les données sont assez variables, cela explique donc que les valeurs de prédictions obtenues ne soient pas meilleures. Ainsi, même en

utilisant un ensemble de données plus grand ou des paramètres différents, nous ne pourrions pas obtenir de meilleurs résultats et cela nécessiterait un temps d'entrainement plus long. Pour notre utilisation, il est préférable d'utiliser des petits modèles qui demandent un temps d'entrainement plus faible afin de pouvoir les entrainer régulièrement sur les nouvelles données transmises par la balance.

Ces premiers résultats fournissent un aperçu de l'efficacité de ces méthodes appliquées à titre de comparaison. Leur mise en place a permis d'améliorer notre compréhension des données et des enjeux liés à notre problématique, en pointant les défis de généralisation du problème. Dans la suite, nous présentons les résultats obtenus en mettant en place les premiers modèles utilisant de l'IA.

### 4.2.3 Modèle 1 : Allongement de la phase de grand débit

L'objectif de ce modèle est de prédire l'allongement de la phase de grand débit. Il s'agit donc d'extrapoler les données afin de prédire les prochaines valeurs si on allongeait la phase de grand débit (voir Annexe B). Le modèle a été entrainé sur les données d'entrainements en utilisant une fenêtre glissante de données successives en entrées et la valeur au temps suivant de cette fenêtre de données en sortie. Les valeurs d'entrées et de sorties ne sont prises que dans les données de poids bruts. La Figure 15 permet de représenter ces éléments, avec le signal orange étant le signal de poids brut et le signal bleu représente les phases du cycle.

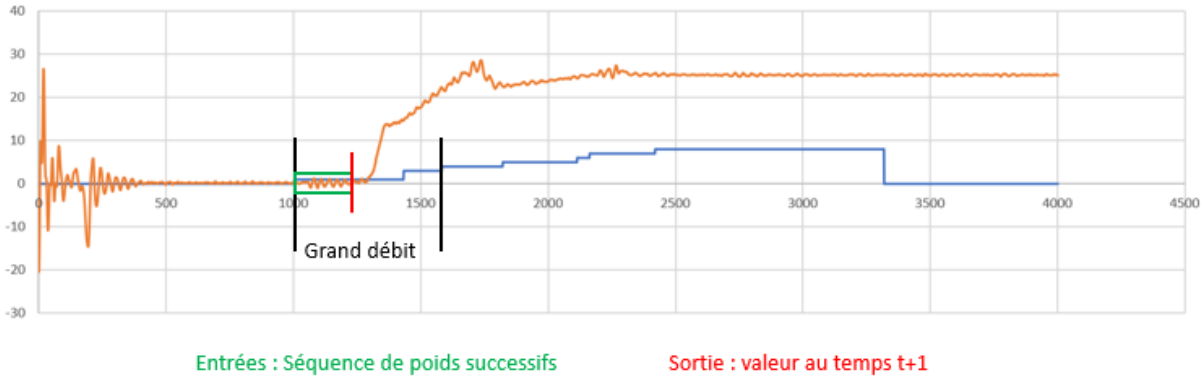



Figure 15. Graphe résumant le fonctionnement du modèle étudié

Ainsi, les entrées sont une séquence de données, représentée par le contour vert sur la Figure 15, et la sortie est composée d'une seule valeur qui est la valeur au pas de temps suivant, de la séquence de données d'entrées. Ces couples entrées-sortie sont formés de façon glissante en décalant à chaque fois d'un pas de temps et sur toute la durée de la phase de grand débit, puis cela est réalisé sur tous les cycles d'entrainement. Le modèle a été testé sur des fenêtres de données de 60 et de 100 valeurs. Les meilleurs résultats pour le modèle LSTM ont été obtenus avec la fenêtre de 100 données. Ainsi lors des prédictions permettant de rallonger la phase de grand débit, les 100 dernières données de cette phase sont fournies en entrée afin de prédire la valeur suivante. Cette nouvelle valeur est ensuite utilisée dans la nouvelle fenêtre glissante de 100 données pour prédire la valeur suivante. Cette méthode permet de prédire autant de données que souhaité.

Les résultats obtenus pour prédire 100 données suivantes et ainsi prolonger la phase de grand débit, sont présentés dans la Figure 16.

Tableau 12. Configuration des algorithmes du modèle 1

| Modèle | Hyperparamètres | Valeurs testées | Meilleure configuration |
|---|---|---|---|
| LSTM | Taux d'apprentissage | 0,0005 ; 0,001 ; 0,01 | 0,001 |
| | Taille du lot | 256 | 256 |
| | Nœuds LSTM | 64 | 64 |
| | Fenêtre de données | 60 ; 100 | 100 |
| CNN | Taux d'apprentissage | 0,0005 ; 0,001 ; 0,01 | 0,001 |
| | Taille du lot | 256 | 256 |
| | Nombre de filtres | 64 ;128 | 128 |
| | Taille du noyau | 3 ;5 ;7 | 5 |
| | Taille de mise en commun | 2 ;3 ;4 | 2 |
| | Fenêtre de données | 60 ;100 | 60 |

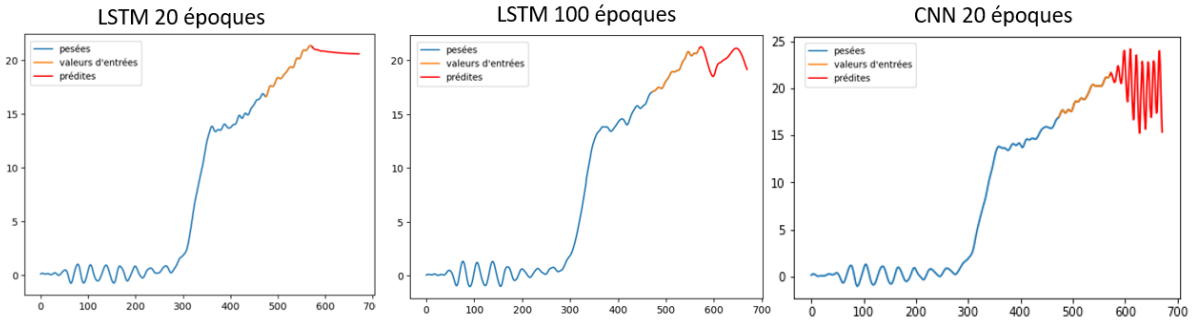



Figure 16. Tracés des valeurs obtenues pour un prolongement de 100 données

La Figure 16 regroupe les résultats pour trois entrainements différents. La partie bleue des données correspond aux poids bruts enregistrés pour la phase de grand débit pour la pesée

testée. En rouge ce sont les 100 données prédites pour le prolongement de la phase. Les valeurs d'entrées en orange représentent la première fenêtre de données, les fenêtres suivantes sont toujours obtenues en réalisant un glissement qui intègre la nouvelle valeur prédite. Ainsi, quand pour le modèle LSTM avec seulement 20 époques, ce qui correspond au nombre d'itérations d'entrainement, le modèle prédit une stabilisation. Dans le modèle avec 100 époques, il comprend que le signal oscille. Quant au modèle CNN il donne des résultats assez incohérents, peut-être qu'il a compris le bruit.

Afin de mieux comprendre les résultats du modèle, nous avons réalisé des prédictions similaires tout au long de la phase. Ces prédictions avaient pour but de voir si un lissage était également effectué ou si le modèle suivait la tendance des données et ainsi mieux comprendre les résultats obtenus pour les prédictions de prolongement de la phase.

Les résultats obtenus sont présentés à la Figure 17 et la Figure 18.

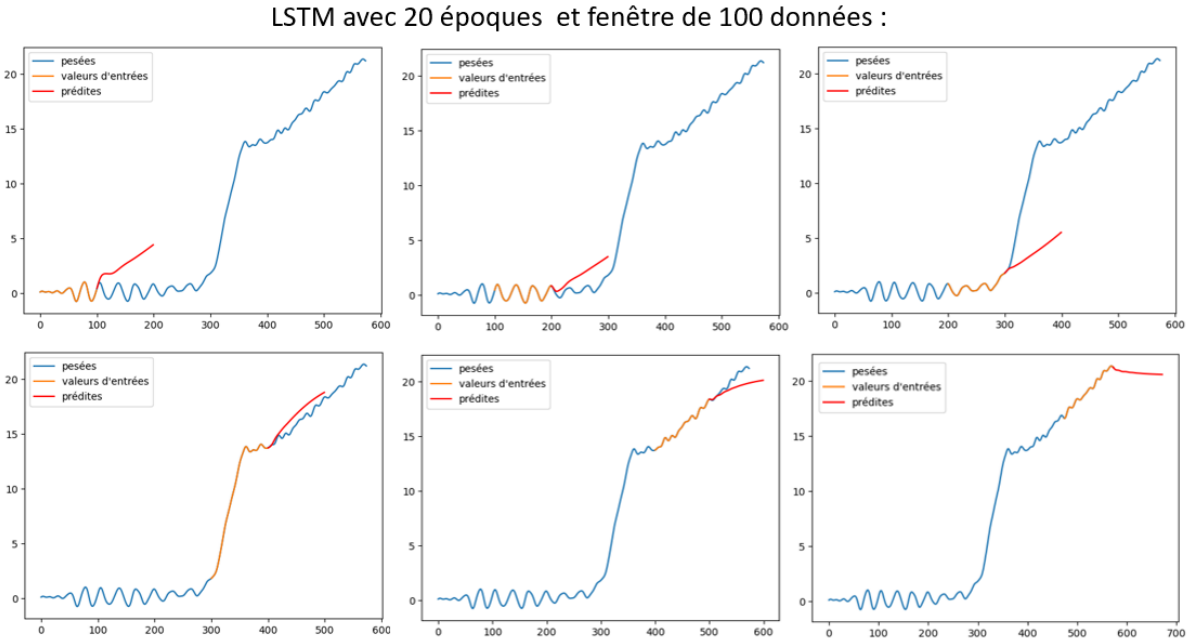



Figure 17. Tracés des prédictions faites à l'intérieur des données pour 20 époques

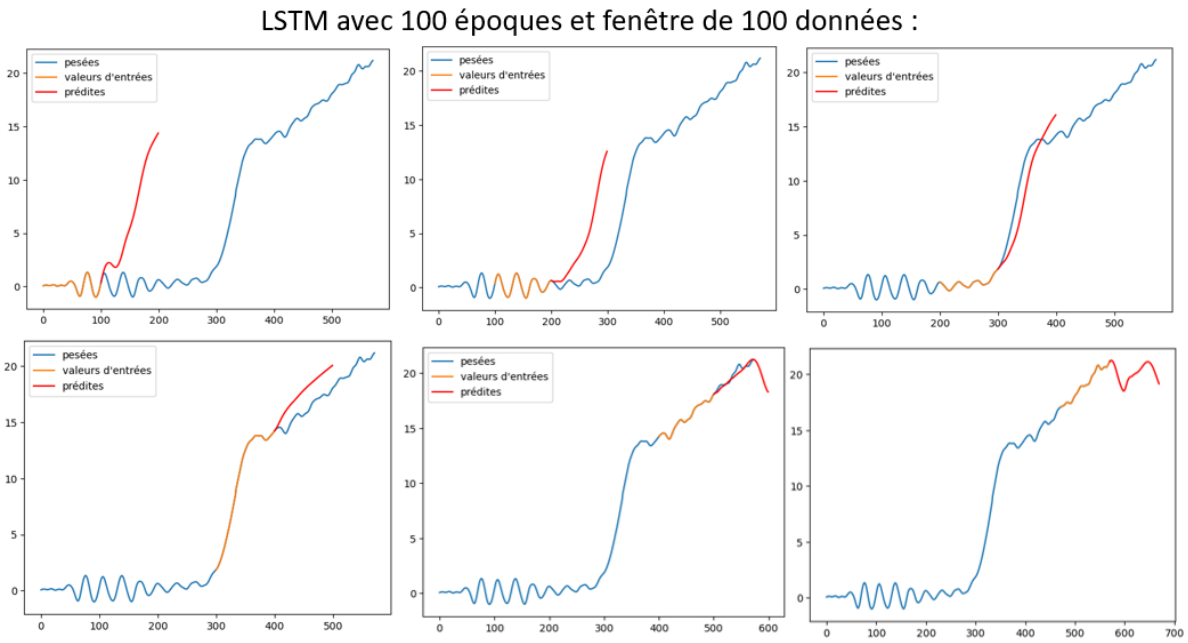



Figure 18. Tracés des prédictions faites à l'intérieur des données pour 100 époques

Les Figure 17 et Figure 18 regroupent les prédictions réalisées à l'intérieur des données pour les cinq premiers tracés et le dernier tracé correspond à la prédiction hors données. Ainsi dans le réseau LSTM entrainé avec seulement 20 époques, nous pouvons voir que le réseau comprend qu'il y a une tendance qui croit, mais il l'applique à chacun des intervalles de la phase. Pour le réseau ayant été entrainé sur 100 époques, il comprend qu'une croissance importante du signal a lieu au début, mais après quelques oscillations puis il suit la tendance qui augmente. Cependant il réalise une diminution durant la partie d'allongement de la phase en dehors des données.

Dans les deux cas, les données sont lissées, car le réseau n'est pas assez complexe ou assez entrainé pour comprendre les variations causées par les vibrations du système et le bruit environnant.

### 4.2.4 Modèle 2 : Vérification des critères de prédiction et de durée

Comme évoqué dans l'approche, ce modèle permet de vérifier que les modifications apportées à la phase de grand débit permettront de toujours répondre aux exigences de temps et de précision en prédisant le temps restant du cycle et le poids final (voir Annexe B).

Les données fournies en entrée de ce modèle correspondent à une fenêtre de données parmi les dernières données de la phase de grand débit, c'est-à-dire dans notre cas, les nouvelles données prédites pour allonger la phase de grand débit. Il y a donc deux sorties : le temps restant qui va du début de la phase de changement de débit à la fin de la phase de stabilisation et le poids final qui est le dernier poids filtré de la phase de stabilisation.

La Figure 19 représente l'approche utilisée pour ce modèle.

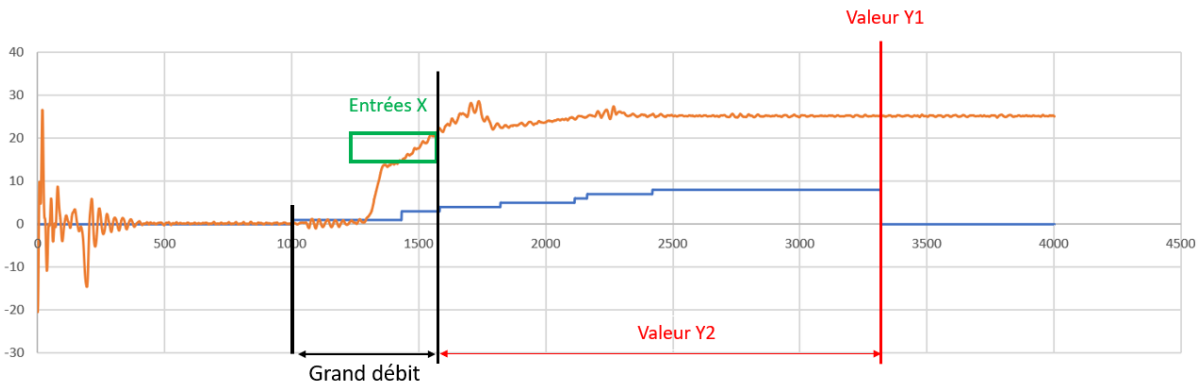



Figure 19. Graphe expliquant le fonctionnement du modèle 2

Les différentes valeurs de paramètres testées pour ce modèle sont affichées dans le Tableau 13.

Les différentes combinaisons de paramètres sont évaluées suivant le score de la valeur de perte. La valeur de perte est fournie lors de l'entrainement du réseau après chaque époque et permet d'évaluer si les prédictions du modèle sont éloignées des valeurs réelles dans le jeu d'entrainement. Il est donc important de minimiser cette valeur.

Tableau 13. Configuration des algorithmes du modèle 2

| Paramètres | Valeurs possibles | Meilleure combinaison |
|---|---|---|
| Taux d'apprentissage | 0,0001 ; 0,001 | 0,0001 |
| Taille du lot (*batch_size*) | 128 | 128 |
| Nombre de noeuds LSTM | 64 | 64 |
| Nombre d'époques | 100 | 100 |
| Taille de la fenêtre | 100; 50; 20 | 100 |

Afin de pouvoir avoir une idée de l'efficacité du modèle, le Tableau 14 donne les valeurs de sept cycles de pesage qui n'étaient pas inclus dans les données d'entrainement.

Tableau 14. Résultats du modèle 2 sur des données de tests

| | **Poids (Y1) attendus** | **Temps (Y2) attendus** | **Poids (Y1) prédits** | **Temps (Y2) prédits** |
|---|---|---|---|---|
| Pesée 1 | 25,003 | 2682 | 25,00188 | 2757,2244 |
| Pesée 2 | 24,995 | 2674 | 25,001902 | 2757,2056 |
| Pesée 3 | 25,029 | 2671 | 25,00206 | 2757,6992 |
| Pesée 4 | 25,024 | 2839 | 25,001968 | 2757,519 |
| Pesée 5 | 24,977 | 2750 | 25,002254 | 2758,2913 |
| Pesée 6 | 24,994 | 2697 | 25,001818 | 2757,0752 |
| Pesée 7 | 24,996 | 2631 | 25,001661 | 2756,4202 |

Concernant les résultats du Tableau 14, les données de tests sont des couples entrées-sortie réalisés de la même manière que les données d'entrainement, mais avec des cycles enregistrés qui n'ont pas été montrés au modèle d'entrainement. Nous pouvons donc observer que les résultats obtenus sont cohérents et du même ordre de grandeur entre les valeurs prédites et celles attendues. Cependant, pour notre application de généralisation, telle que présentée précédemment, le modèle 2 a pour but d'être appliqué à la suite du modèle 1 et sur des données fournies directement par celui-ci. Ainsi, le Tableau 15 regroupe les résultats obtenus après avoir appliqué le modèle 2 entrainé sur les données prédites par le premier modèle.

Tableau 15. Résultats du modèle 2 appliqués aux résultats du modèle 1

| | **Poids prédits** | **Temps prédits** |
|---|---|---|
| Pesée 1 | 28,05217 | -29371,992 |
| Pesée 2 | 27,348698 | -22765,215 |
| Pesée 3 | 31,281803 | -58997,742 |
| Pesée 4 | 30,78451 | -54410,94 |
| Pesée 5 | 34,602463 | -89559,01 |
| Pesée 6 | 30,64135 | -53100,24 |

Les résultats obtenus pour les temps ne sont pas ceux attendus, mais étaient prévisibles étant donné que le modèle n'a jamais vu de données similaires lors de ses entrainements. Cependant, les données de poids obtenus sont intéressantes, car il est possible de penser que le modèle a fait un lien entre la valeur des poids durant cette phase et leur impact sur le poids final. Nous pouvons supposer que le modèle a compris que la durée du remplissage a un impact sur le poids final et que par conséquent si l'on fait durer la phase de grand débit plus longtemps que prévu alors il est normal d'obtenir un poids final plus grand.

Ainsi, pour régler ce problème, deux solutions étaient possibles. La première consistait à enregistrer de nouvelles données pour lesquelles nous modifions les paramètres du système de manière à trouver un moyen d'allonger la phase de grand débit. L'ajustement des paramètres pour y arriver aurait été de donner au système un poids cible plus grand sans changer les pourcentages d'ouverture de l'écoulement du produit dans la balance. Cela aurait ainsi permis d'obtenir des données avec une phase de grand débit allongée et sur lesquelles le modèle aurait pu s'entrainer. Cependant ce n'était pas envisageable, car cela représentait beaucoup de données supplémentaires et que ça aurait permis de résoudre le problème dans notre cas, mais pas pour appliquer le modèle à d'autres produits. L'autre solution était de changer d'approche, car celle-ci ne permettait pas d'être généralisable à différents produits,

Pour conclure sur les résultats obtenus à la suite de ces deux premiers modèles, nous avons été amenés à reconsidérer l'approche envisagée. Notamment, car ces premiers résultats nous ont permis de réaliser que cette approche était limitée en termes de généralisation. En

effet, des modèles d'apprentissage supervisé et basés sur la prédiction amèneront à des résultats incohérents lorsqu'ils sont appliqués à des données très différentes de celles présentes dans les données d'entrainement du modèle. Ainsi, concernant le développement d'un modèle qui puisse devenir généralisable et utilisable pour ensacher des produits différents ou des poids cibles variables, l'étude de la littérature a permis d'identifier l'apprentissage par renforcement comme possibilité. Cette approche sera développée dans le chapitre suivant. L'approche d'apprentissage par renforcement sera également combinée avec une étape préalable de filtrage des données en utilisant différentes techniques.

## 4.3 DISCUSSION

Les résultats obtenus avec cette approche, ont montré un manque d'adaptabilité à de nouvelles valeurs. Cela n'est donc pas optimal lorsqu'il s'agit de données bruitées qui sont très variables. Suite à ces premiers résultats obtenus lors de l'application de la première approche, nous avons donc été amené à revoir notre approche et à la repenser afin de favoriser une seconde approche qui associe le filtrage à l'apprentissage par renforcement. Ajouter une étape de filtrage du signal de poids brut au préalable a pour but de faciliter cette optimisation. De plus, l'API actuel est conçu de cette façon et comme nous avons pu le voir dans le chapitre de revue de littérature, il est très courant de filtrer le signal de poids brut récupéré des balances industrielles.

Néanmoins, le travail réalisé pour ces premiers résultats a été intéressant à développer même s'il n'est pas le plus adapté à notre application, qui comporte trop d'éléments changeants. Ce travail permet tout de même de fournir à l'entreprise des informations dans le cas où ils souhaiteraient développer des modèles de ce type sur des données de séries temporelles.

## 4.4 SYNTHÈSE DU CHAPITRE

Ce chapitre a permis de montrer les premiers résultats obtenus suite à l'application de la première approche. Nous avons donc pu conclure que l'idée initiale de baser l'optimisation de cycle directement sur le signal brut, c'est-à-dire celui incluant les perturbations et les bruits perçus, ne permettait pas d'aboutir à une solution généralisable. Un changement d'approche s'appuyant sur le principe de fonctionnement de la balance a donc été envisagé. Il s'agit donc de combiner une étape de filtrage avec l'optimisation du cycle, mais en essayant des modèles d'IA pour ces deux étapes. Cette approche en deux étapes et les résultats associés sont détaillés dans le chapitre suivant.

# CHAPITRE 5
# OPTIMISATION EN DEUX ÉTAPES

## 5.1 L'APPROCHE

La seconde approche consiste à combiner le filtrage des données bruitées à l'apprentissage par renforcement. Cette approche d'optimisation du cycle de pesage comprend donc une étape de filtrage et une étape d'optimisation du cycle, qui s'appuie sur le filtrage réalisé. L'objectif est d'obtenir un modèle général et adaptable en termes de prises de décisions et de gestion de bruit capté, similaire au fonctionnement de l'API actuel. Les résultats pour ces deux étapes sont détaillés dans ce chapitre.

Les résultats pour l'étape de filtrage sont présentés en premier. Comme l'ont montré les résultats de l'approche en une seule étape, l'intégration d'une étape de filtrage dans la seconde approche s'est avérée indispensable. Différentes catégories de modèles seront développées : des filtres numériques, des modèles d'apprentissage automatique et des modèles d'apprentissage profond. Pour rappel, nous ne connaissons pas le filtre utilisé par l'API actuel, et l'enjeu de ce projet réside dans l'utilisation des poids bruts autant que possible. Ainsi, nous cherchons à filtrer le signal de manière autonome.

Comme discuté dans le CHAPITRE 2 différentes méthodes existent pour optimiser les décisions prises par un API. Les méthodes suivantes sont des méthodes d'optimisation :

- La programmation dynamique : Elle consiste à résoudre un problème en le divisant en sous-problèmes plus petits. Ces sous-problèmes sont alors optimisés, et les solutions trouvées chacun, permettent de résoudre le problème global. Cependant, pour un processus cyclique impacté par son environnement, cette méthode peut s'avérer complexe à mettre en place. En effet, cette technique demande de modéliser précisément le système, or toutes les peseuses-emballeuses installées chez les clients de l'entreprise ne sont pas toutes identiques. Il peut y avoir différentes formes de trémie ou de bac de pesage en fonction du type de

produits et des quantités dans lesquelles le client à l'habitude de les reconditionner. Ainsi, l'optimisation du système ne serait applicable qu'au système d'essais. Une méthode plus adaptable aux changements est donc à envisager.

- Les méthodes heuristiques : Ces méthodes utilisent des règles et des stratégies basées sur une connaissance du domaine. Dans le cas de situations changeantes, cette technique demandera l'intervention régulière d'un expert du domaine afin de modifier les paramètres ou les différentes règles.

- L'optimisation par seuil : Cette méthode consiste à définir un seuil pour chaque étape du cycle et à lancer des actions lorsque ces seuils sont atteints. Il s'agit d'une méthode facile à mettre en place mais qui nécessite des ajustements en fonction du poids cible et du produit principalement. Comme évoqué précédemment, la variabilité des systèmes et des conditions environnantes implique également des ajustements des seuils.

- L'optimisation par taux de variation : Cela consiste à surveiller l'évolution du taux de variation du poids dans la balance et à lancer des actions en fonction de ce taux. Dans notre cas, le taux de variations peut être faussé par les vibrations causées par l'impact du produit ou le bruit.

Même si le fonctionnement de l'API actuel nous est inconnu, nous pouvons voir qu'il utilise une méthode par seuils. En effet, dans les différentes données qui nous sont transmises, les valeurs des seuils sont précisées. Il s'agit des points de consigne du changement de débit et de fermeture détaillés dans le troisième chapitre. Nous savons que ces valeurs sont redéfinies pour chaque nouveau cycle mais nous ne savons pas de quelle manière elles sont calculées. Les seuils sont donc adaptatifs. Pour répondre à notre problème, nous aurions pu choisir d'utiliser une méthode similaire en prédisant de meilleures valeurs de seuils à l'aide de l'apprentissage supervisé. Cependant, cette méthode n'aurait pas pu s'adapter à des situations inconnues, comme cela a été le cas lors de la première approche détaillée dans le chapitre précédent. De plus, le but recherché dans ce projet n'était pas de reproduire le système existant mais d'explorer des techniques liées à l'IA. Après une recherche de l'état

de l'art, notre choix s'est porté sur l'apprentissage par renforcement. Pour cette étape, nous n'avons pas testé d'autres approches d'optimisation pour comparer car le système actuel nous sert d'exemple de comparaison, qu'il faut dépasser en termes de performances.

Dans le cadre de l'optimisation de notre système, l'apprentissage par renforcement nous a donc semblé être adéquat car ses capacités d'adaptation lui permettent d'être efficace dans des environnements complexes et dynamiques. De plus, l'objectif principal de l'apprentissage par renforcement est d'apprendre les meilleures stratégies de contrôle afin de maximiser ses récompenses, or cet objectif est également le nôtre. De plus, comme nous avons pu le voir dans l'état de l'art, cette méthode a déjà été mise en place pour gérer efficacement des ressources. Notre cas est donc adapté à cette méthode.

Pour rappel, dans l'apprentissage par renforcement, des actions (décisions) sont prises de manière itérative à chaque pas de temps. Dans l'apprentissage supervisé, qui est celui appliqué pour les résultats du précédent chapitre, un entrainement est réalisé puis des données sont transmises en entrées et une seule exécution amène à des prédictions. Alors que dans l'apprentissage par renforcement, plusieurs exécutions sont réalisées en fonction des interactions que l'agent a avec son environnement. En effet, l'agent qui est celui qui prend des décisions, reçoit trois informations à chaque fin d'épisode : le dernier état, la dernière action et la récompense. Un épisode pouvant être assimilé dans notre cas à un cycle de pesage complet. Les pas de temps de ce cycle sont nommées itérations dans ce contexte. Ces éléments permettent donc de donner à l'agent l'information de ce que ses décisions ont eu comme impact sur le résultat final de l'épisode et si cela a conduit aux résultats souhaités. Il se base ensuite sur ces éléments pour ajuster sa politique. La politique est la stratégie suivie par l'agent pour prendre ses décisions. En fonction de cette politique, il prend plus ou moins de risques en choisissant l'action qu'il va effectuer à chaque pas de temps. Cette politique évolue après chaque épisode, c'est-à-dire après l'exécution d'un cycle complet.

La Figure 20 résume notre approche à deux étapes et détaille les modèles qui sont utilisés dans chaque étape. L'étape de filtrage permet à l'agent de prendre des décisions sur les bonnes mesures de masses et pas sur les données bruitées lors de l'étape d'optimisation. Lors de la présentation des résultats obtenus à l'aide de ces modèles, un bref rappel sera fait concernant leur rôle et leur fonctionnement.

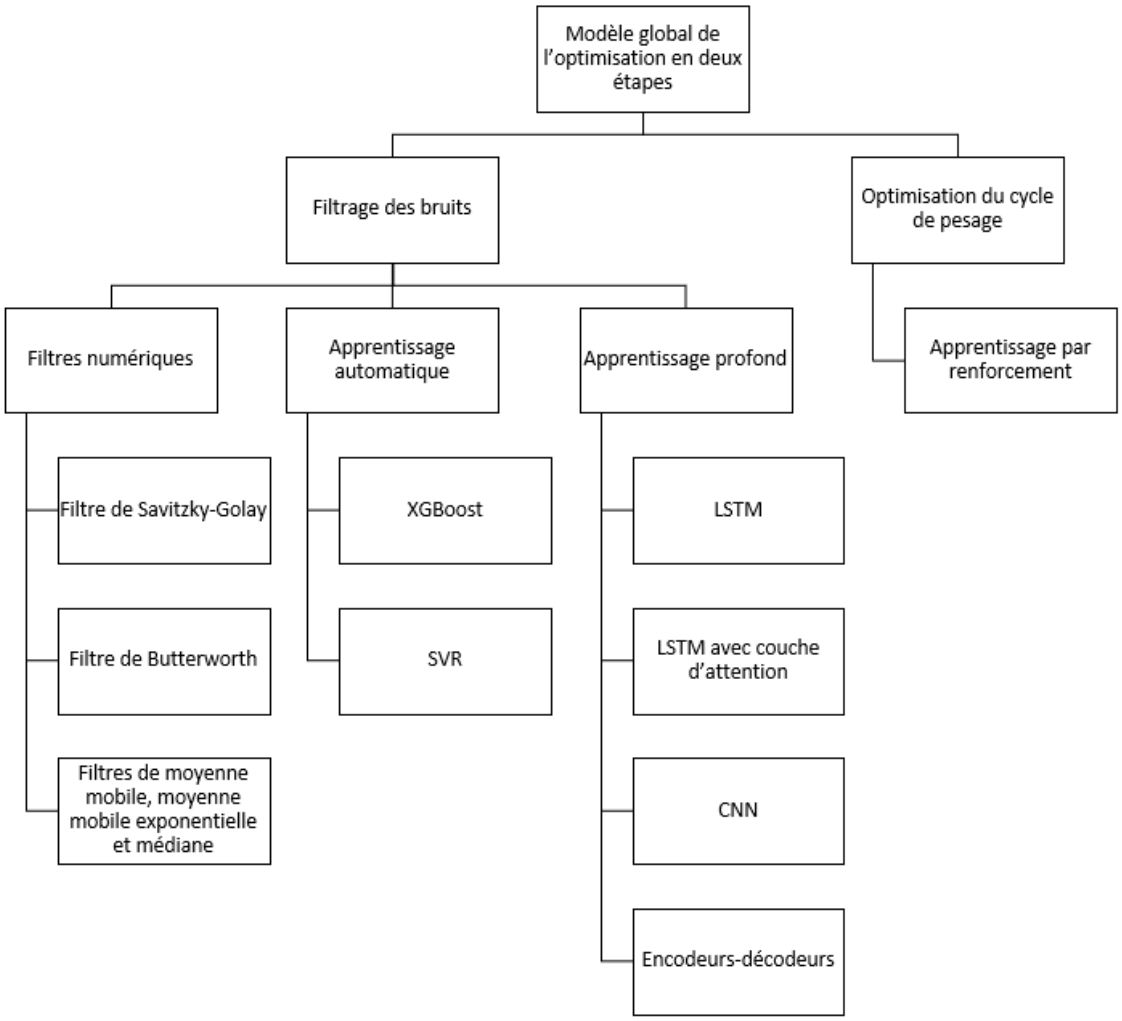



Figure 20. Schéma résumant l'approche en deux étapes et les techniques employées

Pour chaque modèle étudié, différentes valeurs de paramètres seront testées afin de trouver ceux qui permettent d'obtenir les meilleurs résultats. Il n'est pas possible de tester toutes les valeurs des paramètres, car cela demanderait des ressources informatiques non

disponibles. Ainsi, une liste non exhaustive est déterminée pour chaque paramètre soit à partir de valeurs aléatoires, soit, car certaines valeurs sont connues pour mieux fonctionner pour des séries temporelles. Une grille de paramètres est ensuite constituée afin de tester chaque combinaison de paramètres pour le modèle étudié. Puis les résultats sont comparés suivant différentes métriques en fonction du modèle testé. La meilleure combinaison de paramètres est finalement appliquée aux données souhaitées. Le choix des paramètres communs aux différents modèles appliqués est détaillé ci-après. Pour les valeurs des paramètres propres aux modèles, celles-ci sont détaillées lors de la présentation des résultats de ces modèles.

Dans cette partie, les modèle sont appliqués aux différentes phases du cycle, ainsi les valeurs testées pour chaque hyperparamètres communs sont les suivantes :

- Les valeurs des fenêtres de données testées sont les suivantes : 20, 40, 60, 80, 100 et 150. En fonction de l'étape du cycle, les phases ont des durées très variables, ainsi pour cette raison ces valeurs couvrent un grand intervalle.

- La valeur du nombre d'époques est fixée à 100, avec un enregistrement du modèle toutes les 10 époques pour pouvoir tester les résultats pour différents niveaux d'entrainement de chaque modèle. De plus, un arrêt anticipé est mis en place pour se produire lorsque l'apprentissage du modèle n'évolue plus. Cela a pour effet de gagner du temps dans l'exécution du programme et principalement d'éviter le sur-apprentissage.

- Concernant les autres hyperparamètres comme la taille de lot et le taux d'apprentissage, compte tenu de la grande variété de modèles testées sur les différentes phases et pour les différentes fenêtres de données, nous avons choisi d'apporter moins de variabilité. En effet, le temps d'entrainement étant assez conséquent, le choix des autres hyperparamètres s'est basé sur les meilleures configurations obtenues lors de l'approche précédente.

Les hyperparamètres des modèles d'apprentissage par renforcement étant assez spécifiques pour certains, ceux-ci sont détaillés dans la section des résultats de cette approche.

## 5.2 RÉSULTATS DU FILTRAGE DES BRUITS PAR L'APPRENTISSAGE SUPERVISÉ

### 5.2.1 Filtres numériques

Différents filtres numériques ont été appliqués sur les données afin d'avoir des résultats à comparer lors de l'utilisation de l'IA, mais également pour essayer de comprendre un peu mieux le fonctionnement de l'API et essayer de comprendre la technique de filtrage employée dans celui-ci (voir Annexe C).

- Filtre de Savitzky-Golay :

Ce filtre permet de lisser les données en utilisant une régression polynomiale, comme cela a été expliqué plus en détail dans le premier chapitre de ce mémoire. Ce filtre est déjà implémenté dans la bibliothèque SciPy[3] de Python. Ainsi, seules les valeurs des paramètres du filtre et les données à filtrer sont à fournir. Les paramètres du filtre sont la taille de la fenêtre de données et le degré du polynôme.

Différentes valeurs de paramètres ont été testées pour évaluer l'impact de la valeur de la taille de la fenêtre sur l'efficacité de filtrage de ce filtre. La courbe du poids filtré a été décalée pour permettre une comparaison plus facile avec le signal de poids brut. Les phases de grand débit (1 et 3) et de petit débit (5 et 6) ont été regroupées pour distinguer quatre grandes phases : grand débit, changement de débit, petit débit, fermeture.

Pour chaque phase, la taille de la fenêtre et le degré du polynôme ont été variés. Différentes valeurs ont été testées pour chacune des quatre phases étudiées. Les résultats présentés aux Figure 21 et Figure 22, ont une valeur d'ordre du polynôme de 1. Les tailles de fenêtre de 20 (valeur minimale) et 150 (valeur maximale) sont utilisées afin de montrer

[3] Documentation SciPy : https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html (dernier accès le 5 juillet 2024).

l'impact de la valeur de ce paramètre sur les résultats. Notons que pour chacun des tracés, y compris ceux présentés dans la suite de ce chapitre, lorsqu'on visualise la progression des types de poids dans le temps, les abscisses représentent les millisecondes écoulées et les ordonnées indiquent les poids en kilogrammes.

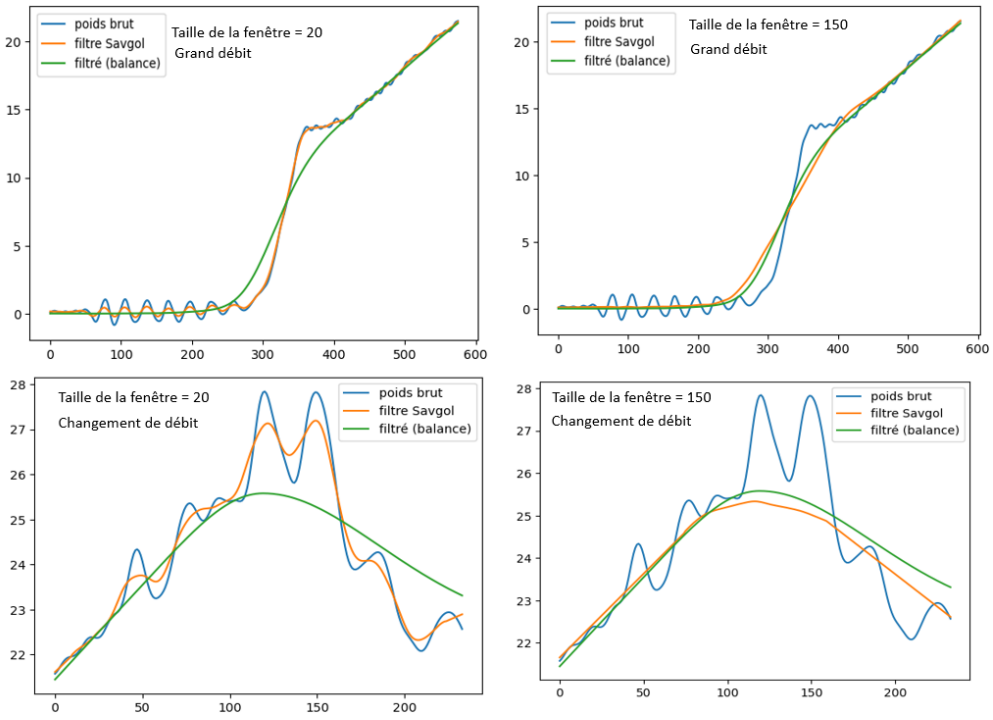



Figure 21. Résultats pour les phases de grand débit et de changement de débit du filtre de Savitzky-Golay pour des fenêtres de 20 et 150 données

Afin de vérifier l'efficacité du filtrage pour les données correspondantes, nous avons décidé de présenter le poids brut et le poids filtré en rectifiant le décalage observé dans les données enregistrées. Ainsi, lorsque la valeur de la fenêtre est de 20, nous pouvons observer que le signal est moins lissé qu'avec une valeur de 150. Le signal obtenu avec une taille de fenêtre de 150 données se rapproche plus de celui fourni par l'API. Cependant, pour une utilisation en temps réel, une fenêtre plus petite est souhaitée afin d'éviter un délai de traitement trop long lors de changements dans le signal.

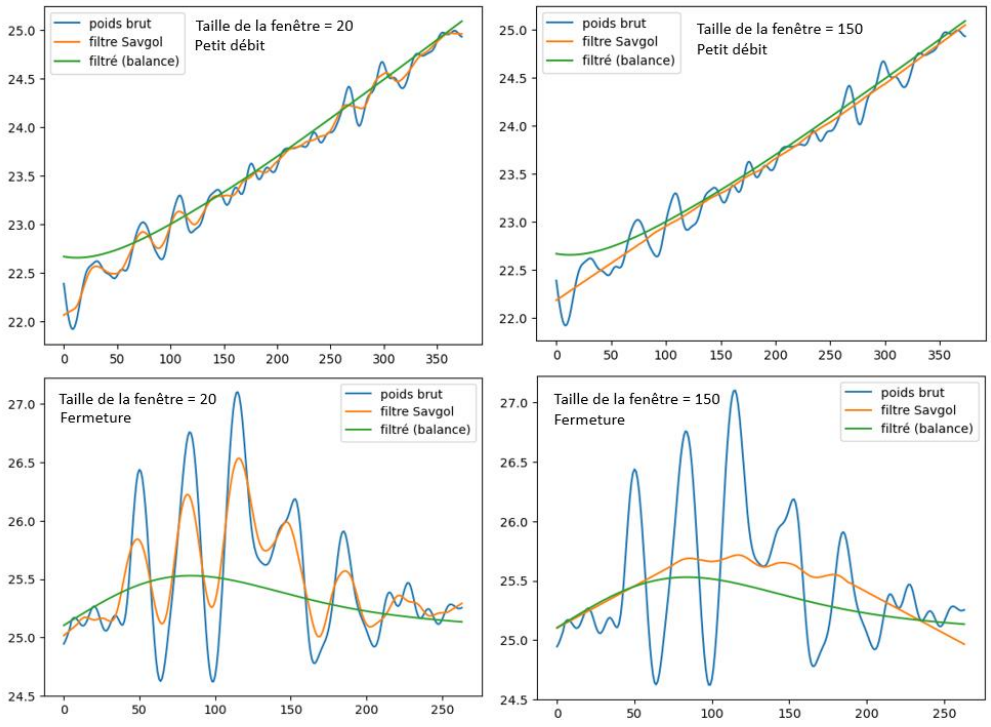



Figure 22. Résultats pour les phases du petit débit et de fermeture du filtre de Savitzky-Golay pour des fenêtres de 20 et 150 données

- Filtre de Butterworth :

Ce filtre atténue les fréquences indésirables tout en préservant la forme d'onde du signal d'entrée. Les fréquences sont atténuées sans modifier les caractéristiques de forme d'onde du signal. Les paramètres du modèle inclus dans la bibliothèque SciPy de Python[4] sont la fréquence d'échantillonnage, la fréquence de coupure du filtre, l'ordre du filtre et le type de filtre.

[4] Documentation SciPy : https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html#butter (dernier accès le 5 juillet 2024).

La Figure 23 regroupe les résultats obtenus en appliquant le filtre de Butterworth à un cycle de pesage. Chaque tracé correspond à une phase du cycle. Les valeurs des paramètres sont les mêmes pour chaque phase et sont les suivantes :

- Ordre du polynôme : 1
- Fréquence d'échantillonnage : 627
- Fréquence de coupure : 3

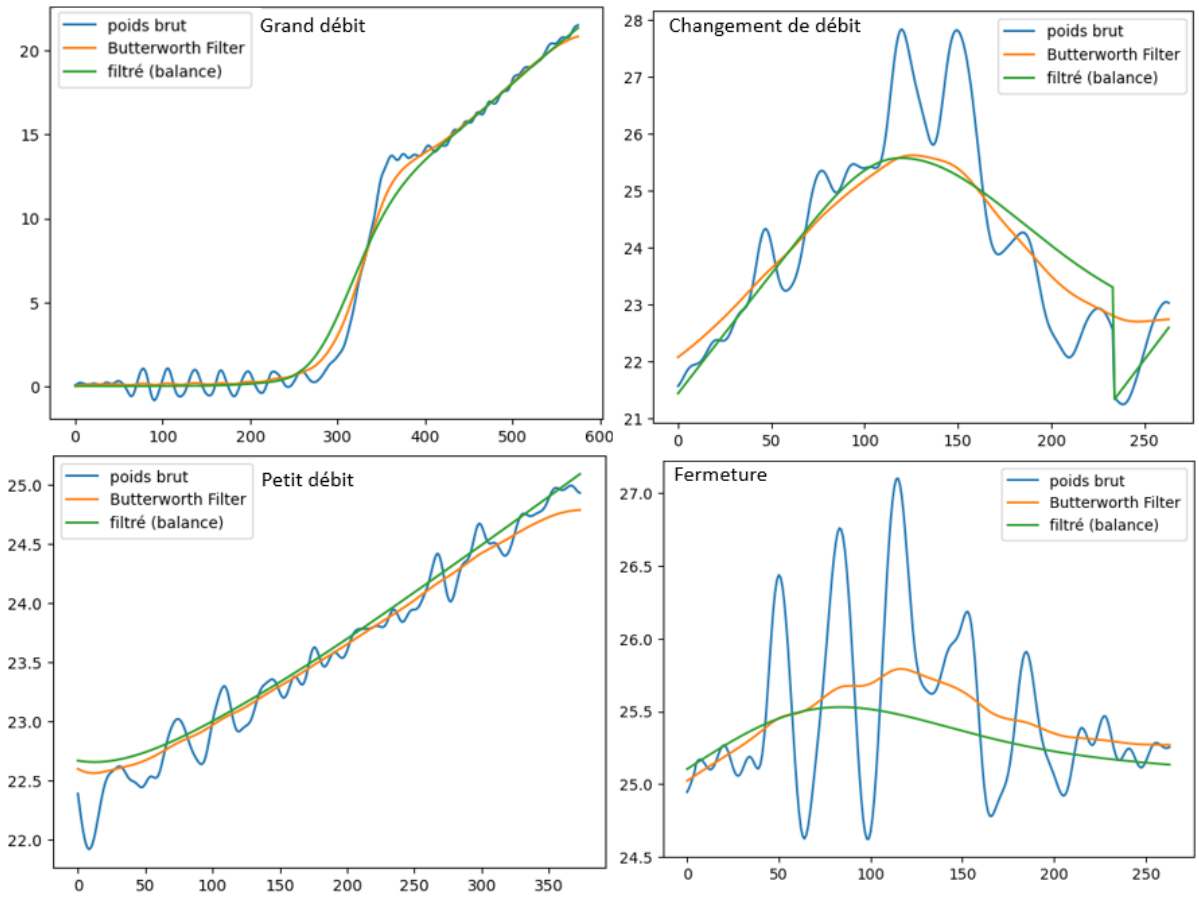



Figure 23. Résultats du filtre Butterworth sur les quatre phases

Pour les trois premières phases du signal, les résultats sont assez proches du signal filtré transmis par l'API actuel. Pour la phase de fermeture, nous pouvons remarquer que ce filtre atténue fortement les oscillations du signal, mais la courbe est moins lissée.

- Filtres de moyenne mobile, de moyenne mobile exponentielle et de médiane :

Le paramètre réglable de ces trois filtres est le nombre de valeurs que comprend la fenêtre glissante de données. La fonction mathématique appliquée à cette fenêtre est ce qui diffère dans chaque filtre. Le filtre de moyenne mobile fait la moyenne des données comprises dans la fenêtre. Le filtre de moyenne mobile exponentielle fait également la moyenne des données, mais accorde un poids plus important aux données récentes. Quant au filtre de médiane, il remplace le point central de la fenêtre par la valeur médiane de celle-ci. Pour chacun des trois filtres appliqués aux données, différentes valeurs de fenêtre ont été testées. Les résultats obtenus après l'application de chaque filtre à la phase de grand débit sont présentés dans la Figure 24 pour les valeurs de taille de fenêtre de 20 et 150.

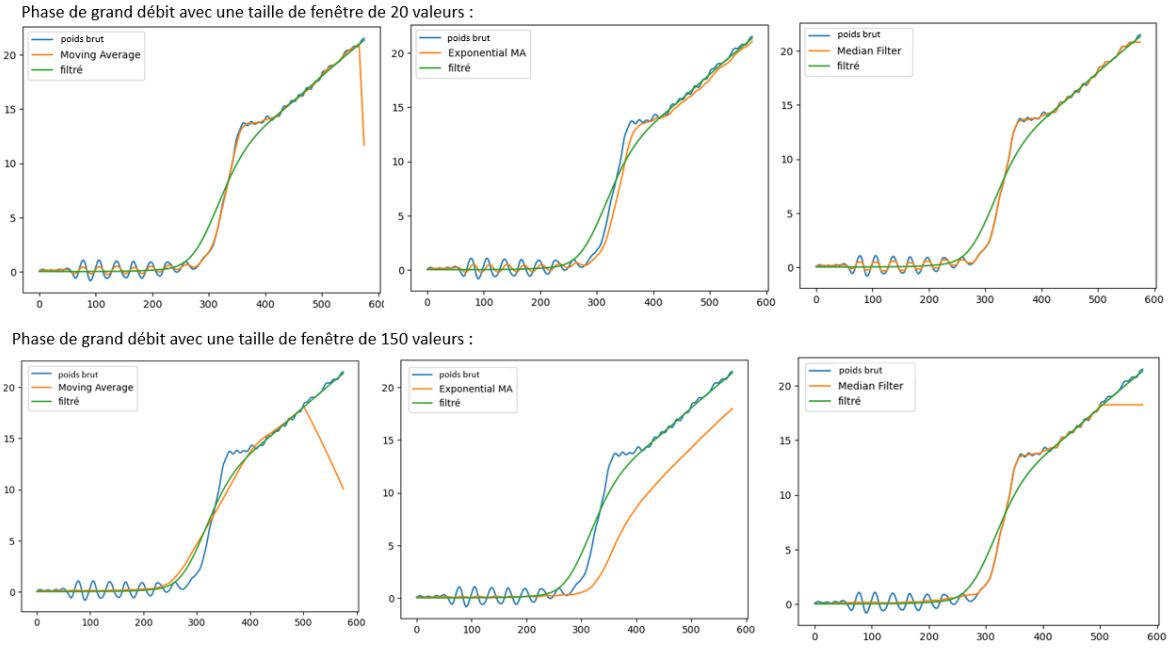



Figure 24. Résultats obtenus pour la phase de grand débit pour les filtres de moyenne mobile, moyenne mobile exponentielle et médiane

Les traits présents à la fin des résultats pour le filtre de moyenne mobile sont causés par le fait que pour les dernières fenêtres de données, il n'y a pas suffisamment de valeurs restantes. Hormis le filtre de moyenne mobile pour une fenêtre de 150 valeurs qui se rapproche du signal filtré de la balance, les signaux obtenus correspondent à des atténuations des variations du signal brut. En effet, ces signaux suivent principalement le signal brut tout en lissant plus ou moins efficacement les oscillations enregistrées.

### 5.2.2 Apprentissage automatique

- Modèle SVR (*Support Vector Regression*) :

Le modèle SVR est entrainé en utilisant les données de poids bruts en entrées et les données de poids filtrés en sortie (Voir Annexe D). Les données sont entrainées séparément pour chaque phase. Cependant, comme cela a pu être observé en analysant les données fournies par le système, un décalage entre le signal de poids bruts et celui de poids filtrés est présent. Ainsi, lors de l'entrainement, un décalage est pris en compte afin que les données filtrées correspondent aux mêmes moments que les données de poids bruts. Le décalage n'étant pas le même tout au long du cycle, un décalage de 115 données a été appliqué pour les phases de grand débit et de changement de débit. Puis, un décalage de 180 données est utilisé pour le petit débit et la fermeture de l'écoulement du produit.

En ce qui concerne la valeur des paramètres du modèle, le noyau utilisé est le noyau « rbf » pour « *Radial Basis Function* ». Ce noyau est couramment utilisé pour les modèles de régression et pour lesquelles la relation entre les données d'entrées et de sorties est non linéaire. Le paramètre qui gère la pénalité lors des erreurs de l'entrainement est de 100. Les paramètres gamma et epsilon valent tous les deux 0,1.

Les résultats présentés à la Figure 25 sont issus d'un cycle de pesée qui ne fait pas partie des données d'entrainement du modèle.

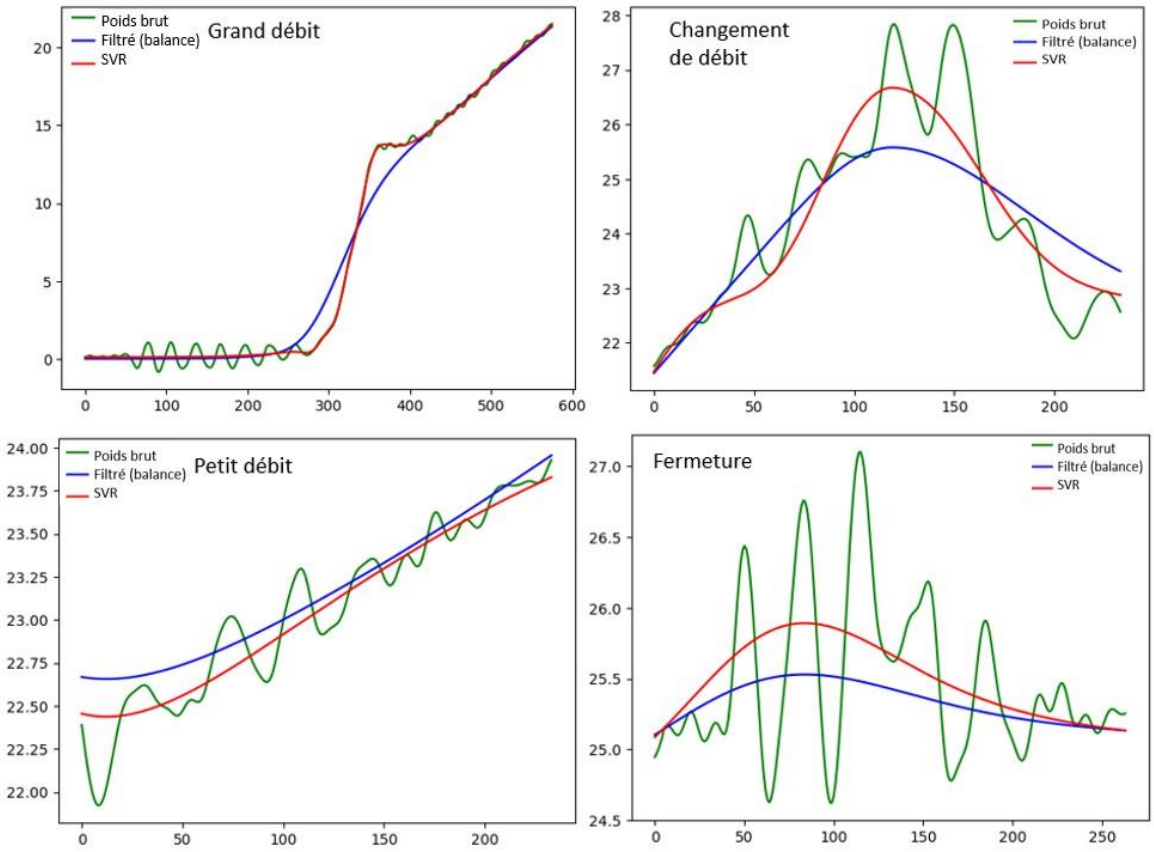



Figure 25. Résultats pour le modèle SVR sur un cycle de pesage

Les quatre phases ont été représentées de façons distinctes afin de voir au mieux les résultats de chacune sans perdre d'informations, notamment pour les phases où il y a une évolution moins importante du poids. Ainsi, nous constatons que ce modèle SVR a bien réalisé un lissage des données, mais ce lissage ne correspond pas au filtrage effectué par l'API de la balance, alors que c'est sur les données de l'API qu'il a été entrainé.

- Modèle XGBoost (*eXtreme Gradient Boosting*) :

Ce modèle appliqué à des signaux bruités apprend à prédire la valeur non bruitée à partir des caractéristiques du signal bruité.

Tout comme pour le modèle SVR, le modèle est entrainé avec en entrées les données de poids bruts et en sorties les données de poids filtrés. De plus, les mêmes décalages sont appliqués pour les différentes phases du cycle, c'est-à-dire 115 données pour les phases de grand débit et de changement et 180 données pour celles de petit débit et de fermeture (voir Annexe D).

Les paramètres correspondant à la profondeur maximale des arbres de décisions et la contribution de chaque arbre à la mise à jour du modèle valent respectivement 3 et 0,1. Pour la fonction d'objectif qui doit être minimisée durant l'entrainement, il s'agit de l'erreur quadratique moyenne. Le nombre d'arbres de décision à entrainer, c'est-à-dire le nombre d'itérations du modèle, est de 50.

La Figure 26 présente les résultats obtenus pour le modèle XGBoost appliqué à une pesée. Les différentes parties du débit sont également représentées séparément afin de bien voir l'efficacité du lissage sur chacune de ces parties. Il est possible de remarquer que pour la phase de grand débit, ce modèle XGBoost permet d'obtenir des résultats proches du signal filtré. Pour les autres parties, la tendance suivie par le signal est respectée, mais un écart est observable.

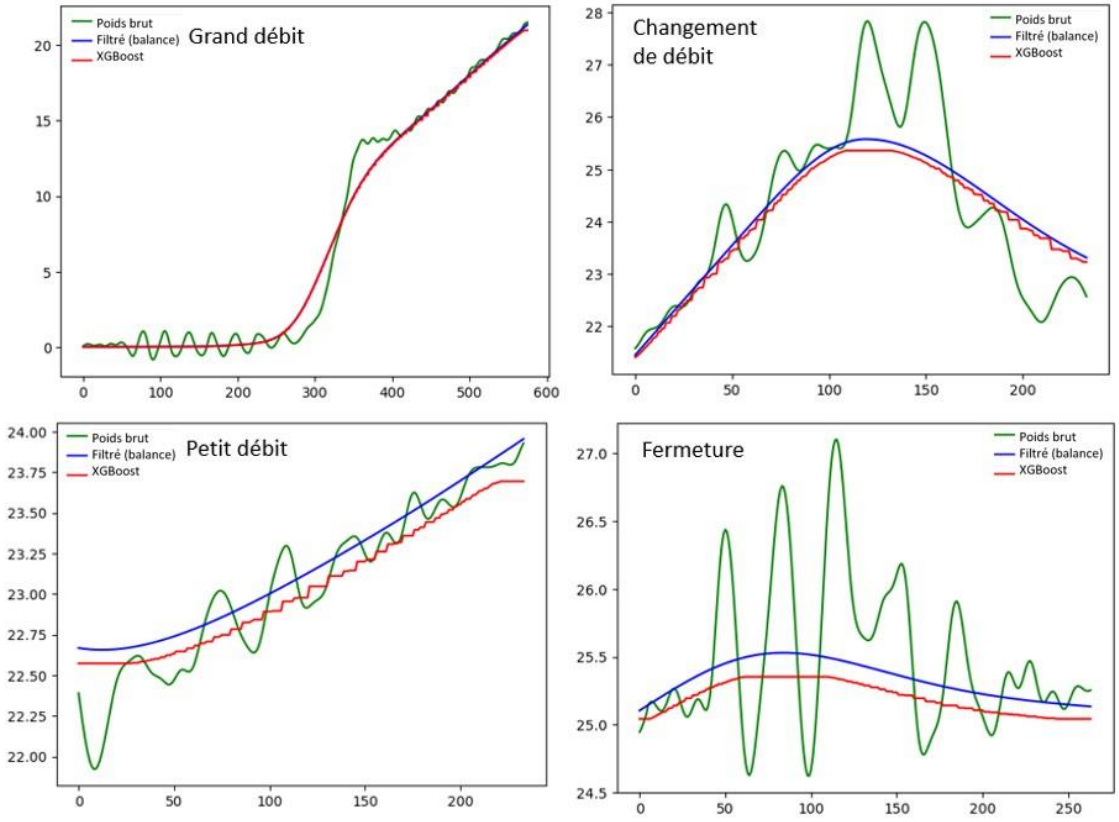



Figure 26. Résultats du modèle XGBoost sur un cycle de pesage

### 5.2.3 Apprentissage profond

- Prédictions de données filtrées :

Comme discuté dans la section précédente, en apprentissage automatique, nous avons utilisé des algorithmes pour prédire directement les valeurs filtrées à partir des données brutes. Par exemple, si l'entrée est une série de poids bruts, le modèle est entraîné pour prédire les poids filtrés. Cette prédiction se fait en utilisant des caractéristiques dérivées des données d'entrée, et le modèle apprend à mapper ces caractéristiques aux poids filtrés.

En revanche, en apprentissage profond, notre objectif en adoptant une approche différente est de capturer des dynamiques temporelles et des tendances locales plus

complexes dans les données. Plutôt que de simplement mapper les poids bruts aux poids filtrés, nous glissons une fenêtre sur les poids bruts pour prédire le poids suivant dans cette fenêtre. En utilisant des architectures comme LSTM et CNN, le modèle peut apprendre des représentations plus riches et complexes des données (voir Annexe E). Cela permet d'améliorer la précision des prédictions en tenant compte des relations entre les points de données successifs, ce qui est crucial pour des séries temporelles. Notre souhait avec cette approche est de mieux filtrer et prédire les poids en utilisant les dynamiques intrinsèques des données.

Pour que les valeurs de poids filtrés correspondent à celles de poids bruts, nous avons fourni en sortie des données décalées de 115 valeurs, comme nous l'avons fait en apprentissage automatique pour le grand débit. Cela permet de faire correspondre les événements dans les deux signaux en prenant en compte le délai de traitement du filtrage. Les modèles testés sont regroupés dans le Tableau 16.

Tableau 16. Meilleures configurations des modèles de prédictions des données filtrées

| **Modèle** | **Paramètres** | **Meilleure configuration** |
|---|---|---|
| LSTM | Taux d'apprentissage | 0,001 |
| | Taille du lot | 128 |
| | Nœuds LSTM | 64 |
| | Fenêtre de données en entrée | 100 |
| CNN | Taux d'apprentissage | 0,0001 |
| | Taille du lot | 256 |
| | Nombre de filtres | 128 |
| | Taille du noyau | 5 |
| | Taille de mise en commun | 2 |
| | Fenêtre de données en entrée | 40 |

Ils ont été testés pour 100 époques. Seuls les résultats pour le meilleur modèle LSTM sont présentés, car les modèles CNN n'ont pas permis d'obtenir des résultats interprétables. En effet, les résultats des modèles CNN donnent des valeurs nulles au bout de quelques données. Les résultats du meilleur modèle LSTM sont présentés après 20, 40 et 60 époques.

Les valeurs de pertes après ces époques devenaient infinies, ce qui implique des résultats de prédictions lors de l'entrainement très éloignés de ceux souhaités. Les prédictions sont donc réalisées dans le but de lisser les données.

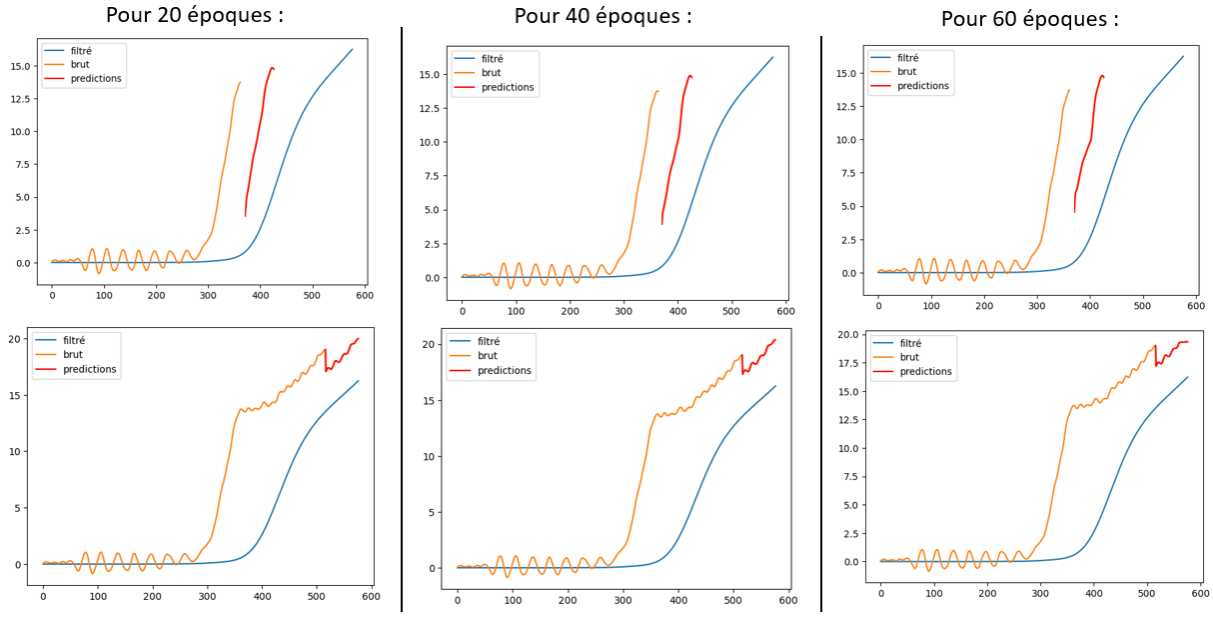



Figure 27. Résultats de prédictions à l'intérieur de la phase

Les tracés de la Figure 27 montrent uniquement la phase de grand débit. Pour chaque valeur d'époque, deux résultats sont proposés. Le premier tracé (en haut) montre une prédiction réalisée au milieu de la phase de grand débit et le second tracé (en bas) affiche les résultats sur le dernier intervalle de la phase. Lors de la première approche présentée dans le CHAPITRE 4, nous avons remarqué que le segment du signal qui comporte l'augmentation de 0 à 15 kg (voir Figure 17 et Figure 18), était déjà une zone problématique pour le modèle. Ainsi dans ce cas également, le modèle comprend qu'une forte augmentation est présente, mais ne la positionne pas forcément au bon endroit. Quant aux tracés du dernier intervalle de la phase, les données sont inférieures et poursuivent le signal avec une tendance et des oscillations semblables au signal de poids brut. Malgré l'utilisation de données de poids filtrées en sortie, le signal n'est pas lissé. Le modèle fournit des résultats ressemblant au poids brut, ce qui n'est pas l'effet attendu.

- Filtrage de séquences de données avec des modèles se rapprochant des encodeurs-décodeurs :

Suite aux résultats obtenus précédemment, il est apparu que garder la même technique que celle appliquée pour faire de la prédiction n'était pas la plus dédiée au filtrage. Nous avons fait l'hypothèse que la meilleure façon de faire du filtrage serait donc d'utiliser des modèles qui se rapprochent des encodeurs-décodeurs, où une séquence de données est fournie en entrée, mais également une séquence de données pour la sortie. Ainsi, les modèles pourraient mieux comprendre qu'en entrée il s'agit d'un signal brut et qu'en sortie c'est une version lissée de ce signal. Dans ce cas, nous gardons le principe de fenêtre glissante afin de fournir plus de données d'entrainements au modèle. Ainsi les différents modèles testés sont : LSTM, LSTM avec couche d'attention et différentes configurations de CNN et d'encodeurs-décodeurs. Le tableau suivant regroupe tous ces modèles, les valeurs de paramètres testés et le temps d'entrainement nécessaire par époques.

Tableau 17. Paramétrage des modèles avec fenêtre glissante

| **LSTM** | | |
|---|---|---|
| Taux d'apprentissage | 0,001 | La durée d'entrainement est de 1500 secondes par époques pour une fenêtre de 20 données et de 2850 secondes pour une de 150 valeurs. |
| Taille des lots | 128 ; 256 | |
| Nœuds LSTM | 32 ; 64 | |
| **LSTM avec couche d'attention** | | |
| Taux d'apprentissage | 0,001 | Durée pour une fenêtre de 20 données : 720 secondes/époques.<br><br>Pour une fenêtre de 100 : 2860 secondes/ époques. |
| Taille des lots | 256 | |
| Nœuds LSTM | 64 | |
| **CNN (2 couches de convolution avec *MaxPooling*) et essais avec 1 et 2 couches denses** | | |
| Taux d'apprentissage | 0,0001 | Durée pour une fenêtre de 20 données : 40 secondes/époques.<br><br>Pour une fenêtre de 150 : 60 secondes/ époques |
| Taille du lot | 128 ;256 | |
| Nombre de filtres | 128 | |
| Taille du noyau | 5 | |
| Taille de mise en commun | 2 | |

| **CNN (2 couches de convolution avec *AveragePooling* et 1 seule couche dense)** | | |
|---|---|---|
| Taux d'apprentissage | 0,0001 | Durée pour une fenêtre de 20 données : 38 secondes/époques.<br><br>Pour une fenêtre de 150 : 57 secondes/ époques |
| Taille du lot | 256 | |
| Nombre de filtres | 128 | |
| Taille du noyau | 5 | |
| Taille de mise en commun | 2 | |
| **CNN (4 couches de convolution avec *MaxPooling* et 1 seule couche dense)** | | |
| Taux d'apprentissage | 0,0001 | Durée pour une fenêtre de 20 données : 230 secondes/époques.<br><br>Pour une fenêtre de 150 : 490 secondes/ époques |
| Taille du lot | 64 | |
| Nombre de filtres | 128 | |
| Taille du noyau | 5 | |
| Taille de mise en commun | 2 | |
| **Encodeur-décodeur CNN à 3 couches de convolution et *AveragePooling*** | | |
| Taux d'apprentissage | 0,0001 | Durée pour une fenêtre de 20 données : 80 secondes/époques.<br><br>Pour une fenêtre de 152 : 130 secondes/ époques |
| Taille du lot | 256 | |
| Nombre de filtres | 128 | |
| Taille du noyau | 5 | |
| Taille de mise en commun | 2 | |
| **Encodeur-décodeur CNN à 3 couches de convolution et *MaxPooling*** | | |
| Taux d'apprentissage | 0,0001 | Durée pour une fenêtre de 20 données : 80 secondes/époques.<br><br>Pour une fenêtre de 152 : 130 secondes/ époques |
| Taille du lot | 256 | |
| Nombre de filtres | 128 | |
| Taille du noyau | 5 | |
| Taille de mise en commun | 2 | |

Dans le cas des CNN, les couches de *MaxPooling* ou de *AveragePooling* sont intercalées après chaque couche de convolution et ont pour but de réduire la dimension temporelle. Dans le cas du CNN avec *MaxPooling*, cette couche sélectionne la valeur maximale dans chaque fenêtre temporelle afin de réduire le nombre de valeurs dans la fenêtre et ainsi conserver les caractéristiques importantes du signal. Lorsque c'est une couche de *AveragePooling* qui est utilisée alors la moyenne des données de la fenêtre est calculée à la place de prendre la valeur maximale. En utilisant la moyenne, cela rend le réseau moins

sensible aux valeurs aberrants. Pour ces raisons, nous avons choisi de tester les deux configurations pour les modèles CNN, y compris pour les modèles d'encodeurs-décodeurs comprenant des CNN. De plus, pour les modèles d'encodeur-décodeur, la fenêtre de données de 150 a dû être modifiée par 152 car les trois convolutions successives nécessitent un nombre divisible par deux plusieurs fois d'affilées.

Les résultats sélectionnés à la Figure 28 parmi les différents modèles, résument tous ceux obtenus. Dans le cas des modèles encodeurs-décodeurs, les données censées être filtrées sont très bruitées et ne permettent pas d'obtenir des résultats à considérer. Concernant les autres modèles, le signal est lissé mais lorsque les fenêtres de données sont mises bout à bout, elles sont décalées les unes par rapport aux autres. Cela est certainement causé par le fait que lors de l'entrainement, une fenêtre glissante est utilisée. Ainsi, lorsque l'on trace les résultats pour chaque section du signal, ceux-ci sont décalés. Dans la partie suivante, nous allons donc présenter les résultats de modèles entrainés sans fenêtre glissante.

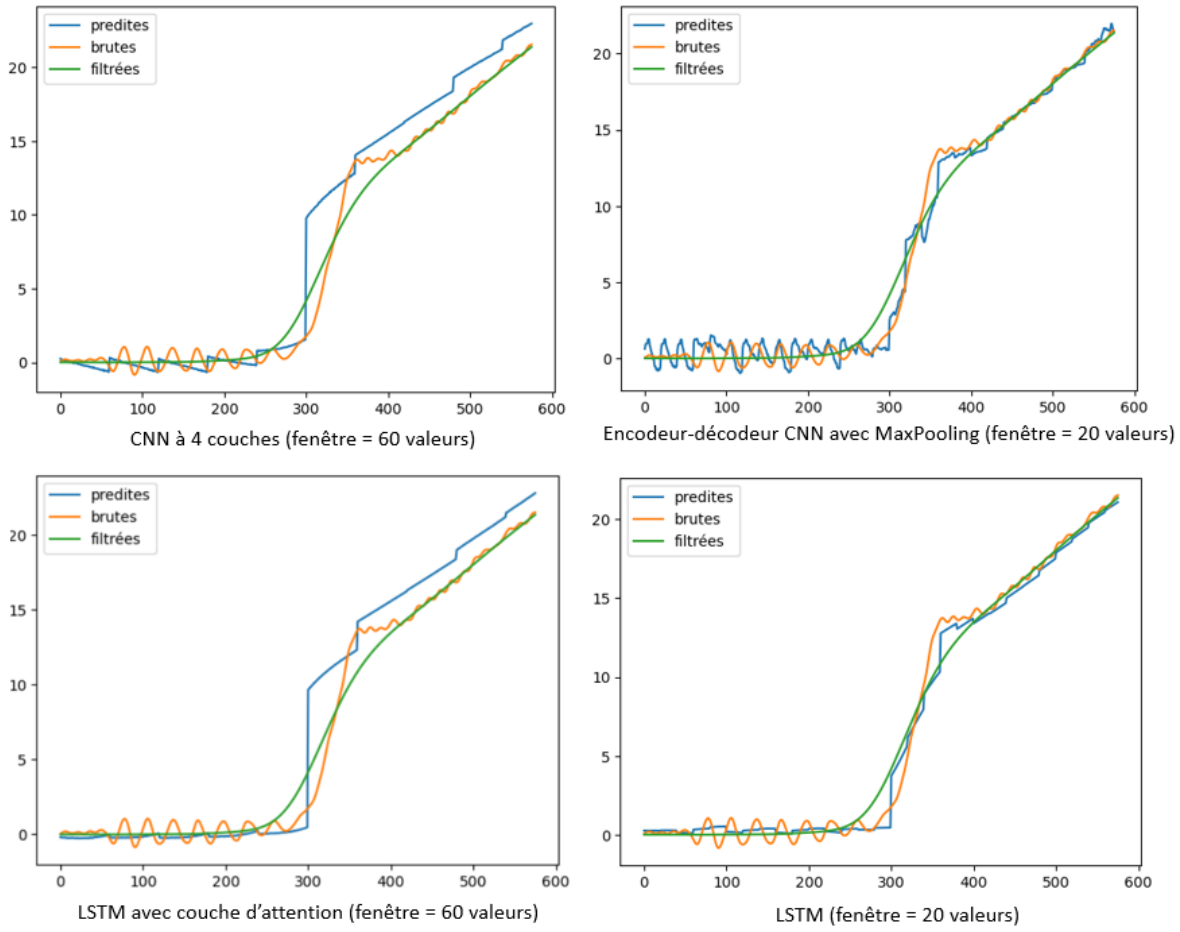



Figure 28. Quelques résultats des modèles d'apprentissage profond avec fenêtre glissante

- <u>Filtrage de séquences de données sans fenêtre glissante :</u>

- Phase de grand débit :

Étant donné les décalages présents entre chaque fenêtre de données dans les résultats précédents, il a donc été décidé de tester de ne pas utiliser de fenêtre glissante lors des entrainements des modèles. La phase de grand débit est donc toujours découpée en intervalles de la valeur du paramètre de la fenêtre. En entrée est fournie une séquence de poids bruts et en sortie la séquence de poids filtrés par l'API actuel et décalé de 115 données afin d'avoir le signal correspondant. Le Tableau 18 regroupe les différents modèles testés ainsi que leurs paramètres et des informations sur les temps d'entrainement de chaque époque pour les tailles

de fenêtres minimales et maximales. Dans ce cas-là, les paramètres des modèles sont fixes. Seules différentes valeurs de fenêtres sont testées.

Tableau 18. Paramétrage des modèles sans fenêtre

| **Modèle** | **Paramètres** | **Temps d'entrainements par époques (en secondes)** |
|---|---|---|
| CNN avec 2 couches de convolution et *MaxPooling* | Taux d'apprentissage : 0,0001<br>Taille des lots : 256<br>Nombre de filtres : 128<br>Taille du noyau : 5<br>Taille de mise en commun : 2 | Fenêtre de 20 données : 2<br>Fenêtre de 150 données : 0,45 |
| CNN avec 2 couches de convolution et *AveragePooling* | | Fenêtre de 20 données : 2<br>Fenêtre de 150 données : 0,45 |
| CNN avec 4 couches de convolution et *MaxPooling* | | Fenêtre de 20 données : 3<br>Fenêtre de 150 données : 1 |
| CNN avec 4 couches de convolution et *AveragePooling* | | Fenêtre de 20 données : 3<br>Fenêtre de 150 données : 1 |
| LSTM | Taux d'apprentissage : 0,001<br>Taille des lots : 256<br>Nombre de nœuds LSTM : 64 | Fenêtre de 20 données : 36<br>Fenêtre de 150 données : 28 |
| LSTM avec couche d'attention | | Fenêtre de 20 données : 38<br>Fenêtre de 150 données : 28 |
| Encodeur-décodeur LSTM | Taux d'apprentissage : 0,0001<br>Taille des lots : 256<br>Nombre de nœuds LSTM : 64 | Fenêtre de 20 données : 130<br>Fenêtre de 150 données : 94 |
| Encodeur-décodeur LSTM avec couche d'attention | | Fenêtre de 20 données : 126<br>Fenêtre de 150 données : 95 |
| Encodeur-décodeur CNN avec 3 couches de convolution et *MaxPooling* puis avec *UpSampling* dans le décodeur | Taux d'apprentissage : 0,0001<br>Taille des lots : 256<br>Nombre de filtres : 128<br>Taille du noyau : 5<br>Taille de mise en commun : 2 | Fenêtre de 20 données : 4<br>Fenêtre de 152 données : 1 |
| Encodeur-décodeur CNN avec 3 couches de convolution et *AveragePooling* puis avec *UpSampling* dans le décodeur | | Fenêtre de 20 données : 4<br>Fenêtre de 152 données : 1 |

| Modèle | | Temps d'entrainements par époques (en secondes) |
|---|---|---|
| Encodeur-décodeur CNN avec 3 couches de convolution et *MaxPooling* puis sans *UpSampling* dans le décodeur | Taux d'apprentissage : 0,0001<br>Taille des lots : 256<br>Nombre de filtres : 128<br>Taille du noyau : 5<br>Taille de mise en commun : 2 | Fenêtre de 20 données : 3<br>Fenêtre de 152 données : 1 |
| Encodeur-décodeur CNN avec 3 couches de convolution et *AveragePooling* puis sans *UpSampling* dans le décodeur | | Fenêtre de 20 données : 3<br>Fenêtre de 152 données : 1 |

Les modèles d'encodeurs-décodeurs avec CNN sont testées avec et sans *Upsampling*. Lors du déroulement de l'encodage dans les modèles d'encodeurs-décodeurs avec CNN, la dimensionnalité est réduite en utilisant des couches de *MaxPooling* et de *AveragePooling*. Cette réduction de dimensionnalité permet de conserver les caractéristiques importantes et de diminuer la complexité du modèle. Ainsi, lors du décodage, il est possible d'utiliser ou non de l'*Upsampling* selon les besoins. Dans le cas où nous n'en utilisons pas, la version réduite par l'encodeur sera gardée dans la sortie du modèle. En utilisant de l'*Upsampling*, nous pourrons récupérer une version plus détaillée. Ainsi, afin de ne pas négliger l'impact de la présence ou non d'une couche d'*Upsampling*, nous avons choisi de tester les deux configurations.

La Figure 29 présente les quatre meilleurs résultats obtenus parmi les différents modèles et tailles de fenêtres testés.

Nous pouvons donc observer que les signaux prédits suivent les signaux filtrés par l'API. Les décalages observés lors de l'utilisation d'une fenêtre glissante lors des entrainements sont très réduits, voire absents dans le cas du modèle CNN avec deux couches de convolution intercalées de *MaxPooling* et pour une fenêtre de 100 données. Cependant, les signaux sont bruités, même s'il s'agit de faibles oscillations. Ainsi, ces résultats sont

meilleurs que ceux des précédents modèles malgré un nombre de données d'entrainement moins important du fait de l'absence de fenêtre glissante.

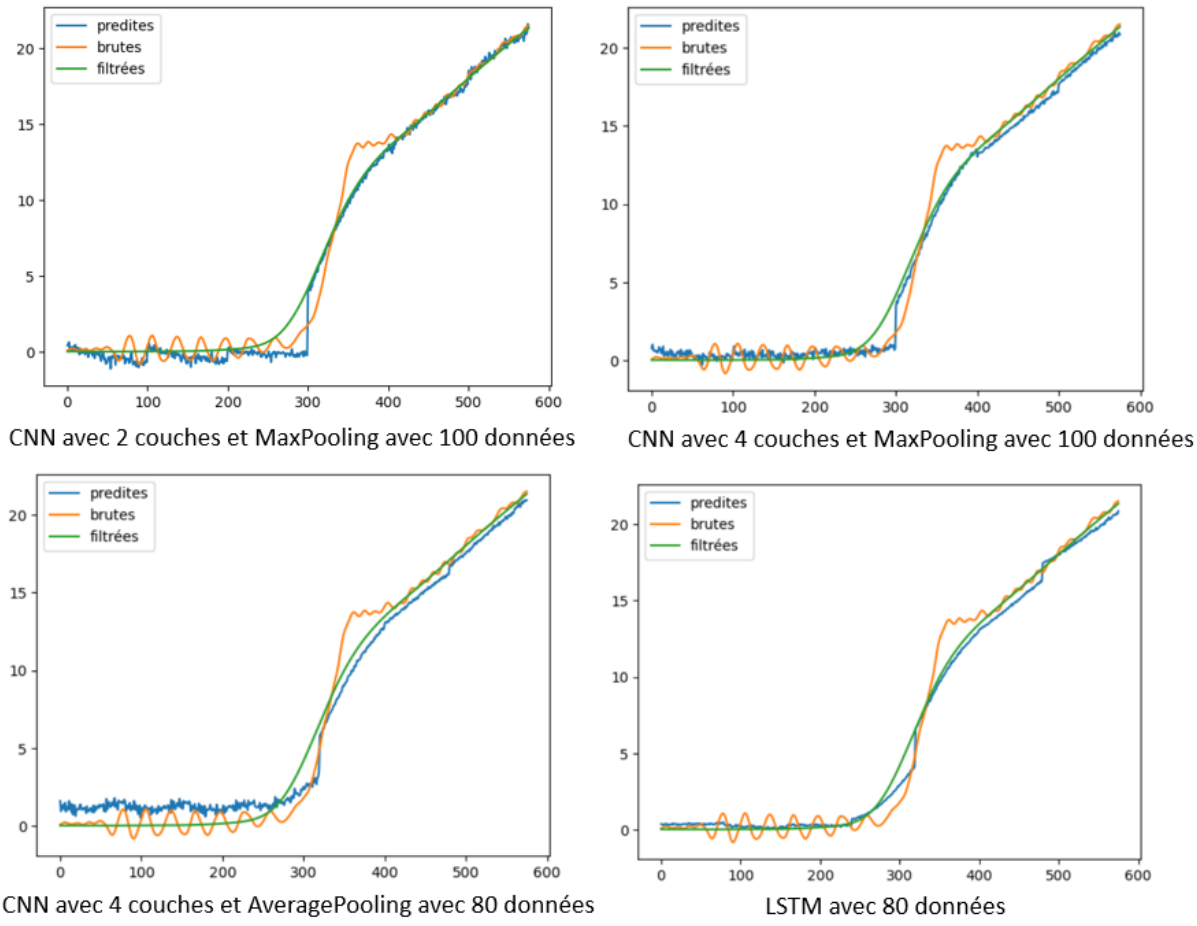
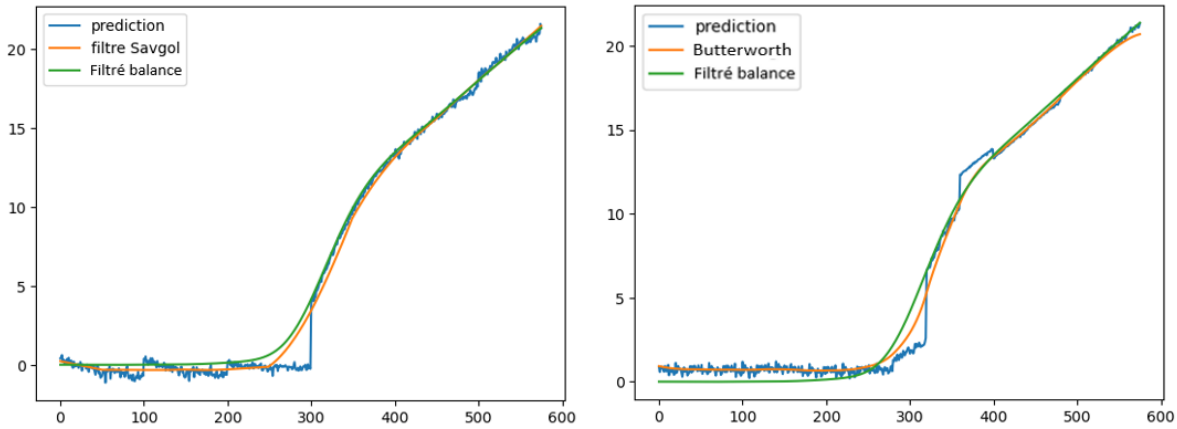



Figure 29. Résultats de filtrage avec apprentissage profond pour la phase de grand débit

Afin de compléter le filtrage réalisé par les modèles d'apprentissage profond et supprimer ce faible bruit présent dans les prédictions, nous avons voulu tester de combiner ces résultats obtenus avec des filtres numériques. Il s'agit des mêmes filtres numériques que ceux employés précédemment sur les données de poids bruts : filtre de Savitzky-Golay, filtre de Butterworth, filtre de moyenne mobile, filtre de moyenne mobile exponentielle et filtre de médiane. La Figure 30 montre deux exemples de bons résultats où un modèle d'apprentissage profond est combiné à un filtre numérique.




CNN avec 2couches et MaxPooling avec 100 données puis filtré avec filtre de Savitzky-Golay d'ordre 1

CNN avec 2 couches et MaxPooling avec 40 données et filtre de Butterworth (ordre :1 ; fréquence d'échantillonnage :627 ; fréquence de coupure :5 )

Figure 30. Résultats obtenus en combinant les résultats précédents avec des filtres numériques

Dans le tracé de gauche, les résultats obtenus par le filtrage à partir d'apprentissage profond suivait assez fidèlement à partir de 300 millisecondes le signal filtré par la balance. Ainsi, en le combinant avec le filtre de Savitzky-Golay, nous obtenons un signal filtré très proche de celui de la balance actuelle. Il en est de même pour le tracé de droite qui permet d'obtenir un filtrage lorsque le filtre de Butterworth est appliqué, alors qu'initialement les résultats fournis par le modèle CNN de droite n'étaient pas aussi bons que les autres. Le désavantage d'utiliser cette combinaison de méthodes de filtrage est que cela nécessite un temps plus long de calcul, car deux opérations doivent être effectuées, ce qui peut limiter son efficacité lors d'une utilisation en temps réel.

Malgré cette limite d'efficacité, les résultats obtenus pour la phase de grand débit ont montré un filtrage très proche de celui de l'API actuel. Par conséquent, nous avons donc décidé de tester cette même approche sur les autres phases du cycle, afin de déterminer si nous pouvons obtenir des résultats similaires au poids filtré de l'API dans ces phases également.

- Phase de changement de débit :

Les mêmes modèles d'apprentissage profond que ceux pour la phase de grand débit sont utilisés pour cette phase. Le principe du découpage en fenêtres et le non glissement de celles-ci est également conservé. Les entrées sont donc des fenêtres de poids bruts et les sorties sont des fenêtres de poids filtrés toujours décalées de 115 données pour les mêmes raisons.

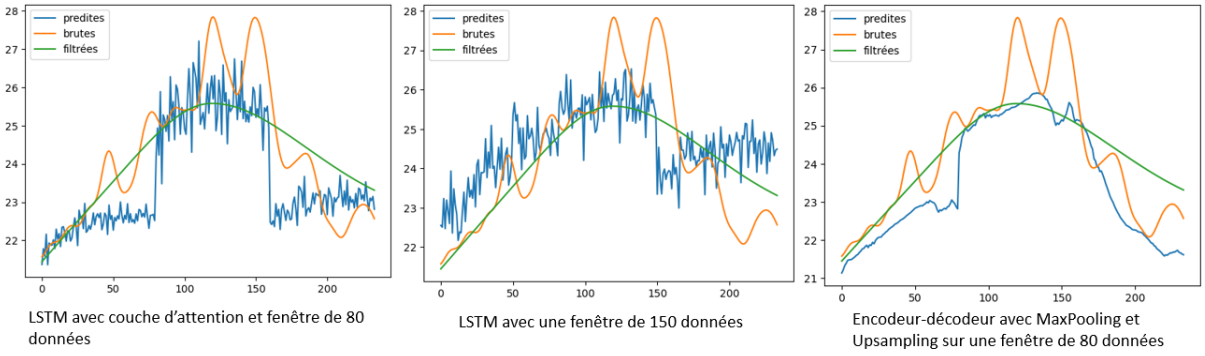



Figure 31. Résultats de filtrage avec l'apprentissage profond pour la phase de changement de débit

Ces trois résultats ont été sélectionnés parmi tous les résultats obtenus car ce sont ceux qui étaient les plus proches du signal filtré par l'API. Dans les deux premiers résultats affichés qui concernent un modèle LSTM et un modèle LSTM avec couche d'attention, les résultats suivent le tracé du filtre, notamment au niveau du point maximal atteint. Cependant un décalage est présent entre chaque fenêtre de données et le signal est très bruité. Les différents modèles ont été testés avec différentes tailles de fenêtres : 20, 40, 60, 80, 100, 150. Ainsi, les meilleurs résultats ont été obtenus avec des tailles de fenêtres parmi les plus grandes proposées alors que cette phase dure moins longtemps que la phase de grand débit. Concernant le résultat affiché pour l'encodeur-décodeur, le signal obtenu ne compte pas de bruit comme les deux autres et tend à atténuer les oscillations du signal de poids brut.

Nous avons également essayé de combiner ces résultats avec les filtres numériques utilisés précédemment. La Figure 32 montre un exemple d'application du filtre de Butterworth aux résultats obtenus par le modèle LSTM avec couche d'attention.

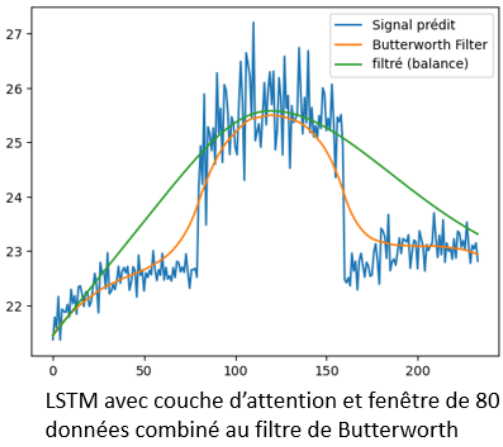



Figure 32. Résultats en ajoutant le filtre de Butterworth (signal orange) sur les résultats d'apprentissage profond (signal bleu)

L'utilisation d'un filtre numérique en supplément des résultats obtenus par le modèle d'apprentissage profond permet de lisser ou d'accentuer et atténuer certaines fréquences dans les données prédites qui étaient très bruitées et de réduire les écarts liés aux décalages entre les phases. Le point important dans cette phase est le point maximal car lorsque l'API actuel s'aperçoit que l'extrema local a été atteint, il sait que le changement de débit a bien été effectué. Ainsi, le fait que le nouveau filtrage soit également sous forme de cloche et que l'extrema local soit atteint est un bon résultat de filtrage même si celui-ci n'est pas identique à celui de l'API actuel qui nous sert de référence.

- Phases de petit débit et de fermeture de l'écoulement :

Cette section regroupe les résultats de filtrage avec des modèles d'apprentissage profond pour les phases de petit débit et de fermeture de la trémie d'alimentation du produit. Comme pour les deux autres phases présentées précédemment, les mêmes modèles ont été testés sur ces deux phases avec des paramètres fixes sauf la valeur de la fenêtre de données.

Un décalage entre les données d'entrées de poids bruts et de sorties de poids filtrés est toujours appliqué, sauf que pour ces deux phases le décalage est plus important que pour les deux premières. Nous appliquons donc un décalage de 180 données.

La Figure 33 ci-dessous regroupe les résultats de trois modèles parmi tous ceux testés pour la phase de petit débit.

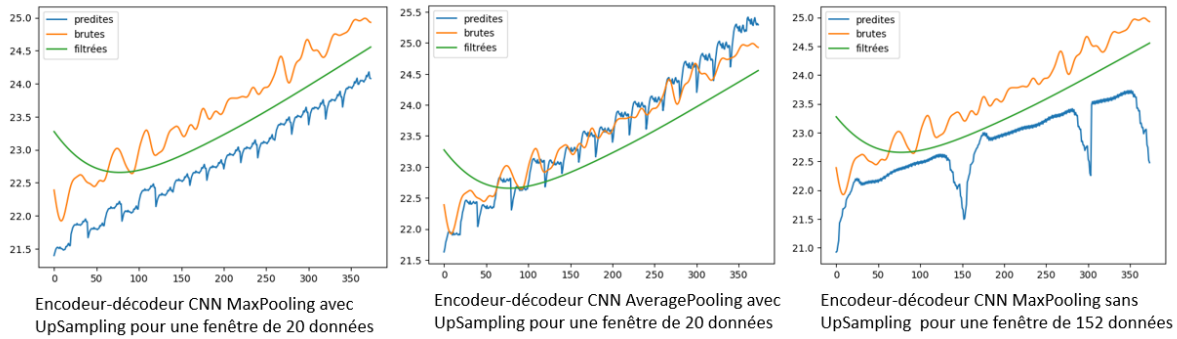



Figure 33. Résultats pour la phase de petit débit

La Figure 34 regroupe quelques résultats de la phase de fermeture :

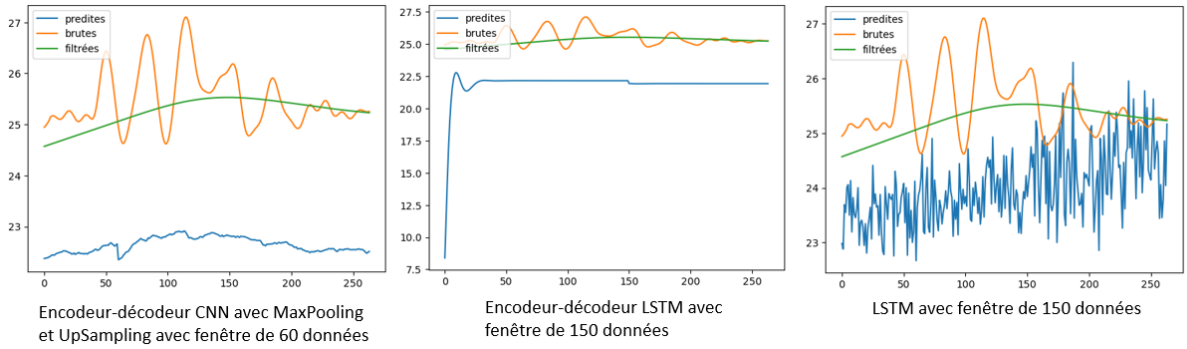



Figure 34. Résultats pour la phase de fermeture

Ainsi, il est possible d'observer que ces modèles de filtrage appliqués sur ces deux phrases, ne permettent pas d'obtenir de bons résultats. En effet, dans la majorité de ces résultats, les signaux filtrés sont inférieurs aux autres signaux de poids bruts et de poids filtrés par l'API, ce qui si on les appliquait en conditions réelles amènerait à un poids ensaché bien

supérieur à celui attendu. De plus, ils présentent énormément d'oscillations et de fluctuations. L'objectif de lissage n'est pas donc pas rempli pour ces applications.

### 5.2.4 Discussion sur la première étape de l'approche

Pour cette première étape, plusieurs techniques ont donc été testées pour trouver une méthode de filtrage efficace, incluant des filtres numériques, de l'apprentissage automatique et de l'apprentissage profond. Les résultats ont montré que les filtres numériques, en particulier la méthode de conception des coefficients du filtre de Butterworth, ont donné de bons résultats. Les modèles d'apprentissage automatique (SVR et XGBoost) ont montré également des résultats intéressants. En revanche, les résultats obtenus avec l'apprentissage profond n'étaient pas concluants.

La cause principale de la mauvaise performance de l'apprentissage profond semble être la quantité de données d'entrainement, qui n'est pas suffisante. Cependant, même avec un volume de données plus important, ce type de modèle n'aurait pas été idéal pour notre application. En effet, la collecte de données nécessaire pour chaque changement de poids ou de matériaux est très contraignante, tout comme l'entrainement de nouveaux modèles. De plus, la variabilité des données futures et la difficulté de garantir des résultats précis en font une méthode peu adaptée pour un contexte industriel où l'efficacité est primordiale. En comparaison, les filtres numériques nécessitent moins de réglages et permettent d'obtenir des résultats robustes aux variations du signal. L'absence de contraintes liées à la gestion de grandes quantités de données et l'entrainement de nouveaux modèles en fait une solution simple et adaptée à notre utilisation.

## 5.3 RÉSULTATS DE L'APPRENTISSAGE PAR RENFORCEMENT

### 5.3.1 Description du l'environnement et de son interaction avec l'agent

En raison de l'impossibilité de déployer l'agent directement sur le système réel, nous avons recréé un environnement simulant le fonctionnement de la balance industrielle étudiée. L'implémentation suit notre spécification SysML, en se basant notamment sur les diagrammes SysML présentés dans le troisième chapitre, et en particulier le diagramme d'activité « contrôler le pesage ».

Afin d'imiter le fonctionnement décrit dans le diagramme d'activité, une mémoire tampon a été mise en place pour collecter des échantillons de données transmis par l'environnement, les filtrer et les retransmettre à l'agent pour qu'il puisse prendre une décision. L'environnement créé se base sur les cycles de pesage enregistrés lors des prises de données. Pour recréer un signal de poids ressemblant au signal brut et contenant autant de variabilité que les cycles réels, nous nous sommes basés sur le calcul des débits. À chaque nouvelle pesée lancée par l'agent, ces calculs sont réalisés en amont du lancement de la pesée sélectionnée. Étant donné la grande variabilité de la perception de l'écoulement par la balance, le calcul des débits est réalisé sur plusieurs portions du signal. Chaque phase du cycle est découpée en intervalles de 25 pas de temps, où le débit est calculé pour chacun de ces intervalles. La taille des intervalles a été choisie de manière à ne pas être trop grande, afin de limiter la perte d'informations. Le débit nous permet donc de connaitre la masse ajoutée à chaque pas de temps dans le sac. Le débit n'étant pas constant sur tout le cycle, nous tenons compte de ces variations par palier de 25 données. Ainsi, nous utilisons cette information pour extrapoler les données. Pour cela, à chaque pas de temps la valeur de débit correspondant est ajoutée à la masse déjà présente dans le sac.

Ainsi, au début de la pesée générée, l'environnement renvoie les valeurs telles qu'elles ont été enregistrées lors des prises de données. Ensuite, lorsque l'agent choisit de réaliser une action différente du cycle de référence, les données sont extrapolées en s'appuyant sur les

débits calculés correspondants. Si l'agent décide d'interrompre ou de continuer une phase et que cette décision diffère de ce qu'il se passait dans l'enregistrement, alors des données vont être générées en fonction de l'avancement dans la phase. Par exemple, s'il choisit de rester dans la phase de grand débit plus longtemps, l'environnement récupère le débit pour le dernier intervalle de 25 données de cette phase et l'utilise pour générer de nouvelles données. À l'inverse, si l'agent décide de passer en petit débit plus tôt que prévu, l'environnement prend la valeur de poids atteinte juste avant le changement de phase et génère un signal à partir de cette valeur en appliquant les variations de poids correspondantes. Il utilise alors la valeur du dernier poids atteint dans la phase actuelle et extrapole en appliquant le débit de la première portion de la phase suivante, c'est-à-dire la phase de changement de débit. Après avoir généré 25 données, il passe aux informations de débits de la deuxième portion du signal initial pour cette phase. Ainsi, la variation de poids propre au premier intervalle est appliquée pour générer 25 données, puis les 25 données suivantes sont générées à partir des variations de poids du second intervalle de la phase, et ainsi de suite. L'extrapolation des données se poursuit de cette manière pour le reste du cycle de pesage, car les données enregistrées ne correspondent plus aux décisions prises par l'agent. Ces données sont donc générées au fur et à mesure afin d'imiter un cycle de pesage en cours d'exécution.

Pour les phases de changement de débit et de fermeture, l'agent ne peut pas choisir de raccourcir ou de prolonger leurs durées. En effet, il s'agit de phases dans lesquelles une action de mouvement mécanique est réalisée par le servomoteur de la trémie, leur durée ne peut donc pas dépendre des décisions de l'agent et il est contraint de respecter le temps que cette étape a duré lors de l'enregistrement.

Concernant la phase de stabilisation, il doit respecter un temps minimum et décide de l'arrêter lorsque les oscillations de poids se sont réduites (Voir Annexe F).

### 5.3.2 Les modèles

Le modèle d'apprentissage par renforcement que nous avons choisi est un modèle DQN avec différence temporelle, spécialement adapté pour des environnements complexes et continus. Notre environnement est considéré comme complexe en raison de la multitude d'états possibles. Bien que les actions soient limitées, l'évolution du poids est très variable. L'agent peut décider soit de rester dans la phase actuelle, soit de passer à la phase suivante. Nos différents modèles testés comprennent différentes configurations d'états, pour trouver une configuration efficace :

- Un état : Ce modèle utilise uniquement le dernier poids atteint comme état.
- Quatre états : Il prend en compte le dernier poids atteint, le poids atteint lors de l'étape précédente, le poids rempli en grand débit et l'action prise.
- Sept états : Cette version utilise le dernier poids atteint, le poids rempli en grand débit, la moyenne du signal filtré, la médiane du signal filtré, la variance du signal filtré, ainsi que les valeurs minimale et maximale atteintes par le signal filtré.

Nous avons également testé deux types de modèles différents concernant l'enchaînement des actions. Dans le modèle contraint, l'enchaînement des actions est prédéterminé pour respecter dès le départ le cycle de pesage de la balance. Ce cycle suit la séquence suivante : grand débit, changement, petit débit, fermeture et stabilisation. Ce modèle impose donc un ordre fixe des étapes. En revanche, un modèle autonome a été également développé. Dans ce modèle, l'agent doit apprendre à enchaîner les actions de manière autonome. Par exemple, durant la phase de petit débit, l'agent peut décider de retourner en grand débit. Ces changements peuvent être effectués autant de fois que nécessaire, la seule contrainte étant de respecter le temps de changement de phase. Si l'agent réalise une séquence d'étapes différente de celle de l'API, il est pénalisé. Ainsi, il finit par apprendre lui-même l'enchainement optimal des étapes du cycle. Cependant, pour optimiser l'apprentissage et réduire le nombre de pesées nécessaires à l'entrainement, nous nous concentrons principalement sur les modèles avec des séquences d'actions prédéfinies. Pour

évaluer nos différentes stratégies et vérifier que les modèles généralisent bien, les données sont divisées en trois ensembles : entrainement, développement et test. Pour des applications standard d'apprentissage par renforcement, la performance est directement ajustée en fonction des observations faites sur les résultats. Dans notre cas, étant donné qu'il s'agit d'une application hors ligne sur un ensemble de données fixes, il est important d'utiliser plusieurs ensembles.

L'ensemble d'entrainement est le plus conséquent, contenant 80% des données, soit 8000 pesées. L'ensemble de développement (ou de validation) est utilisé pour ajuster les hyperparamètres et prendre des décisions sur les modèles à sélectionner. Il représente 10% des données, soit 1000 pesées. L'ensemble de test, qui représente également 10% des données, est utilisé pour évaluer la performance finale de l'agent.

### 5.3.3 Les hyperparamètres

Chacun de ces hyperparamètres constituent le modèle et leurs valeurs doivent être ajustées en fonction des performances de l'apprentissage et de la variabilité des états observés :

- Taille de lot (*batch size*) : Comme décrit pour les modèles d'apprentissage supervisés, il correspond au nombre d'échantillons utilisés à chaque mise à jour du modèle. Une valeur élevée peut améliorer la convergence du modèle mais va nécessiter plus de ressources de calcul, tandis qu'une valeur plus faible permet un entrainement plus rapide mais avec une stabilité potentiellement réduite. Les valeurs typiques varient entre 32 et 128 et peuvent aller jusqu'à 256. La valeur de 64 a d'abord été choisie comme étant un bon compromis entre stabilité et vitesse de calcul afin d'avoir une vitesse d'entrainement raisonnable.

- Taux d'apprentissage (*learning rate*) : Cet hyperparamètre définit la taille des ajustements des poids du réseau de neurones à chaque itération. Un taux d'apprentissage de 0,001 est couramment utilisé car il permet à l'agent de s'adapter rapidement aux nouvelles informations tout en minimisant les risques d'oscillations et d'instabilité. Une valeur trop élevée peut rendre le modèle instable et l'empêcher de converger, alors qu'une valeur trop faible peut ralentir l'apprentissage.

- Fréquence de mise à jour du réseau cible (*target update frequence*) : Ce paramètre détermine la fréquence de mise à jour du réseau cible par rapport au réseau principal. Le réseau principal est celui qui est entrainé et qui apprend à estimer les valeurs Q. Ses poids sont mis à jour à chaque étape d'entrainement. Le réseau cible est une copie du réseau principal, mais ses poids ne sont mis à jour qu'à des intervalles spécifiques. Cela permet d'obtenir des estimations plus stables des futures valeurs Q et d'aider le réseau à converger. Étant donné que notre environnement suit une progression continue, la valeur 100 a d'abord été définie pour que le réseau cible soit mis à jour plusieurs fois par épisode et qu'il s'ajuste aux changements de poids.

- Nombre minimum de pas de temps d'exploration : Il détermine le nombre de pas de temps que l'agent passera à explorer l'environnement avant de commencer à exploiter les connaissances acquises. Cela permet de s'assurer que l'agent explore suffisamment l'espace d'état-action et qu'il ne se retrouve pas bloqué dans une stratégie sous-optimales. Pour notre application, la valeur choisie est de 150 000, ce qui lui permet d'explorer entre 100 et 150 pesées avant de commencer à appliquer la politique apprise.

- Taux de déclin (*epsilon decay*) : Le taux de déclin permet de réguler la transition entre l'exploration et l'exploitation des connaissances de l'agent. Un taux lent indique que l'exploration de nouvelles actions continuera pendant un grand nombre de pas de

temps avant de passer à l'exploitation de la politique. Néanmoins, il y aura toujours une possibilité d'exploration, car cette décision est basée sur une valeur aléatoire générée entre 0 et 1. Si cette valeur est inférieure à epsilon, alors l'agent explore en choisissant une action aléatoire. S'il est supérieur ou égal, alors l'agent fait de l'exploitation et choisit la meilleure action connue d'après ses expériences passées. Le choix du taux de déclin appliqué à epsilon s'appuie sur la formule suivante :

$$\varepsilon_{final} = \varepsilon_{initial} * decay^{N} \quad (2)$$

Où $\varepsilon_{final}$ est la valeur cible de l'epsilon (0,1 dans notre cas) avec N pas de temps, $\varepsilon_{initial}$ est la valeur initiale de l'epsilon et vaut 1,0, *decay* est le taux de déclin à déterminer. Ainsi pour calculer le taux d'exploration, nous avons :

$$decay = (\frac{\varepsilon_{final}}{\varepsilon_{initial}})^{\frac{1}{N}} \quad (3)$$

Étant donné que le nombre minimum de pas de temps d'exploration est assez conséquent, une valeur du taux qui décline assez rapidement n'empêchera pas l'exploration.

- Maximum d'épisodes : Cette valeur correspond à la quantité totale d'épisodes que l'agent peut réaliser lors de l'entrainement. Dans notre cas, il s'agit de la taille de l'ensemble d'entrainement, soit 8000 épisodes.

- Taille de la mémoire tampon : Lors de l'interaction entre l'agent et son environnement, chaque expérience est enregistrée dans la mémoire tampon. Lors de l'apprentissage, le modèle n'utilise pas uniquement les expériences récentes, mais il échantillonne de façon aléatoire des expériences stockées dans la mémoire tampon. Le modèle peut ainsi apprendre en utilisant des expériences provenant de différents

moments du passé. Cela permet de stabiliser et améliorer l'apprentissage en évitant les biais causés par la corrélation entre les échantillons consécutifs. Pour définir la valeur de ce paramètre, un compromis doit être fait entre un entrainement plus long dû à l'utilisation de la mémoire et un apprentissage plus riche. Pour trouver la valeur, il suffit de multiplier le nombre d'étapes par épisode avec le nombre d'épisodes que l'on souhaite stocker dans la mémoire. Étant donné que nos épisodes durent en moyenne entre 1500 et 2000 pas de temps, en prenant une valeur de 100 000, nous pouvons stocker plus d'une cinquantaine d'épisodes. Cette valeur est faible mais cela permet à l'agent de s'adapter plus rapidement aux changements dans l'environnement, ce qui est utile dans notre cas où l'environnement est variable.

- Taux d'actualisation des récompenses (*discount factor*) : Ce paramètre détermine l'importance des récompenses futures par rapport aux réponses immédiates et est également appelé facteur de remise. Sa valeur est comprise entre 0 et 1. Si le facteur vaut 1, alors l'agent favorise les récompenses futures autant que les récompenses immédiates. Si la valeur est de 0, l'agent ne prend en compte que les récompenses immédiates. Un facteur de remise à 0,9 permet à l'agent de réagir aux récompenses immédiates tout en tenant compte des récompenses futures.

### 5.3.4 Le système de récompenses

Lors de la conception du système de récompenses, plusieurs approches et types de récompenses sont possibles. Les récompenses peuvent être classées en deux grandes catégories, les récompenses finales et les intermédiaires.

Les récompenses finales sont perçues à la fin de l'épisode, ce qui permet à l'agent de se concentrer sur l'atteinte de l'objectif final en évaluant les actions suivant leur contribution à son accomplissement. Cela facilite l'évaluation des performances. Cependant, l'agent ne reçoit aucun retour durant l'épisode, ce qui peut ralentir l'apprentissage car il n'a pas

d'indications sur les actions intermédiaires qu'il prend. Les récompenses intermédiaires sont obtenues durant l'épisode en fonction des décisions prises par l'agent et lui fournissent des retours continus, ce qui peut accélérer l'apprentissage et lui permettre d'ajuster ses actions en temps réels. Cependant, cela peut l'empêcher de distinguer les actions permettant de répondre à l'objectif final car il pourrait adopter un comportement où il optimise principalement les récompenses intermédiaires et néglige les finales. Si les récompenses intermédiaires ne sont pas bien conçues, elles peuvent encourager des comportements indésirables de l'agent. La conception du système de récompenses peut nécessiter plusieurs essais et erreurs avant de trouver la bonne méthode. Il est également possible de créer un système de récompenses avec des récompenses intermédiaires et finales qui soient déterminées différemment. De plus, les récompenses intermédiaires peuvent être cumulatives ou non-cumulatives. Dans le cas des récompenses cumulatives, les récompenses perçues sont additionnées durant tout l'épisode. Cette méthode permet d'encourager l'agent à maintenir des récompenses constantes pour tout l'épisode, mais cela peut amener trop de rétroaction pour l'agent et l'empêcher de distinguer quelles sont les meilleures actions. Pour les récompenses non cumulatives, chaque récompense perçue est indépendante et ne s'additionne pas avec la précédente. La rétroaction des récompenses est donc plus claire pour l'agent mais elles représentent moins les performances de tout l'épisode.

Lorsque le système étudié comporte plusieurs critères à prendre en compte, il est nécessaire de choisir une méthode appropriée pour combiner ces critères dans la formule de récompense. Deux approches souvent utilisées sont l'approche de récompenses additives et celle de récompenses multiplicatives.

Les récompenses additives consistent à sommer les différentes composantes de la récompense. Chaque composante contribue de manière indépendante à la récompense totale, permettant ainsi d'ajuster l'importance de chaque critère. Cependant, l'inconvénient est que l'agent peut apprendre à optimiser uniquement la composante qui lui rapporte le plus, au détriment des autres. Pour pallier ce problème, il est possible d'ajouter des facteurs de pondération à chaque élément additionné, et ainsi accorder plus d'importance à certains

critères. Trouver la pondération optimale pour maximiser l'apprentissage est toutefois complexe, surtout dans des environnements continus ayant de nombreux états, comme dans notre cas où il y a autant d'états que de poids possibles jusqu'à atteindre le poids cible et au-delà. Les valeurs de pondérations des critères doivent être choisies de manière à encourager l'agent à prendre des décisions en vue de l'objectif final, tout en évitant un comportement qui favorise les récompenses à court terme. Cependant, une pondération efficace dans une situation peut ne pas l'être dans une autre. Un mauvais choix de pondération peut entrainer à un surapprentissage, où l'agent performe bien lors de l'entrainement et des tests similaires, mais échoue à se généraliser à des changements mêmes minimes dans l'environnement. Ainsi, bien que les récompenses additives avec pondération soient faciles à mettre en place, leur calibrage est un processus complexe qui peut nécessiter plusieurs itérations.

Les récompenses multiplicatives consistent à multiplier les différents composants, ce qui garantit que tous les critères soient optimisés pour maximiser la récompense totale. Une mauvaise performance d'un critère affecte directement la récompense totale, incitant l'agent à maintenir un équilibre entre les différents critères. Par conséquent, la valeur totale est très sensible aux valeurs de chaque composant, car ils sont interdépendants. Par exemple, une valeur nulle annule toute la récompense, ce qui empêche l'agent de savoir s'il excelle dans un autre critère, créant ainsi des incohérences. De plus, la valeur d'un critère peut grandement augmenter ou diminuer la valeur totale. L'agent doit donc optimiser tous les critères simultanément, ce qui ne lui permet pas de s'adapter à une situation où un critère est temporairement moins important. Dans un environnement où les valeurs ont tendance à beaucoup osciller, les récompenses peuvent présenter des fluctuations importantes, entrainant un apprentissage instable. Ainsi, un mauvais choix des différents critères à multiplier peut entrainer un apprentissage inefficace.

Ainsi, l'ajustement du système de récompense nécessite plusieurs tentatives et ajustements. Dans notre cas où l'agent doit être récompensé pour sa précision et sa vitesse, plusieurs options sont possibles. Dans un premier temps, nous avons fait le choix d'utiliser une récompense finale uniquement, car nous avons un système continu avec beaucoup d'états

et que notre principal enjeu est de faire en sorte qu'il respecte le poids à atteindre. En effet, trouver un système optimal de récompense, composé de récompenses intermédiaires, nous aurait demandé trop d'itérations pour parvenir à un résultat incertain. Étant limités en temps et en ressources computationnelles, nous avons privilégié un système de récompenses finales. Ainsi lorsque l'agent atteint l'intervalle de poids souhaité, il perçoit une récompense finale non nulle, qui est complétée par une récompense liée au temps. Le système de récompense finale mis en place pour notre application est le suivant :

La récompense attribuée pour la précision est donnée par :

$$récompense\ précision\ finale = 1{,}0 - k_7 * |\ poids\ actuel - poids\ cible\ |\quad (4)$$

Où la valeur de poids cible est 25 et nous avons fixé la pénalité $k_7$ à 0,5 pour éviter qu'elle soit trop punitive ou trop indulgente, afin que l'agent ne réagisse pas de façon excessive aux petites erreurs. La valeur obtenue avec cette récompense est plus importante que la deuxième composante de la récompense finale, afin de garantir que l'agent respecte le critère de précision. La récompense liée à la durée du cycle complet et qui vient compléter la précision est :

$$récompense\ temps\ ajusté = \frac{poids\ actuel}{pas\ de\ temps\ actuel + \varepsilon} \quad (5)$$

Où $\varepsilon$ vaut 0,1 afin d'éviter une division nulle. Cette récompense encourage l'agent à atteindre le poids cible tout en minimisant le temps pour le faire. Étant donné que cette récompense est finale, la valeur du poids actuel sera toujours très proche de 25 mais le temps mis pour l'atteindre permettra à l'agent d'avoir un retour sur l'impact de la durée du cycle.

$$récompense\ finale = récompense\ précision\ finale + récompense\ temps\ ajusté \quad (6)$$

Bien que la récompense liée au temps puisse sembler petite par rapport à la récompense de précision, elles sont complémentaires. En effet, se concentrer uniquement sur la précision pourrait inciter l'agent à prendre plus de temps pour être précis. La récompense liée au temps apporte ainsi l'information nécessaire pour qu'il ne néglige pas la rapidité. De plus, pour

compléter ce système de récompense, si la précision finale est très mauvaise, alors la valeur de récompense finale sera directement définie à -1, afin que la mauvaise précision ne puisse pas être compensée par la récompense de temps. Il en va de même pour les modèles où l'agent est autonome dans le choix de l'enchaînement des phases du signal. Ainsi, si l'enchainement n'est pas correct, la récompense finale sera également de -1, ce qui permettra d'apprendre le bon séquencement.

### 5.3.5 Les résultats

Dans un premier temps, nous avons entrainé quatre modèles obtenant des récompenses finales lorsque le poids final se situe entre 24,5 et 25,5 kg. Les modèles entrainés sont les trois types de modèles avec des états différents (un état, quatre états, sept états) et un modèle à un état où les données de l'état ont été normalisées. La normalisation est réalisée entre -1 et 1 avec la formule suivante :

$$poids\ normalisé = 2 * \left(\frac{poids - x_{min}}{x_{max} - x_{min}}\right) - 1 \quad (7)$$

Avec $x_{min}$ = -20 et $x_{max}$ = 30, qui sont les valeurs minimale et maximale que peut atteindre le signal de poids, notamment lors de mouvements des portes de pesage.

La normalisation des données peut aider les modèles à mieux se stabiliser et converger, nous avons donc essayer de voir si cela pouvait faire une différence sur les résultats. Pour ces modèles les valeurs des hyperparamètres sont les mêmes et sont les suivants :

- Taille de lot : 64
- Taux d'apprentissage : 0,001
- Fréquence de mise à jour du réseau cible : 100
- Nombre minimum de pas de temps d'exploration : 150 000
- Taux de déclin : 0,99885

- Taille de la mémoire tampon : 100 000
- Taux d'actualisation des récompenses : 0,9

Ces modèles ont été entrainés sur 8000 pesées et validés sur 100 pesées.

Le Tableau 19 affiche la proportion de pesées dont le poids final est compris dans l'intervalle 24,5 et 25,5, c'est-à-dire le nombre de pesées répondant à ce critère divisé par le nombre total de pesées entrainées ou testées.

Tableau 19. Résultats pour l'intervalle de récompense 24,5-25,5

| | **Proportion de cycles ayant un poids final dans l'intervalle de récompense pour l'entrainement (en pourcentage)** | **Proportion de cycles ayant un poids final dans l'intervalle de récompense pour la validation (en pourcentage)** |
|---|---|---|
| Un état | 1,32 | 22,52 |
| Un état normalisé | 1,52 | 0,00 |
| Quatre états | 1,43 | 18,18 |
| Sept états | 1,39 | 26,00 |

Ainsi, les résultats obtenus pour la validation réalisé sur 100 pesées, sont du même ordre de grandeur, sauf pour le modèle normalisé. Lorsque nous analysons les résultats, nous traçons le graphique de la Figure 35, afin de se rendre compte des durées et récompenses perçues.

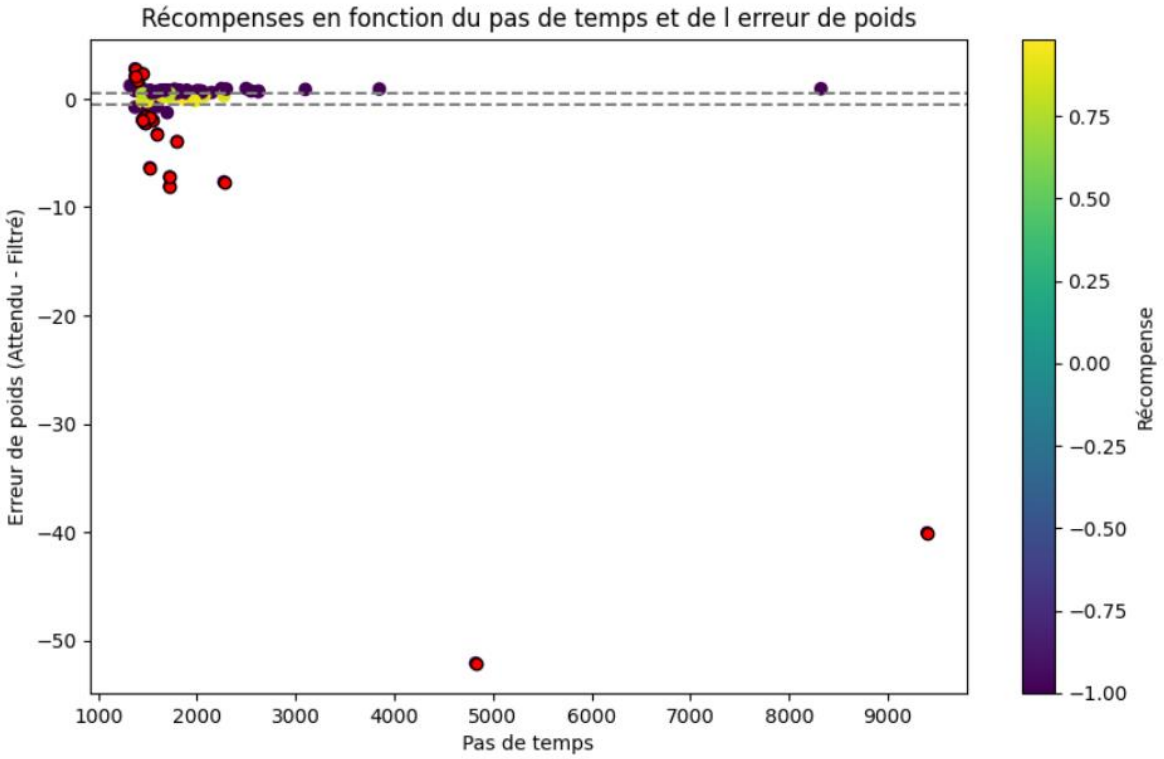



Figure 35. Résultats de validation pour le modèle à un état et récompense entre 24,5 et 25,5

Les numéros affichés en bleu correspondent aux numéros de cycle afin que nous puissions aller regarder le détail du déroulement de la pesée dans le fichier Excel correspondant. L'erreur de poids correspond à la différence entre le poids cible et le poids atteint par l'agent. Les points rouges correspondent aux pesées dont le poids final est à l'extérieur de l'intervalle 23,5 et 26,5. L'intervalle 24,5 et 25,5 est délimité par les traits pointillés gris. Ainsi, grâce à ce graphe nous observons que beaucoup de pesées sont dans l'intervalle 23,5 et 26,5 (points violets) mais n'atteignent pas l'intervalle 24,5 et 25,5 (points jaunes). Afin de faire comprendre à l'agent qu'il se rapproche de la valeur cible et de l'inciter à continuer un peu plus afin d'atteindre l'intervalle plus restrictif, nous avons repensé le système de récompenses.

Le système de récompense est le même que précédemment, seule l'attribution de la récompense pour la précision finale varie. Il a été modifié afin que la récompense ne dépende plus d'une équation mais qu'elle soit progressive par paliers en fonction de la précision atteinte. Le Tableau 20 présente les récompenses attribuées en fonction de la précision atteinte, en valeur absolue.

Tableau 20. Récompense de précision pour l'intervalle 23,5-26,5

| **Condition (si la valeur est inférieure ou égale)** | **Récompense attribuée** |
|---|---|
| 0,1 | 0,95 |
| 0,3 | 0,9 |
| 0,5 | 0,85 |
| 0,7 | 0,7 |
| 0,9 | 0,6 |
| 1,1 | 0,5 |
| 1,3 | 0,4 |
| 1,5 | 0,3 |
| Au-delà de 1,5 | -1 |

Les résultats obtenus sont regroupés dans le Tableau 21 et combinés à ceux obtenus précédemment.

En comparant les pourcentages pour l'apprentissage, les meilleurs résultats sont obtenus pour les récompenses finales entre 23,5 et 26,5, et notamment le modèle avec un état dont les valeurs sont normalisées. Dans la Figure 36, il est possible d'observer la répartition des récompenses perçues pour le modèle ayant obtenu les meilleurs résultats. Les traits pointillés horizontaux représentent les intervalles : [23,5 ;26,5] en rouge, [24,5 ;25,5] en orange, [24,9 ;25,1] en gris. Les quatre cycles dont la masse finale est en dehors de

[23,5 ;26,5] sont bien discernables. Enfin, il est possible de remarquer que l'agent a tendance à trop remplir les sacs.

Tableau 21. Regroupement des différents résultats obtenus

| | **Modèle** | **Résultats d'entrainement (en pourcentage)** | | **Résultats de validation (en pourcentage)** | |
|---|---|---|---|---|---|
| | | % 23,5 – 26,5 | % 24,5 – 25,5 | % 23,5 – 26,5 | % 24,5 – 25,5 |
| Récompense finale entre 24,5 et 25,5 | Un état | 3,62 | 1,32 | 82,88 | 22,52 |
| | Un état normalisé | 4,48 | 1,52 | 1,00 | 0,00 |
| | Quatre états | 3,63 | 1,43 | 35,35 | 18,18 |
| | Sept états | 3,58 | 1,39 | 62,00 | 26,00 |
| Récompense finale entre 23,5 et 26,5 | Un état | 6,43 | 2,51 | 0,00 | 0,00 |
| | Un état normalisé | 27,5 | 9,67 | **96,00** | **36,00** |

Afin de vérifier le bon apprentissage des modèles, nous avons étudié les valeurs Q du réseau DQN, qui représentent les estimations de récompenses futures pour chaque action possible dans un état donné. Cela nous permet d'évaluer les performances du modèle DQN et de vérifier s'il converge de façon optimale. Il est apparu que leur évolution démontrait un apprentissage stable, mais que les valeurs avaient une évolution très lente.

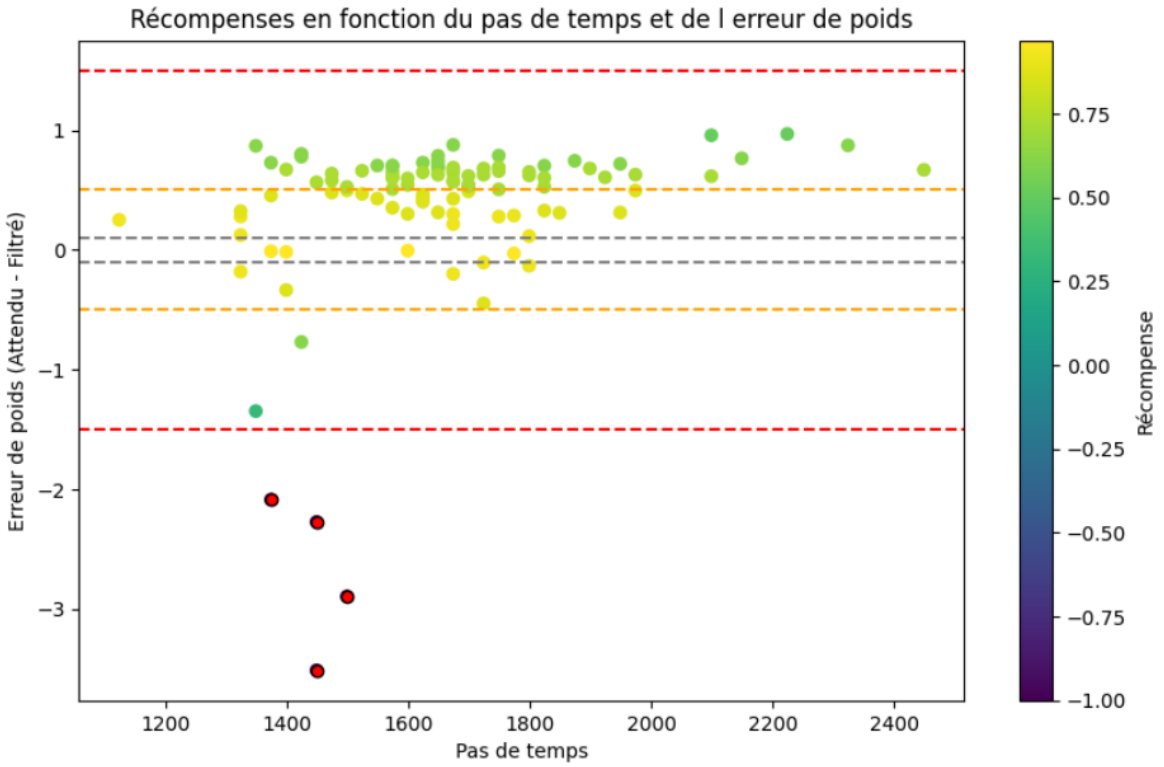



Figure 36. Résultats de validation pour le modèle à un état normalisé et récompense entre 23,5 et 26,5

Ainsi, les résultats obtenus pour le meilleur modèle, nous montre que l'agent apprend mais qu'il apprend trop lentement. Nous allons donc conserver cette configuration, c'est-à-dire celle à un état normalisé et récompense finale entre 23,5 et 26,5, et modifier des hyperparamètres propres aux performances du modèle DQN afin d'accélérer l'apprentissage. Les hyperparamètres testés et leurs valeurs possibles sont les suivants :

- Le taux d'apprentissage : 0,002 ; 0,005 ; 0,01
- La taille de lot : 64 ; 128 ; 256
- La fréquence de mise à jour du réseau cible : 50 ; 100 ; 500

Au total, sept combinaisons avec des valeurs d'hyperparamètres différents ont pu être testés sur la meilleure configuration, le Tableau 22 regroupe les résultats obtenus. Les valeurs sont en pourcentage et correspondent aux proportions de pesées présentes dans l'intervalle de poids indiqué.

Tableau 22. Résultats avec des valeurs différentes d'hyperparamètres

| | **Hyperparamètres** | | | **Entrainement** | | | **Validation** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Taux d'appren-tissage** | **Taille de lot** | **Fréquence de mise à jour du réseau cible** | **23,5-26,5** | **24,5-25,5** | **24,9-25,1** | **23,5-26,5** | **24,5-25,5** | **24,9-25,1** |
| Modèle 1 | 0,002 | 64 | 50 | 90,92 | 44,38 | 7,03 | 91,09 | 40,59 | **7,92** |
| Modèle 2 | 0,002 | 128 | 500 | 77,18 | 38,42 | 6,44 | 77,23 | 35,64 | 4,95 |
| Modèle 3 | 0,005 | 64 | 500 | 93,25 | 39,3 | 5,56 | 96 | 46 | 5 |
| Modèle 4 | 0,01 | 128 | 100 | 86,19 | 43,63 | 7,79 | 94,06 | 34,65 | 5,94 |
| Modèle 5 | 0,002 | 64 | 500 | 27,06 | 9,69 | 1,3 | 96 | 30 | 1 |
| Modèle 6 | 0,002 | 256 | 100 | 90,24 | 33,59 | 5,13 | 93,07 | 38,61 | 2,97 |
| Modèle 7 | 0,01 | 64 | 100 | 88,85 | 40,36 | 6,48 | 65,69 | 27,45 | 4,9 |

Concernant les résultats de tests, le modèle qui a permis d'obtenir le plus de pesées dans l'intervalle le plus restrictif, c'est-à-dire entre 24,9 et 25,1, est le modèle 1 qui a la plus petite valeur de fréquence de mise à jour du réseau cible. De plus, nous pouvons observer que le modèle ayant les moins bons résultats est le modèle 5, qui a les mêmes valeurs de taille de lot et de taux d'apprentissage que le premier modèle. Néanmoins, une valeur de fréquence élevée n'est pas synonyme de mauvais résultats, comme nous pouvons le voir avec les résultats des modèles 2 et 3. Ces résultats montrent qu'il n'est pas possible de définir une valeur d'hyperparamètre unique comme étant plus propice à de meilleurs résultats. Cependant, une certaine cohérence dans les valeurs des hyperparamètres semble nécessaire pour obtenir de meilleurs résultats. En effet, lorsqu'une valeur de fréquence élevée est combinée à une taille de lot plus grande (voir modèle 2), cela permet d'obtenir de meilleurs

résultats que lorsque ces valeurs élevées sont attribuées isolément (voir modèles 5 et 6). Un élément qui a été mis en place initialement et qui est présent dans tous les modèles présentés précédemment, est l'obligation pour l'agent de rester en grand débit jusqu'à atteindre 20 kg. Cela permet d'éviter qu'il perde du temps à tester des configurations où il s'arrête trop tôt et doit continuer très longtemps en petit débit. Ainsi, ce n'est qu'une fois les 20 kg atteint qu'il prend la décision de rester dans la phase de grand débit ou de passer en petit débit. Dans le but de réduire la quantité d'épisodes nécessaires à l'entrainement, nous avons essayé de modifier cette valeur à 22 kg. Nous avons testé cela sur le modèle un état normalisé avec récompenses entre 23,5 et 26,5, et avec les valeurs d'hyperparamètres initiales. Le modèle a été entrainé sur uniquement 5000 pesées, au lieu de 8 000, les résultats obtenus sont regroupés avec ceux du modèle où cette valeur est à 20 kg, dans le Tableau 23, afin de mieux les comparer.

Tableau 23. Résultats obtenus pour une valeur de changement à 22kg

| | **Proportion de cycles ayant un poids final dans l'intervalle 23,5-26,5 (en pourcentage)** | **Proportion de cycles ayant un poids final dans l'intervalle 24,5-25,5 (en pourcentage)** | **Proportion de cycles ayant un poids final dans l'intervalle 24,9-25,1 (en pourcentage)** |
|---|---|---|---|
| Changement à 22 kg Entrainement | 74,21 | 29,94 | **5,78** |
| Changement à 22 kg Validation | 72,92 | 30,21 | **5,21** |
| Changement à 20 kg Entrainement | 27,5 | 9,67 | 1,44 |
| Changement à 20 kg Validation | 96,00 | 36,00 | 4,02 |

Si l'on compare les résultats d'entrainement entre le modèle initial et celui-ci, les résultats obtenus sont meilleurs concernant les valeurs d'entrainement et de poids finaux entre 24,9 et 25,1, pour ce modèle avec une valeur de changement autorisée à partir de 22 kg. Par contre,

pour les résultats de validation, les proportions obtenues sont moins intéressantes. En effet, même s'il y a moins de pesées arrivées dans l'intervalle le plus restrictif, pour le modèle à 20 kg, l'agent a été bien plus capable d'appliquer ses connaissances en produisant quasiment que des pesées dont le poids est entre 23,5 et 26,5. Pour le modèle à 22 kg, il serait peut-être nécessaire de prolonger l'entrainement jusqu'à 8 000 cycles ou, préférablement, de combiner cette valeur de changement de phase possible à partir de 22 kg avec la combinaison d'hyperparamètres du modèle 1.

Afin d'avoir un élément de comparaison entre les récompenses finales et les récompenses intermédiaires, nous avons entrainé sur 4000 pesées le modèle à un état normalisé. À la place de donner comme récompense finale une addition de la récompense liée au temps et de celle liée à la précision, nous avons gardé la récompense finale attribuée en fonction de la précision entre 23,5 et 26,5 qui a été présentée dans le Tableau 20. Pour la récompense intermédiaire, nous avons fourni au modèle l'autre composante du système de récompense initial, c'est-à-dire la formule suivante :

$$récompense\ intermédiaire = \frac{poids\ actuel}{pas\ de\ temps\ actuel + \varepsilon} \quad (7)$$

Elle est non-cumulative, c'est-à-dire que cette valeur est recalculée et attribuée telle qu'elle est à chaque pas de temps du cycle. Les résultats obtenus sont présentés dans le Tableau 24.

Tableau 24. Résultats obtenus pour un modèle avec récompenses intermédiaires

| | **Proportion de cycles ayant un poids final dans l'intervalle 23,5-26,5** | **Proportion de cycles ayant un poids final dans l'intervalle 24,5-25,5** | **Proportion de cycles ayant un poids final dans l'intervalle 24,9-25,1** |
|---|---|---|---|
| Entrainement | 94,45 | 39,68 | 5,52 |
| Validation | 92,86 | 41,84 | 8,16 |

Malgré un entrainement réalisé uniquement sur 4000 pesées, les résultats obtenus pour cette configuration de récompenses sont aussi bons, voire meilleurs en ce qui concerne l'intervalle le plus restrictif pour la validation.

En ce qui concerne les durées des phases du cycle, le Tableau 25 regroupe les résultats de validation pour la meilleure configuration.

Tableau 25. Valeurs de durée des cycles pour la meilleure configuration

| | **Grand débit** | **Changement de débit** | **Petit débit** | **Fermeture** | **Stabilisation** |
|---|---|---|---|---|---|
| Minimum | 399 | 249 | 24 | 224 | 74 |
| Maximum | 1024 | 324 | 11949 | 374 | 324 |
| Moyenne | 606.2 | 255.6 | 389.7 | 278.3 | 146.5 |
| Médiane | 599 | 249 | 374 | 274 | 124 |
| Écart-type | 66.24 | 19.56 | 523.8 | 27.31 | 51.79 |

Ces résultats sont donnés à titre informatif, car il n'est pas possible d'en déduire de conclusions étant donné que peu de cycles atteignent l'intervalle de [24,9 ;25,1]. Ainsi les valeurs de temps de ces cycles correspondent à des cycles où la masse est majoritairement entre 23,5 et 26,5 kg. Cependant, pour atteindre la précision souhaitée, le petit débit doit être utilisé et c'est ce qui a va le plus rallonger la durée de cycle.

### 5.3.6 Discussion sur la deuxième étape de l'approche

Ainsi, ces premiers résultats prouvent que le modèle d'apprentissage par renforcement est capable d'apprendre de nos données, malgré un environnement continu avec de nombreux états. Néanmoins, les modèles ayant un système de récompenses finales montrent des résultats perfectibles, notamment en combinant les meilleurs combinaisons d'hyperparamètres avec un seuil de changement de phase autorisé plus grand. Cette approche pourrait constituer une première piste d'améliorations. Cependant, d'autres hyperparamètres, tels que ceux liés à l'exploration et à l'exploitation pourraient également être une piste

d'améliorations. Concernant l'entrainement réalisé avec des récompenses intermédiaires, un entrainement sur un plus grand volume de données pourrait être intéressant afin de vérifier si une période d'apprentissage prolongée permet de surpasser les autres modèles.

D'autres configurations spécifiques au modèle DQN pourraient également être testés, notamment l'intégration d'un modèle appelée « *prioritized experience replay* ». Comme nous avons pu le voir, le rejeu d'expérience standard sélectionne des échantillons de façon aléatoire dans la mémoire tampon pour réutiliser les expériences passées, stabiliser l'apprentissage et améliorer la convergence de l'agent. En revanche, le rejeu d'expériences priorisées va attribuer une priorité à chaque transition afin de favoriser celles qui sont jugées plus importantes pour l'apprentissage.

Ainsi, ces tests à réaliser constituent les travaux à court terme pouvant être réalisés à la suite de ce projet, pour trouver un modèle demandant moins de données d'entrainement.

## 5.4 SYNTHÈSE DU CHAPITRE

L'approche en deux étapes, développée dans ce chapitre, s'est montrée prometteuse pour notre application. Concernant l'étape de filtrage, les modèles d'apprentissage supervisé se sont avérés moins fiables et moins efficaces que les filtres numériques. Les modèles d'IA nécessitant un entrainement trop conséquent pour obtenir des résultats bien moins performants, nous avons donc conclu que les filtres numériques étaient plus appropriés pour notre application, notamment le filtre de Butterworth. Ce filtre, dont les paramètres dépendent des caractéristiques du signal à filtrer, est reconfigurable en cas de changement dans le signal. Pour la deuxième étape de cette approche d'optimisation, les premiers résultats de l'apprentissage par renforcement se sont montrés encourageants, malgré le temps limité pour développer cette approche. Les résultats confirment que l'agent apprend, bien que lentement. Les différentes configurations d'hyperparamètres testées pour le meilleur modèle trouvé ont permis de dégager des pistes pour continuer l'optimisation de l'agent. Néanmoins,

un travail d'optimisation est également nécessaire au niveau du système de récompenses. La garantie d'un modèle qui apprend à respecter l'objectif, nous a incités à favoriser l'utilisation de récompenses finales. Comme précisé précédemment, les récompenses intermédiaires nécessitent plusieurs essais et erreurs afin de trouver la bonne combinaison qui amène l'agent à apprendre plus vite sans négliger l'objectif final.

Le temps d'entrainement des modèles étant d'environ d'une dizaine d'heures pour mille cycles, soit plus de trois jours pour entrainer les 8000 cycles, et l'accès aux ressources computationnelles étant limité, nous avons dû faire des choix stratégiques pour nos tests des modèles. Néanmoins, des travaux sont en cours pour continuer d'entrainer et de tester différentes configurations. L'application des connaissances de l'agent sur l'ensemble de test montre qu'il est performant lorsqu'il s'agit des cycles qu'il n'a jamais vu. Cette idée d'accélérer l'apprentissage de l'agent et de réduire le nombre de pesées nécessaires est importante afin de proposer une solution qui soit généralisable à d'autres poids cibles et matières pesées sans nécessiter un entrainement trop contraignant.

Ainsi ces premiers résultats confirment que l'apprentissage par renforcement combiné au filtrage, est adapté pour fournir un modèle complet, basé uniquement sur le poids brut.

# CONCLUSION GÉNÉRALE

Pour répondre aux exigences en matière de fiabilité et de rapidité, l'IA est de plus en plus intégrée dans l'industrie, permettant aux entreprises de rester toujours compétitives. Dans ce contexte, nous avons été chargés d'un projet visant à optimiser le cycle de pesage d'une balance industrielle. L'objectif était de développer un modèle basé sur l'IA capable d'optimiser la durée du cycle de pesage sans compromettre la précision.

La première étape a consisté à constituer une base de données de 10 000 cycles de pesage, ce qui nous a permis de comprendre en profondeur le fonctionnement du processus. Pour répondre au souhait de l'entreprise de travailler à partir du poids brut, nous avons d'abord développé une approche basée sur la prédiction de données temporelles. Les résultats obtenus avec cette méthode nous ont conduit à changer notre stratégie d'optimisation. La seconde approche mise en place s'inspire du fonctionnement de l'API actuel, en combinant une étape de filtrage avec une étape d'optimisation par apprentissage par renforcement. Repenser notre approche suite aux résultats de la première nous a permis de trouver une solution plus adaptée. Pour l'étape de filtrage, les résultats ont démontré que l'IA n'était pas la méthode la plus appropriée. En effet, nous ne disposions pas suffisamment de données d'entrainement pour obtenir des résultats probants avec des modèles d'apprentissage profond. Les données étant assez variables et l'apprentissage profond nécessitant une quantité de données plus importantes, continuer la recherche de paramètres ou de modèles différents aurait pu donner de meilleurs résultats. Cependant, les modèles auraient été peu adaptables à des données qu'ils n'auraient jamais vues. La constitution de la base de données composée de 10 000 cycles étant déjà une tâche conséquente, il n'était pas envisageable de proposer une solution nécessitant un nombre encore plus important de cycles. De plus, cette opération de collecte des données devrait être répétée à chaque changement de matière, ce

qui limite la fiabilité des résultats face à des cycles différents de ceux utilisés pour l'entrainement. Les résultats obtenus avec l'apprentissage automatique sont supérieurs, mais les meilleurs sont ceux obtenus avec les filtres numériques. Ces filtres se sont donc avérés les plus appropriés pour notre application, notamment le filtre de Butterworth, car ils sont fiables et rapides à mettre en place. En cas de changement de matière pesée, il sera possible d'ajuster les paramètres du filtre sans avoir à faire un nouvel entrainement. Il suffira de modifier les paramètres et de vérifier si le résultat obtenu est satisfaisant. Cette méthode de filtrage est donc plus adaptée pour développer un modèle généralisable.

Pour l'étape d'optimisation, le principe de l'apprentissage par renforcement nous laissait penser que l'optimisation serait possible, en raison de sa capacité à apprendre et à optimiser les actions choisies. Les premiers résultats confirment que l'agent est capable d'apprendre et de respecter nos contraintes, bien qu'il manque encore d'entrainement. Les résultats obtenus en testant différentes valeurs d'hyperparamètres, nous offre des pistes d'amélioration. Cependant, les contraintes liées au temps et à l'accès aux ressources computationnelles ont limité le nombre d'essais que nous avons pu effectuer. De plus, nous avons testé seulement un modèle DQN. L'objectif pour la suite de ce projet serait d'optimiser le modèle DQN afin qu'il puisse apprendre avec le moins de données possibles, et si celui-ci se montre limité, peut-être envisager de changer l'architecture du modèle de l'agent. Ainsi, les travaux futurs à court terme seraient de continuer les essais afin de trouver une configuration qui respecte les critères de précision et de temps, tout en nécessitant le moins de pesées possibles pour l'entrainement, dans le but de créer un modèle généralisable. Les possibilités d'hyperparamètres et de configurations étant assez importantes, il pourrait être envisagé d'entrainer les données sur une quantité plus réduite de pesées, par exemple 2 000 pesées, et comparer les résultats sur ces entrainements réduits. Cela permettrait de réduire le temps d'entrainement et d'ainsi tester plus de configurations possibles sur les ressources computationnelles à disposition.

Concernant les travaux à long terme, l'implémentation du modèle entrainé sur la balance industrielle pourrait être envisagée pour évaluer son efficacité en temps réel. Pour

cela, il serait nécessaire de donner au modèle d'apprentissage par renforcement la décision de mise en mouvement du servomoteur de la trémie d'écoulement. Les autres opérations, comme l'ensachage ou la tare, seraient alors gérées par l'API de la balance industrielle. Parallèlement, des collectes de données pourraient être réalisées pour quelques milliers de cycles, à chaque nouveau type de produit ensaché. Ces données constitueraient une base de données permettant de pré-entrainer les modèles avant de les intégrer aux balances pour l'ensachage d'un nouveau produit.

Ces démarches permettront de proposer une solution optimisée et généralisable, renforçant ainsi la compétitivité et la fiabilité des processus industriels grâce à l'intégration intelligente de l'IA.

# ANNEXE A – CODES DE L'OPTIMISATION EN UNE ÉTAPE : MÉTHODES STATISTIQUES

- Méthodes statistiques : prédictions des valeurs suivantes dans la phase de stabilisation

```
# Appel des bibliothèques

import warnings  # Pour ignorer les avertissements
warnings.filterwarnings("ignore")  # Pour désactiver les avertissements
import math  # Pour les opérations mathématiques
import numpy as np  # Pour les opérations sur les tableaux et les calculs numériques
import csv  # Pour lire et écrire des fichiers CSV
import os  # Pour interagir avec le système de fichiers
from matplotlib import pyplot as plt  # Pour créer des graphiques
import random  # Pour générer des valeurs aléatoires
import time  # Pour gérer les opérations liées au temps
import datetime  # Pour manipuler les dates et les heures
import pandas as pd  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Pour lire les fichiers CSV
from matplotlib import pyplot  # Pour créer des graphiques
from statsmodels.tsa.statespace.sarimax import SARIMAX  # Pour créer et ajuster des modèles SARIMAX
import numpy  # Pour les opérations sur les tableaux et les calculs numériques
from sklearn.metrics import mean_squared_error  # Pour évaluer les modèles en calculant l'erreur quadratique moyenne

# Télécharger les données

data_initial = read_csv('données prétraitées.csv',names=['a','bulk','dribble','time','phase','f','g','p_filtrefiltre',
                                                         'i','j','brut','tare','precision'], header=None)
poids_brut=data_initial.brut
phase=data_initial.phase
donnees_exog=data_initial.precision

#autres valeurs possibles exog:
#donnees_exog=data_initial.dribble
#donnees_exog=data_initial.bulk
```

```
# Identifie où se situe chaque phase de stabilisation

iterated_index_position_list8 = [ i for i in range(0,len(phase)) if phase[i] == 8]

iterated_index_position_list8

debut_phase_8=[]
fin_phase_8=[]

def interval_extract(t):
    length = len(t)
    i = 0
    while (i< length):
        low = t[i]
        while i <length-1 and t[i]+1 == t[i + 1]:
            i += 1
        high = t[i]
        if (high - low >= 1):
            debut_phase_8.append(low)
            fin_phase_8.append(high)
        else:
            debut_phase_8.append(high)
            fin_phase_8.append(low)
        i += 1
    return debut_phase_8,fin_phase_8

interval_extract(iterated_index_position_list8 )
```

```
# Découpage des données en entraînement et validation

data_pesee1=poids_brut[debut_phase_8[0]:fin_phase_8[0]]
split_point=int(0.78*len(data_pesee1))
entrainement_pesee1 = data_pesee1[:split_point]
validation_pesee1 =data_pesee1[split_point:]

data_exog_pesee1=donnees_exog[debut_phase_8[0]:fin_phase_8[0]]
exog_train_pesee1=data_exog_pesee1[:split_point]
exog_test_pesee1=data_exog_pesee1[split_point:]

fin_valid_pesee1=len(validation_pesee1)
```

```
# Boucle d'identification des meilleurs hyperparamètres

# Valeurs possibles des hyperparamètres
p_values = [0, 1, 2, 3, 4, 5]
d_values = [0, 1, 2, 3]
q_values = [0, 1, 2, 3, 4, 5]

rmse_pesee_1=[]
mape_pesee_1=[]
params_pesee_1=[]

best_cfg, best_score = 10000,10000 # Valeurs au hasard, volontairement très grandes
p=0
q=0
d=0
for p in p_values:
        for d in d_values:
            for q in q_values:
                config = (p,d,q)
                params_pesee_1.append(b)
                try:
                    print(config)
                    model=SARIMAX(entrainement_pesee1,order=config,exog=exog_train_pesee1,
                                  initialization='approximate_diffuse')
                    model_fit_test=model.fit()
                    forecast_test=model_fit_test.predict(start=split_point, end=split_point+fin_valid_pesee1-1,
                                                         exog=exog_test_pesee1)
                    prediction_test=[]
                    for yhat_pesee_1 in forecast_test:
                        prediction_test.append(yhat_pesee_1)
                    expected_test=validation_pesee1[0:fin_valid_pesee1+1]
                    expected_test=np.array(expected_test)
                    prediction_test=np.array(prediction_test)
```

```
                    AICs=model_fit_test.aic
                    rmse=np.sqrt(mean_squared_error(expected_test, prediction_test)).round(6)
                    mape=np.around(np.mean(np.abs(expected_test-prediction_test)/expected_test)*100,2)
                    print('RMSE=',rmse)
                    print('MAPE=',mape)
                    print('AIC=', AICs)
                    rmse_pesee_1.append(rmse)
                    mape_pesee_1.append(mape)
                    if rmse<best_score:
                        best_score=rmse
                        best_cfg=config
                except ValueError:
                    rmse_pesee_1.append(None)
                    mape_pesee_1.append(None)
                    print('RMSE=',rmse)
                    print('MAPE=', mape)
                    continue
                except np.linalg.LinAlgError:
                    rmse_pesee_1.append(None)
                    mape_pesee_1.append(None)
                    print('RMSE=',rmse)
                    print('MAPE=', mape)
                    continue
print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))
```

```
# Appliquer le meilleur modèle trouvé

model=SARIMAX(entrainement_pesee1,order=best_cfg,exog=exog_train_pesee1, initialization='approximate_diffuse')
model_fit_test=model.fit()
print(model_fit_test.summary())

pred=model_fit_test.predict(start=split_point,end=split_point+fin_valid_pesee1-1, exog=exog_test_pesee1)

# Tracés des données d'entrainement
plt.plot(entrainement_pesee1)
plt.plot(pred)
plt.show()
```

```
# Application du meilleur modèle aux pesées suivantesi

index=0
RMSE_pesees_suivantes=[]
MAPE_pesees_suivantes=[]
erreur_maximale=[]
moyenne_des_erreurs=[]
longueur_entrainement=[]
longueur_validation=[]
longueur_phase_8=[]
liste_index_debut_ph8=[]
liste_index_fin_ph8=[]

while index!=len(debut_phase_8):
    try:
        # Préparation des données
        index_debut_ph8=debut_phase_8[index]+1
        liste_index_debut_ph8.append(index_debut_ph8)
        index_fin_ph8=fin_phase_8[index]+1
        liste_index_fin_ph8.append(index_fin_ph8)
        poids_brut_ph8=poids_brut[index_debut_ph8:index_fin_ph8]
        split_point=int(0.78*len(poids_brut_ph8))
        entrainement = poids_brut_ph8[:split_point]
        validation=poids_brut_ph8[split_point:]
        index_fin_valid=len(validation)
        data_exog_pesees_suivantes=donnees_exog[index_debut_ph8:index_fin_ph8]
        exog_train_pesees_suivantes=data_exog_pesees_suivantes[:split_point]
        exog_test_pesees_suivantes=data_exog_pesees_suivantes[split_point:]
```

```
        longueur_phase_8.append(len(poids_brut_ph8))
        longueur_entrainement.append(len(entrainement))
        longueur_validation.append(len(validation))
        print('Dataset %s, Validation %d' % (len(entrainement), len(validation)))

        # Application du modèle pour les prédictions
        model=SARIMAX(entrainement,order=best_cfg, exog=exog_train_pesees_suivantes,initialization='approximate_diffuse')
        model_fit=model.fit()
        forecast=model_fit.predict(start=split_point,end=split_point+index_fin_valid-1,exog=exog_test_pesees_suivantes)
        prediction=[]
        for yhat in forecast:
            prediction.append(yhat)
        expected=validation[0:index_fin_valid+1]
        expected=np.array(expected)
        prediction=np.array(prediction)
        rmse=np.sqrt(mean_squared_error(expected, prediction)).round(6)
        mape=np.around(np.mean(np.abs(expected-prediction)/expected)*100,2)
        print('RMSE=',rmse)
        print('MAPE=', mape)
        RMSE_pesees_suivantes.append(rmse)
        MAPE_pesees_suivantes.append(mape)
        expected = pd.DataFrame(expected)
        prediction = pd.DataFrame(prediction)
        erreur=np.array(abs(prediction-expected))
        moyenne =erreur.mean()
        erreur_maximale.append(max(erreur))
        moyenne_des_erreurs.append(moyenne)
        index = index+1
```

```
    except ValueError:
        RMSE_pesees_suivantes.append(None)
        MAPE_pesees_suivantes.append(None)
        erreur_maximale.append(None)
        moyenne_des_erreurs.append(None)
        index=index+1
        pass
    except np.linalg.LinAlgError:
        RMSE_pesees_suivantes.append(None)
        MAPE_pesees_suivantes.append(None)
        erreur_maximale.append(None)
        moyenne_des_erreurs.append(None)
        index=index+1
        pass
```

- Méthodes statistiques : prédictions des poids finaux des pesées suivantes

```
# Appel des bibliothèques

from math import sqrt  # Pour calculer la racine carrée
from warnings import catch_warnings  # Pour gérer les avertissements dans le code
from warnings import filterwarnings  # Pour filtrer les avertissements spécifiques
from sklearn.metrics import mean_squared_error  # Pour évaluer les modèles en calculant l'erreur quadratique moyenne
from pandas import read_csv  # Pour lire les fichiers CSV
import numpy  # Pour les opérations sur les tableaux et les calculs numériques
from matplotlib import pyplot as plt  # Pour créer des graphiques
import pandas as pd  # Pour la manipulation et l'analyse des données
import statsmodels.api as sm  # Pour les modèles statistiques (ARIMA, ARIMAX ...)


# Téléchargement des données

data = pd.read_csv('poids_finaux.csv', names=['index','filtré','poids_brut', 'point_consigne_bulk',
                                              'point_consigne_dribble',
                                              'precision_poids_final'],header=None, index_col=None)

poids_brut=data.poids_brut
donnees_exogenes=data.precision_poids_final


p_values = [0, 1, 2, 3, 4, 5]
d_values = [0, 1, 2]
q_values = [0, 1, 2, 3, 4, 5]
proportion_train = 0.75
```

```
# Préparation des données

train_end=int(proportion_train*len(data))
validation_end=len(data)

poids_brut_train = poids_brut[:train_end]
donnees_exogenes_train = donnees_exogenes[:train_end]

poids_brut_test=poids_brut[train_end:test_end+1]
donnees_exogenes_test=donnees_exogenes[train_end:validation_end+1]
fin_test=len(poids_brut_test)

validation=poids_brut[train_end:validation_end+1]
plt.plot(poids_brut_train)
```

```
# Grille de paramètres pour trouver la meilleure configuration

best_cfg_ARIMAX, best_score = 10000,10000 # Valeurs au hasard, volontairement très grandes
q=0
d=0
for p in p_values:
        for d in d_values:
            for q in q_values:
                config = (p,d,q)
                try:
                    print(config)

                    # Avec prise en compte de valeurs exogènes
                    arimax=sm.tsa.statespace.SARIMAX(poids_brut_train, order=config, exog=donnees_exogenes_train,
                                                     initialization='approximate_diffuse').fit()
                    prediction_ARIMAX=arimax.predict(train_end,test_end-1, exog=donnees_exogenes_test)

                    # Sans prise en compte de valeurs exogènes :
                    #arimax=sm.tsa.statespace.SARIMAX(poids_brut_train, order=config).fit()
                    #prediction_ARIMA=arimax.predict(train_end,test_end-1)

                    rmse_test=numpy.sqrt(mean_squared_error(validation, prediction_ARIMAX)).round(6)
                    mape_test=numpy.around(numpy.mean(numpy.abs(validation-prediction_ARIMAX)/validation)*100,2)
                    print('RMSE=',rmse_test)
                    print('MAPE=',mape_test)
                    if rmse_test<best_score:
                        best_score=rmse_test
                        best_cfg_ARIMAX=config
```

```
                except ValueError :
                    rmse_test=None
                    mape_test=None
                    print('RMSE=',rmse_test)
                    print('MAPE=', mape_test)
                    continue
                except numpy.linalg.LinAlgError:
                    rmse_test=None
                    mape_test=None
                    print('RMSE=',rmse_test)
                    print('MAPE=', mape_test)
                    continue
print('Best ARIMAX%s RMSE=%.6f' % (best_cfg_ARIMAX, best_score))
```

```
# Générer les résultats avec la meilleure configuration trouvée


# Avec valeurs exogènes
arimax=sm.tsa.statespace.SARIMAX(poids_brut_train, order=best_cfg_ARIMAX).fit()

# Sans valeurs exogènes
#arima=sm.tsa.statespace.SARIMAX(poids_brut_train, order=best_cfg_ARIMA).fit()

arimax.summary()

prediction_ARIMAX_best_config=arimax.predict(train_end,test_end-1, exog=donnees_exogenes_test)

validation=poids_brut[train_end:test_end+1]

RMSE_ARIMAX_best_config=sqrt(mean_squared_error(validation, prediction_ARIMAX_best_config))

print('ARIMAX model RMSE:',RMSE_ARIMAX_best_config)


# Tracés des résultats :
plt.plot(poids_brut_train)
plt.plot(prediction_ARIMAX_best_config)
plt.show()
```

# ANNEXE B – CODES DE L'OPTIMISATION EN UNE ÉTAPE : MODÈLES 1 ET 2

- Modèle 1 : Allongement de la phase de grand débit avec un réseau CNN

```
#Appel des bibliothèques

import numpy  # Pour les opérations sur les tableaux et les calculs numériques
import pandas  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Lire les fichiers CSV

from sklearn.preprocessing import MinMaxScaler  # Pour normaliser les données
import matplotlib.pyplot as plt  # Créer des graphiques
from math import sqrt  # Ccalculer la racine carrée
from sklearn.metrics import mean_squared_error  # Evaluer les modèles

import tensorflow as tf  # Pour la création et l'entraînement de modèles de machine learning

from keras.models import Sequential  # Créer des modèles séquentiels
from keras.optimizers import Adam  # Importer l'optimiseur Adam
from keras.models import load_model  # Charger des modèles sauvegardés

from keras.models import Model  # Pour créer des modèles plus complexes
from keras.layers import Input, Dense, LSTM  # Importation des couches Input, Dense et LSTM
from keras.layers import LayerNormalization, MultiHeadAttention, Dropout, Add  # Importation de couches avancées
from keras.layers import Conv1D, MaxPooling1D, Flatten  #Couches Conv1D, MaxPooling1D et Flatten pour les CNN
```

```
#téléchargement de toutes les données

name=['flux','bulk_cutoff','dribble_cutoff','time','step','f','g','filter_weight','open_percentage',
      'empty_activation','raw_weight','tare']


dataset_complet=read_csv('donnees_concat', header=None, names=name)

phase=dataset_complet.step
poids_brut=dataset_complet.raw_weight

data=poids_brut.to_frame()
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

```
# Hyperparamètres du modèle :

alpha = [0.001, 0.002, 0.005, 0.0001]  # Taux d'apprentissage pour l'optimisation du modèle
batch_size = [16, 32, 64, 128]  # Tailles de lots pour l'entraînement du modèle
nodes = [16, 32, 64, 128]  # Nombres de neurones pour les couches du modèle

nb_epochs=100 # Nombre d'époques pour l'entraînement du modèle

window_size=[25,60,100,150] # Tailles de fenêtres pour les séquences de données d'entrée


# Taille des ensembles d'entraînement et de validation :
train_size = 0.75  # Proportion des données utilisées pour l'entraînement
validation_size = 0.15  # Proportion des données utilisées pour la validation
```

```python
# Fournir la liste des configurations de paramètres testées
def model_configs(parametre_1,parametre_2, parametre_3, parametre_4):
    # create configs
    configs = list()
    for i in parametre_1:
        for j in parametre_2:
            for k in parametre_3:
                for l in parametre_4:
                    cfg = [i, j, k, l]
                    configs.append(cfg)
    print('Total configs: %d' % len(configs))
    return configs

# Afficher les différentes configurations de paramètres testées
cfg= model_configs(batch_size, alpha, window_size, nodes)
print("liste des configurations = ",cfg)
```

```python
# Trouver les durées de phases
def valeurs_des_index(dataset_phase,numero_phase,a,b):
    debut_phase,fin_phase = [],[]
    index= [i for i in range(a,b) if dataset_phase[i] == numero_phase]
    length = len(index)
    i = 0
    while (i< length):
        low = index[i]
        while i <length-1 and index[i]+1 == index[i + 1]:
            i += 1
        high = index[i]
        if (high - low >= 1):
            debut_phase.append(low)
            fin_phase.append(high)
        else:
            debut_phase.append(high)
            fin_phase.append(low)
        i += 1
    return debut_phase,fin_phase
```

```python
# Définir la phase de grand débit :
debut_3, fin_3 = valeurs_des_index(phase, 3, 0, len(phase))
debut_1, fin_1 = valeurs_des_index(phase, 1, 0, len(phase))

debut_grand_debit = debut_1
fin_grand_debit = fin_3

# Définir les ensembles de test et d'entrainement :

train_size = int(proportion_train_size * len(fin_grand_debit))
validation_size = int(proportion_validation_size * len(fin_grand_debit))
test_size = len(fin_grand_debit)-train_size-validation_size

print("train_size = ", train_size)
print("validation_size = ", validation_size)
print("test_size=", test_size)

debut_train, fin_train = debut_grand_debit[0:train_size],fin_grand_debit[0:train_size]
debut_validation = debut_grand_debit[train_size:train_size+validation_size]
fin_validation = fin_grand_debit[train_size:train_size+validation_size]
debut_test, fin_test = debut_grand_debit[-test_size:],fin_grand_debit[-test_size:]
```

```python
# Créer les entrées/sorties
def create_sequences(data, debut, fin, window_size):
    X = []
    y = []
    for t in range(len(debut)):
        for i in range(debut[t],fin[t]-window_size+1):
            X.append(data[i:i+window_size])
            y.append(data[i+window_size])
    return numpy.array(X), numpy.array(y)
```

```python
#Création du réseau CNN

val_loss_score=[]

def model_CNN(configuration, X_train, y_train, X_validation, y_validation, tracé_loss):
    n_features=1
    batch_size, alpha, window_size, nodes = configuration
    optimizer= Adam(learning_rate=alpha)
    # Define the Transformer model architecture
    input_layer = Input(shape=(window_size, n_features))
    x = Conv1D(nodes, 3, activation='relu')(input_layer)
    x = MaxPooling1D(1)(x)
    x = Conv1D(nodes, 3, activation='relu')(x)
    x = MaxPooling1D(1)(x)
    x = Flatten()(x)
    x = Dense(1, activation='relu')(x)
    output_layer = Dense(1, activation='relu')(x)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(loss='mse', optimizer=optimizer)
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=25)
    history=model.fit(X_train, y_train,validation_data=(X_validation, y_validation), epochs=nb_epochs,
                      batch_size=batch_size,
                      callbacks=[callback_sans_enregistrement], verbose=0, shuffle=False)
    val_loss_score=history.history['val_loss'][-1]
    if tracé_loss==1:
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.legend(["train", "validation"], loc="upper right")
        plt.show()
    return model, val_loss_score
```

```python
# Prédiction de la valeur suivante

def forecast(model, history, window_size):
    data = numpy.array(history)
    data = data.reshape((len(history), 1))
    input_x = data[-window_size:, 0]
    input_x = input_x.reshape((1, len(input_x), 1))
    yhat = model.predict(input_x, verbose=0)
    return yhat
```

```python
# Comparaison des différentes configurations d'hyperparamètres

train,validation,test=scaled_data[:train_size],scaled_data[train_size:train_size+validation_size],scaled_data[-test_size:]

RMSE_test=[]
best_val_loss=10000 # valeur élevée volontairement afin qu'elle soit remplacée

for n in range(len(cfg)):
    batch_size, alpha, window_size, nodes = cfg[n]
    print("paramètres = ", cfg[n])
    X_train, y_train = create_sequences(scaled_data, debut_train, fin_train, window_size)
    X_validation, y_validation = create_sequences(scaled_data, debut_validation, fin_validation, window_size)
    X_test, y_test  = create_sequences(scaled_data, debut_test, fin_test, window_size)
    model, val_loss_score=model_CNN(cfg[n], X_train, y_train, X_validation, y_validation,0)
    print(val_loss_score)
    if val_loss_score< best_val_loss :
        best_cfg=cfg[n]
        best_val_loss=val_loss_score
        X_test_model, y_test_model= X_test, y_test
        best_model=model
        # Enregistrer le modèle de la meilleure configuration
        best_model.save('modele_1_CNN.h5')

print('la meilleure configuration de paramètres est : ', best_cfg, "val_loss=", best_val_loss)
```

```
# Prédictions avec le modèle de la meilleure configuration

debut_test, fin_test = debut_grand_debit[-test_size:],fin_grand_debit[-test_size:]

batch_size, alpha, window_size, nodes = best_cfg
best_model=load_model('modele_1_CNN.h5')

for y in range(len(debut_test)):
    data_walk_forward_prediction=scaled_data[debut_test[y]:fin_test[y]-window_size]
    predictions = list()
    for i in range(0,window_size):
        yhat_sequence=forecast(best_model, data_walk_forward_prediction,window_size)
        predictions=[]
        predictions.append(yhat_sequence[0][0])
        predictions=numpy.array(predictions).reshape(-1,1)
        data_walk_forward_prediction=numpy.concatenate((data_walk_forward_prediction,predictions), axis=0)

    # Dénormaliser les données
    data_walk_forward_prediction_inverted=scaler.inverse_transform(data_walk_forward_prediction)
    y_test_init=poids_brut[debut_test[y]:fin_test[y]+1].reset_index(drop=True)

    # Tracés
    plt.plot(y_test_init)
    plt.plot(data_walk_forward_prediction_inverted)
    plt.legend(["initiales", "prédites"], loc="upper right")
    plt.show()

    # Calcul de l'erreur quadratique moyenne (RMSE) des valeurs prédites
    RMSE_prediction=sqrt(mean_squared_error(y_test_init[-window_size:],data_walk_forward_prediction_inverted[-window_size:]))
    print("RMSE_prediction = ",RMSE_prediction)
```

- Modèle 2 : Vérification des critères de prédiction et de durée avec un réseau CNN

```
#Appel des bibliothèques

import numpy  # Pour les opérations sur les tableaux et les calculs numériques
import pandas  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Lire les fichiers CSV

from sklearn.preprocessing import MinMaxScaler  # Pour normaliser les données
import matplotlib.pyplot as plt  # Créer des graphiques
from math import sqrt  # Ccalculer la racine carrée
from sklearn.metrics import mean_squared_error  # Evaluer les modèles

import tensorflow as tf  # Pour la création et l'entraînement de modèles de machine learning

from keras.models import Sequential  # Créer des modèles séquentiels
from keras.optimizers import Adam  # Importer l'optimiseur Adam
from keras.models import load_model  # Charger des modèles sauvegardés

from keras.models import Model  # Pour créer des modèles plus complexes
from keras.layers import Input, Dense, LSTM  # Importation des couches Input, Dense et LSTM
from keras.layers import LayerNormalization, MultiHeadAttention, Dropout, Add  # Importation de couches avancées
from keras.layers import Conv1D, MaxPooling1D, Flatten  #Couches Conv1D, MaxPooling1D et Flatten pour les CNN
```

```
#Téléchargement de toutes les données

name=['flux','bulk_cutoff','dribble_cutoff','time','step','f','g','filter_weight','open_percentage',
      'empty_activation','raw_weight','tare']

dataset_complet=read_csv('donnees_concat', sep=';', header=None, names=name)

phase=dataset_complet.step
poids_brut=dataset_complet.raw_weight
poids_filtre=dataset_complet.filter_weight
temps=dataset_complet.time
```

```
# Hyperparamètres du modèle :
alpha = [0.001, 0.002, 0.005, 0.0001]  # Taux d'apprentissage pour l'optimisation du modèle
batch_size = [16, 32, 64, 128]  # Tailles de lots pour l'entraînement du modèle
nodes = [16, 32, 64, 128]  # Nombres de neurones pour les couches du modèle

nb_epochs = 100  # Nombre d'époques pour l'entraînement du modèle

window_size = [100, 50, 20, 10]  # Tailles de fenêtres pour les séquences de données d'entrée

# Taille des ensembles d'entraînement et de validation :
train_size = 0.75  # Proportion des données utilisées pour l'entraînement
validation_size = 0.15  # Proportion des données utilisées pour la validation
```

```
# Trouver les durées de phases
def valeurs_des_index(dataset_phase,numero_phase,a,b):
    debut_phase,fin_phase = [],[]
    index= [i for i in range(a,b) if dataset_phase[i] == numero_phase]
    length = len(index)
    i = 0
    while (i< length):
        low = index[i]
        while i <length-1 and index[i]+1 == index[i + 1]:
            i += 1
        high = index[i]
        if (high - low >= 1):
            debut_phase.append(low)
            fin_phase.append(high)
        else:
            debut_phase.append(high)
            fin_phase.append(low)
        i += 1
    return debut_phase,fin_phase

def valeur_correspondante(liste_index,data_correspondante):
    valeur_dataset=[]
    for i in range (0,len(liste_index)):
        x=liste_index[i]
        valeur_dataset.append(data_correspondante[x])
    return valeur_dataset

def delta_des_donnees(debut_phase,fin_phase):
    delta=[]
    for i in range(len(debut_phase)):
        delta.append(fin_phase[i]-debut_phase[i])
    return delta
```

```
# Fournir la liste des configurations de paramètres testées
def model_configs(parametre_1,parametre_2, parametre_3, parametre_4):
    # create configs
    configs = list()
    for i in parametre_1:
        for j in parametre_2:
            for k in parametre_3:
                for l in parametre_4:
                    cfg = [i, j, k, l]
                    configs.append(cfg)
    print('Total configs: %d' % len(configs))
    return configs

# Afficher les différentes configurations de paramètres testées
config=model_configs(alpha, batch_size,nodes,window_size)
print("liste des configurations = ",config)


# Obtenir les index des phases utilisées :

debut_3, fin_3 = valeurs_des_index(phase, 3, 0, len(phase))
debut_1, fin_1 = valeurs_des_index(phase, 1, 0, len(phase))
debut_4, fin_4 = valeurs_des_index(phase, 4, 0, len(phase))
debut_8, fin_8 = valeurs_des_index(phase, 8, 0, len(phase))

debut_grand_debit = debut_1
fin_grand_debit = fin_3
```

```
# Entrées :
def create_sequences(data_entree, fin_phase, window_size):
    X = []
    for i in (fin_phase):
        X.append(data_entree[i-window_size+1:i+1])
    return numpy.array(X)

data=poids_brut.to_frame()
scaler_brut = MinMaxScaler()
scaled_poids_brut = scaler_brut.fit_transform(data)

# Sortie indiquant le poids final :
poids_final_filtre=valeur_correspondante(fin_8, poids_filtre)

# Sortie indiquant le temps restant :
temps_debut_4=valeur_correspondante(debut_4, temps)
temps_fin_8=valeur_correspondante(fin_8, temps)
temps_restant=delta_des_donnees(temps_debut_4, temps_fin_8)


# Assembler les deux sorties et les normaliser :
scaler_poids=MinMaxScaler()
Y1=pandas.DataFrame(poids_final_filtre)
scaled_Y1 = scaler_poids.fit_transform(Y1)

scaler_temps=MinMaxScaler()
Y2=pandas.DataFrame(temps_restant)
scaled_Y2 = scaler_temps.fit_transform(Y2)

data_sorties=numpy.concatenate((scaled_Y1,scaled_Y2), axis=1)
```

```
# Séparer les ensembles d'entraînement, validation, test:

def create_dataset(train_size, validation_size, data_entrees, data_sorties):
    n_features=1
    len_train=int(train_size*len(data_entrees))
    len_validation=int(validation_size*len(data_entrees))

    X_train=data_entrees[0:len_train]
    X_train=X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))

    X_validation=data_entrees[len_train:len_train+len_validation]
    X_validation=X_validation.reshape((X_validation.shape[0], X_validation.shape[1], n_features))

    X_test=data_entrees[len_train+len_validation:]
    X_test=X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))

    Y_train=data_sorties[0:len_train]
    Y_validation=data_sorties[len_train:len_train+len_validation]
    Y_test=data_sorties[len_train+len_validation:]

    return X_train, X_validation, X_test, Y_train, Y_validation, Y_test
```

```
# Création du réseau CNN

def model_CNN(configuration, tracé_loss):
    n_features=1
    alpha, batch_size, nodes, window_size = configuration
    X=create_sequences(scaled_poids_brut, fin_3, window_size)
    X_train, X_validation, X_test, Y_train, Y_validation, Y_test= create_dataset(0.75, 0.15, X ,data_sorties)
    optimizer= Adam(learning_rate=alpha)
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=25)
    input_layer = Input(shape=(window_size, n_features))
    x = Conv1D(nodes, 3, activation='relu')(input_layer)
    x = MaxPooling1D(1)(x)
    x = Conv1D(nodes, 3, activation='relu')(x)
    x = MaxPooling1D(1)(x)
    x = Flatten()(x)
    x = Dense(2, activation='relu')(x)
    output_layer = Dense(2, activation='relu')(x)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(loss='mse', optimizer=optimizer)
    history=model.fit(X_train,Y_train,validation_data=(X_validation, Y_validation),epochs=nb_epochs,batch_size=batch_size,
                      callbacks=[callback_sans_enregistrement], verbose=0, shuffle=True)
    val_loss_score=history.history['val_loss'][0]
    if tracé_loss==1:
        plt.plot(history.history["loss"])
        plt.plot(history.history['val_loss'])
        plt.legend(["train", "validation"], loc="upper right")
        plt.show()
    return model, val_loss_score, X_test, Y_test
```

```
# Comparaison des différentes configurations d'hyperparamètres

best_val_loss=10000 # valeur élevée volontairement afin qu'elle soit remplacée

for i in range(len(config)):
    model, val_loss_score, X_test, Y_test=model_CNN(config[i], 0)
    print("paramètres testés = ",config[i])
    print("val_loss_score = ",val_loss_score, '\n')
    if val_loss_score<best_val_loss:
        best_val_loss=val_loss_score
        X_test_model= X_test
        best_cfg=config[i]
        best_model=model
        # Enregistrer le modèle de la meilleure configuration
        best_model.save('modele_2_CNN.h5')

print("Les meilleurs paramètres sont : ", best_cfg)

# Prédictions le modèle de la meilleure configuration

best_model=load_model('modele_2_CNN.h5')
print(best_model)

pred=best_model.predict(X_test_model)
```

```
# Valeurs prédites dénormalisées :
poids_pred_s=pred[:,0]
poids_pred=scaler_poids.inverse_transform(poids_pred_s.reshape(-1,1))
print("poids prédit= \n",poids_pred, '\n')

temps_pred_s=pred[:,1]
temps_pred=scaler_temps.inverse_transform(temps_pred_s.reshape(-1,1))
print("temps prédit= \n",temps_pred, '\n')

# Valeurs initiales dénormalisées :
poids_init_s=Y_test[:,0]
poids_init=scaler_poids.inverse_transform(poids_init_s.reshape(-1,1))
print("poids initial= \n",poids_init, '\n')

temps_init_s=Y_test[:,1]
temps_init=scaler_temps.inverse_transform(temps_init_s.reshape(-1,1))
print("temps initial= \n",temps_init, '\n')

# Calcul de l'erreur quadratique moyenne (RMSE) pour le poids
rmse_poids = sqrt(mean_squared_error(poids_init, poids_pred))
print("RMSE poids final =", rmse_poids)

# Calcul de l'erreur quadratique moyenne (RMSE) pour le temps
rmse_temps = sqrt(mean_squared_error(temps_init, temps_pred))
print("RMSE temps restant =", rmse_temps)
```

# ANNEXE C – CODES DE L'OPTIMISATION EN DEUX ÉTAPES : FILTRES NUMÉRIQUES

```
# Appel des bibliothèques

import numpy  # Pour les opérations sur les tableaux et les calculs numériques
import pandas  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Pour lire les fichiers CSV
import matplotlib.pyplot as plt  # Pour créer des graphiques
from math import sqrt  # Pour calculer la racine carrée

import scipy  # Pour les fonctions et algorithmes scientifiques supplémentaires
from scipy.signal import savgol_filter  # Pour appliquer un filtre de Savitzky-Golay
from scipy.signal import butter, filtfilt  # Pour appliquer des filtres Butterworth
```

```
# Téléchargement de toutes les données

name=['flux','bulk_cutoff','dribble_cutoff','time','step','f','g','filter_weight','open_percentage',
      'empty_activation','raw_weight','tare']

dataset_complet=read_csv('donnees_concat', header=None, names=name).dropna()

phase=dataset_complet.step
poids_brut=dataset_complet.raw_weight
poids_filtre=dataset_complet.filter_weight

window_size=100
decalage = 115 # Permet d'aligner le signal brut avec le signal filtré de la balance

# Définir la phase étudiée ou la pesée complète :

debut_1, fin_1 = valeurs_des_index(phase, 1, 0, len(phase))
debut_8, fin_8 = valeurs_des_index(phase, 8, 0, len(phase))

debut_signal_a_filtrer = debut_1
fin_signal_a_filtrer = fin_8

signal_a_filtrer = poids_brut[debut_signal_a_filtrer:fin_signal_a_filtrer]

signal_filtre_balance = poids_filtre[debut_signal_a_filtrer+decalage:fin_signal_a_filtrer+decalage]
```

```
#FILTRE DE SAVITZKY-GOLAY

# Appliquer le filtre Savitzky-Golay au signal
window = window_size
poly_order = 1
signal_SG = savgol_filter(signal_a_filtrer, window, poly_order)

# Tracer le signal original et le signal filtré

plt.plot(signal_a_filtrer, label='poids brut')
plt.plot(signal_SG, label='filtre Savgol')
plt.plot(signal_filtre_balance, label='filtré (balance)')
plt.legend()
plt.show()
```

```
#FILTRE DE BUTTERWORTH

# Définir les paramètres du filtre Butterworth
order = 1
fs=627
cutoff =3

# Calculer les coefficients du filtre Butterworth
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='low', analog=False)
signal_butterworth = filtfilt(b, a, signal_a_filtrer)

# Tracer les signaux
plt.plot(signal_a_filtrer, label='poids brut')
plt.plot(signal_butterworth, label='Butterworth Filter')
plt.plot(signal_filtre_balance, label='filtré (balance)')
plt.legend()
plt.show()
```

```
# FILTRE DE MOYENNE MOBILE

# Appliquer le filtre de moyenne mobile au signal
window = window_size
signal_moyenne_mobile = numpy.convolve(signal_a_filtrer, numpy.ones(window)/window, mode='same')


plt.plot(signal_a_filtrer, label='Signal prédit')
plt.plot(signal_moyenne_mobile, label='Moving Average')
plt.plot(signal_filtre_balance, label='filtré (balance)')
plt.legend()
plt.show()
```

```
# FILTRE DE MOYENNE MOBILE EXPONENTIELLE

# Appliquer le filtre de moyenne mobile exponentielle au signal
window = window_size
signal_exponentielle_MA = pd.Series(signal_a_filtrer).ewm(span=window, adjust=False).mean()

# Tracer le signal original et le signal filtré
import matplotlib.pyplot as plt

plt.plot(signal_a_filtrer, label='Signal prédit')
plt.plot(signal_exponentielle_MA, label='Exponential MA')
plt.plot(signal_filtre_balance, label='filtré (balance)')
plt.legend()
plt.show()
```

```
#FILTRE MEDIANE

# Appliquer le filtre de médiane au signal
window = window_size+1 # Doit être impaire
signal_mediane = scipy.signal.medfilt(signal_a_filtrer,window)


plt.plot(signal_a_filtrer, label='Signal prédit')
plt.plot(signal_mediane, label='Median Filter')
plt.plot(signal_filtre_balance, label='filtré (balance)')
plt.legend()
plt.show()
```

# ANNEXE D – CODES DE L'OPTIMISATION EN DEUX ÉTAPES : APPRENTISSAGE AUTOMATIQUE

```
# Appel des bibliothèques

import numpy as np  # Pour les opérations sur les tableaux et les calculs numériques
import pandas  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Pour lire les fichiers CSV
from math import sqrt  # Pour calculer la racine carrée

from sklearn.metrics import mean_squared_error  # Pour évaluer les modèles
from sklearn.preprocessing import MinMaxScaler  # Pour normaliser les données

from sklearn.svm import SVR  # Modèle Support Vector Regression (SVR) pour la régression
import xgboost as xgb # Modèle XGBoost

import pickle  # Pour la sérialisation et la désérialisation des objets Python

pandas.options.plotting.backend = "plotly"  # Tracer des graphiques
```

```
# Téléchargement de toutes les données

name=['flux','bulk_cutoff','dribble_cutoff','time','step','f','g','filter_weight','open_percentage',
      'empty_activation','raw_weight','tare']

dataset_complet=read_csv('donnees_concat', sep=',', names=name).dropna()

phase=dataset_complet.step
poids_brut=dataset_complet.raw_weight
poids_filtre=dataset_complet.filter_weight

data=poids_brut.to_frame()
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

data2=poids_filtre.to_frame()
scaler2 = MinMaxScaler()
scaled_filtre = scaler2.fit_transform(data2)
```

```
# Trouver les valeurs de phases
def valeurs_des_index(dataset_phase,numero_phase,a,b):
    debut_phase,fin_phase = [],[]
    index= [i for i in range(a,b) if dataset_phase[i] == numero_phase]
    length = len(index)
    i = 0
    while (i< length):
        low = index[i]
        while i <length-1 and index[i]+1 == index[i + 1]:
            i += 1
        high = index[i]
        if (high - low >= 1):
            debut_phase.append(low)
            fin_phase.append(high)
        else:
            debut_phase.append(high)
            fin_phase.append(low)
        i += 1
    return debut_phase,fin_phase


def duree_des_phases(debut_phase,fin_phase):
    delta=[]
    for i in range(len(debut_phase)):
        delta.append(fin_phase[i]-debut_phase[i])
    return delta
```

```
# Définir la phase étudiée :
debut_1, fin_1 = valeurs_des_index(phase, 1, 0, len(phase))
debut_3, fin_3 = valeurs_des_index(phase, 3, 0, len(phase))

debut_grand_debit = debut_1
fin_grand_debit = fin_3
```

```
# Taille des ensembles d'entrainement, de validation et de test

proportion_train=0.1
proportion_validation=0.1975

train_size = int(proportion_train * len(fin_grand_debit))
validation_size = int(proportion_validation * len(fin_grand_debit))
test_size = len(fin_grand_debit)-train_size-validation_size

print("train_size = ", train_size)
print("validation_size = ", validation_size)
print("test_size=", test_size)

debut_train, fin_train = debut_grand_debit[0:train_size],fin_grand_debit[0:train_size]
debut_validation = debut_grand_debit[train_size:train_size+validation_size]
fin_validation = fin_grand_debit[train_size:train_size+validation_size]
debut_test, fin_test = debut_grand_debit[-test_size:],fin_grand_debit[-test_size:]

nb_donnees_train =duree_des_phases(debut_train,fin_train)
nb_donnees_validation =duree_des_phases(debut_validation,fin_validation)
nb_donnees_test =duree_des_phases(debut_test,fin_test)
```

```
decalage_brut_filtre = 115

# Entrainement
debut_data_train=debut_train[0]
fin_data_train=fin_train[-1]

X_train_all=poids_brut[debut_data_train:fin_data_train]
Y_train_all=poids_filtre[debut_data_train+decalage_brut_filtre:fin_data_train+decalage_brut_filtre]

# Entrées / Sorties
y_noisy=np.array(X_train_all).reshape(-1, 1) # poids brut
X=np.array(Y_train_all).reshape(-1, 1) # poids filtré de la balance


# Validation
debut_data_val=debut_validation[0]
fin_data_val=fin_validation[0]

X_validation_all=poids_brut[debut_data_val:fin_data_val]
Y_validation_all=poids_filtre[debut_data_val+decalage_brut_filtre:fin_data_val+decalage_brut_filtre]

# Entrées / Sorties
X_valid=np.array(X_validation_all).reshape(-1, 1)
y_noisy_valid=np.array(Y_validation_all).reshape(-1, 1)
```

- Modèle SVR :

```
# Ajuster le modèle SVR
svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_rbf.fit(X, y_noisy)

# Enregistrement du modèle
model_filename = 'svm_model.pkl'
pickle.dump(svr_rbf, open(model_filename, 'wb'))
print("Modèle enregistré avec succès !")

# Charger le modèle entraîné
model_filename = 'svm_model.pkl'
loaded_model = pickle.load(open(model_filename, 'rb'))

# Prédiction du signal débruité
y_pred = loaded_model.predict(X_valid)

#Tracé des prédictions
df = pandas.DataFrame(dict(brut_test=X_valid[:,0], filtre_test=y_noisy_valid[:,0], SVM_test=y_pred))
fig = df.plot.line()
fig.show()
```

- Modèle XGBoost :

```
# Entraînement du modèle XGBoost
model = xgb.XGBRegressor(objective='reg:squarederror')
model.fit(X_train_entiere, Y_train_entiere)

# Enregistrement du modèle
model_filename = 'xgboost_model.pkl'
pickle.dump(model, open(model_filename, 'wb'))
print("Modèle enregistré avec succès !")

# Charger le modèle entraîné
model_filename = 'xgboost_model.pkl'
loaded_model = pickle.load(open(model_filename, 'rb'))

# Prédiction du signal débruité
y_pred = loaded_model.predict(X_validation_entiere)


#Tracé des prédictions
df = pandas.DataFrame(dict(brut_test=X_valid[:,0], filtre_test=y_noisy_valid[:,0], SVM_test=y_pred))
fig = df.plot.line()
fig.show()
```

# ANNEXE E – CODES DE L'OPTIMISATION EN DEUX ÉTAPES : APPRENTISSAGE PROFOND

```
# Appel des bibliothèques

import numpy  # Pour les opérations sur les tableaux et les calculs numériques
import pandas  # Pour la manipulation et l'analyse des données
from pandas import read_csv  # Pour lire les fichiers CSV
import matplotlib.pyplot as plt  # Pour créer des graphiques
from math import sqrt  # Pour calculer la racine carrée

from sklearn.metrics import mean_squared_error  # Pour évaluer les modèles en calculant l'erreur quadratique moyenne
from sklearn.preprocessing import MinMaxScaler  # Pour normaliser les données

import tensorflow as tf  # Pour créer et entraîner des modèles de machine learning

import time  # Pour gérer les opérations liées au temps
from tensorflow.keras.callbacks import Callback  # Pour créer des callbacks personnalisés pour l'entraînement du modèle
from tensorflow.keras.models import Model, load_model  # Pour créer et charger des modèles
from tensorflow.keras.layers import Input, Dense  # Pour définir les couches d'entrée et denses du modèle
from tensorflow.keras.layers import Add  # Pour ajouter des couches dans le modèle
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten  # Pour les couches convolutives et de pooling
from tensorflow.keras.optimizers import Adam  # Pour l'optimiseur Adam
```

```
# Téléchargement des données

name=['time','step','filter_weight','raw_weight']

dataset_complet=read_csv('donnees_concat_colonnes_utiles', sep=',',header=0)

phase=dataset_complet.step
poids_brut=dataset_complet.raw_weight
poids_filtre=dataset_complet.filter_weight

# Normalisation des données

data=poids_brut.to_frame()
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

data2=poids_filtre.to_frame()
scaler2 = MinMaxScaler()
scaled_filtre = scaler2.fit_transform(data2)
```

```
# Hyperparamètres du modèle :

alpha=[0.0001, 0.001]
batch_size=[256, 128]
nb_filters=[128]
kernel_size=[5]
pooling_size=[2]

nb_epochs=100

window_size=[20, 40, 60, 80, 100, 150]

LSTM_nodes = [32, 64]

decalage = 115

proportion_train = 0.8
proportion_validation = 0.1975
```

```
# Fournir la liste des configurations de paramètres testées

def model_configs(parametre_1,parametre_2, parametre_3, parametre_4, parametre_5, parametre_6):
    # create configs
    configs = list()
    for i in parametre_1:
        for j in parametre_2:
            for k in parametre_3:
                for l in parametre_4:
                    for m in parametre_5:
                        for n in parametre_6:
                            cfg = [i, j, k, l,m, n]
                            configs.append(cfg)
    print('Total configs: %d' % len(configs))
    return configs

# Affichage des différentes configurations de paramètres testées
cfg= model_configs(batch_size, alpha, window_size, nb_filters,kernel_size,pooling_size )
print("liste des configurations = ",cfg)
```

```
# Trouver les valeurs de phases
def valeurs_des_index(dataset_phase,numero_phase,a,b):
    debut_phase,fin_phase = [],[]
    index= [i for i in range(a,b) if dataset_phase[i] == numero_phase]
    length = len(index)
    i = 0
    while (i< length):
        low = index[i]
        while i <length-1 and index[i]+1 == index[i + 1]:
            i += 1
        high = index[i]
        if (high - low >= 1):
            debut_phase.append(low)
            fin_phase.append(high)
        else:
            debut_phase.append(high)
            fin_phase.append(low)
        i += 1
    return debut_phase,fin_phase
```

```
# Définir la phase étudiée

debut_1, fin_1 = valeurs_des_index(phase, 1, 0, len(phase))
debut_3, fin_3 = valeurs_des_index(phase, 3, 0, len(phase))


debut_grand_debit = debut_1
fin_grand_debit = fin_3

decalage = 115

# Taille des datasets :

train_size = int(proportion_train * len(fin_grand_debit))
validation_size = int(proportion_validation * len(fin_grand_debit))
test_size = len(fin_grand_debit)-train_size-validation_size

print("train_size = ", train_size)
print("validation_size = ", validation_size)
print("test_size=", test_size)

debut_train, fin_train = debut_grand_debit[0:train_size],fin_grand_debit[0:train_size]
debut_validation = debut_grand_debit[train_size:train_size+validation_size]
fin_validation = fin_grand_debit[train_size:train_size+validation_size]
debut_test, fin_test = debut_grand_debit[-test_size:],fin_grand_debit[-test_size:]
```

```python
# Créer les entrées/sorties, fenêtres glissantes
def create_sequences(data_entree,data_sortie, debut, fin, window_size):
    X = []
    y = []
    for t in range(len(debut)):
        for i in range(debut[t],fin[t]-window_size+1):
            X.append(data_entree[i:i+window_size])
            y.append(data_sortie[i+window_size+decalage])
    return numpy.array(X), numpy.array(y)


# Créer les entrées/sorties, fenêtres non glissantes
def create_sequences_no_sliding(data_entree, data_sortie, debut, fin,nb_donnees_data, window_size):
    X = []
    y = []

    for t in range (len(nb_donnees_data)):
        j_iteration=(int(nb_donnees_data[t]/window_size))
        data_ent=data_entree[debut[t]:fin[t]]
        data_sor=data_sortie[debut[t]+decalage:fin[t]+decalage]
        for i in range(0,j_iteration):
            X.append(data_ent[i*window_size:(i+1)*window_size])
            y.append(data_sor[i*window_size:(i+1)*window_size])

    X = numpy.array(X)
    y = numpy.array(y)

    # Redimensionner les séquences pour qu'elles puissent être utilisées avec le réseau de neurones
    X = numpy.reshape(X, (X.shape[0], window_size,1))
    y = numpy.reshape(y, (y.shape[0], window_size,1))

    return X, y
```

```python
# Classe pour mesurer et enregistrer la durée de chaque époque durant l'entraînement
class TimingCallback(Callback):


    def __init__(self, log_file_path):
        super().__init__()
        self.log_file_path = log_file_path  # Chemin du fichier de journalisation pour enregistrer les temps d'époque

    def on_epoch_begin(self, epoch, logs=None):
        self.start_time = time.time()  # Enregistre le temps de début de l'époque

    def on_epoch_end(self, epoch, logs=None):
        epoch_time = time.time() - self.start_time  # Calcule la durée de l'époque
        logs['epoch_time'] = epoch_time  # Ajoute la durée de l'époque aux logs

        with open(self.log_file_path, 'a') as f:
            # Enregistre la durée de l'époque
            f.write('Epoch {}: {:.2f} seconds\n'.format(epoch, epoch_time))
```

- Réseau CNN :

```
# Réseau CNN
base_filename2='training_B-F_phase_1-3_CNN'
base_filename3='temps_B-F_phase_1-3_CNN'
base_filename4='plot_B-F_phase_1-3_CNN'
extension_csv='.csv'
extension_pdf='.pdf'

val_loss_score=[]

def model_CNN(configuration, X_train, y_train, X_validation, y_validation, trace_loss):
    n_features=1
    batch_size, alpha, window_size, nb_filters,kernel_size,pooling_size = configuration
    filepath='B-F_ph1-3_CNN_after_{epoch:d}epoch-b%d_a%.4f_w%d_f%d_k%d_p%d.h5' % (batch_size, alpha, window_size,
                                                                                  nb_filters,kernel_size,pooling_size)
    optimizer= Adam(learning_rate=alpha, clipvalue=1.0)
    logger=tf.keras.callbacks.CSVLogger(f'{base_filename2}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_
                                        K{kernel_size}_P{pooling_size}_{n}{extension_csv}')
    input_layer = Input(shape=(window_size, n_features))
    x = Conv1D(nb_filters, kernel_size, activation='relu')(input_layer)
    x = MaxPooling1D(pooling_size)(x)
    x = Conv1D(nb_filters, kernel_size, activation='relu')(x)
    x = MaxPooling1D(pooling_size)(x)
    x = Flatten()(x)
    x = Dense(1, activation='relu')(x)
    output_layer = Dense(1, activation='relu')(x)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(loss='mse', optimizer=optimizer)
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=25)
    history=model.fit(X_train, y_train,validation_data=(X_validation, y_validation), epochs=nb_epochs,batch_size=batch_size,
                      callbacks=[callback_sans_enregistrement,logger,
                                 TimingCallback(f'{base_filename3}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_
                                                K{kernel_size}_P{pooling_size}_{n}{extension_csv}')],
                                                verbose=2, shuffle=False)
```

```
    val_loss_score=history.history['val_loss'][-1]

    if trace_loss==1:
        # Trace la courbe de perte de l'entraînement
        plt.plot(history.history['loss'])
        # Trace la courbe de perte de la validation
        plt.plot(history.history['val_loss'])
        plt.legend(["train", "validation"], loc="upper right")
        # Sauvegarde le graphique avec un nom de fichier formaté incluant plusieurs hyperparamètres
        plt.savefig(f'{base_filename4}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_K{kernel_size}_
                    P{pooling_size}_{n}{extension_pdf}',format="pdf")
        plt.show()
    return model, val_loss_score
```

- Réseau LSTM :

```
# Réseau LSTM
base_filename2='training_B-F_phase_1-3_LSTM'
base_filename3='temps_B-F_phase_1-3_LSTM'
base_filename4='plot_B-F_phase_1-3_LSTM'
extension_csv='.csv'
extension_pdf='.pdf'
val_loss_score=[]

def model_LSTM(configuration, X_train, y_train, X_validation, y_validation, trace_loss):
    n_features=1
    batch_size, alpha, window_size, LSTM_nodes = configuration
    filepath='B-F_ph1-3_LSTM_after_{epoch:d}epoch-b%d_a%.4f_w%d_L%d.h5' % (batch_size, alpha, window_size, LSTM_nodes)
    optimizer= Adam(learning_rate=alpha, clipvalue=1.0)
    logger=tf.keras.callbacks.CSVLogger(f'{base_filename2}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_
                                        {n}{extension_csv}')
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=10)
    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath,save_weights_only=False,save_freq='epoch',
                                                          period=5)
    model= Sequential()
    model.add(LSTM(LSTM_nodes, activation='relu', input_shape=(window_size, n_features)))
    model.add(Dense(window_size))
    model.compile(optimizer=optimizer , loss='mse')
    history=model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=nb_epochs,
                      batch_size=batch_size,max_queue_size=20,workers=8,callbacks=[callback_sans_enregistrement,logger,
                        TimingCallback(f'{base_filename3}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_
                                       {n}{extension_csv}'),model_checkpoint], verbose=2, shuffle=False)
    val_loss_score=history.history['val_loss'][-1]

    if trace_loss==1:
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.legend(["train", "validation"], loc="upper right")
        plt.savefig(f'{base_filename4}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_{n}{extension_pdf}',format="pdf")
        plt.show()

    return model, val_loss_score
```

- Réseau LSTM avec attention

```
# Réseau LSTM avec attention
base_filename2='training_B-F_phase_1-3_AttentionLSTM'
base_filename3='temps_B-F_phase_1-3_AttentionLSTM'
base_filename4='plot_B-F_phase_1-3_AttentionLSTM'
extension_csv='.csv'
extension_pdf='.pdf'
val_loss_score=[]

def model_LSTM_attention(configuration, X_train, y_train, X_validation, y_validation, trace_loss):
    n_features=1
    batch_size, alpha, window_size, LSTM_nodes = configuration
    filepath='B-F_ph1-3_AttentionLSTM_after_{epoch:d}epoch-b%d_a%.4f_w%d_L%d.tf' % (batch_size, alpha, window_size,
                                                                                    LSTM_nodes)
    input_shape=(window_size,n_features)
    optimizer= Adam(learning_rate=alpha, clipvalue=1.0)
    logger=tf.keras.callbacks.CSVLogger(f'{base_filename2}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_
                                        {n}{extension_csv}')
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=10)
    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath,save_weights_only=False,save_freq='epoch',
                                                          period=5)

    x=Input(shape=input_shape)
    LSTM_layer = LSTM(LSTM_nodes, return_sequences=True, activation='relu')(x)
    attention_layer = attention()(LSTM_layer)
    outputs=Dense(100, trainable=True)(attention_layer)
    model=Model(x,outputs)
    model.compile(optimizer=optimizer , loss='mse')
```

```
    history=model.fit(X_train, y_train,validation_data=(X_validation, y_validation), epochs=nb_epochs,
                      batch_size=batch_size,callbacks=[callback_sans_enregistrement,logger,
                        TimingCallback(f'{base_filename3}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_
                                       {n}{extension_csv}'),model_checkpoint], verbose=2, shuffle=False)
    val_loss_score=history.history['val_loss'][-1]

    if trace_loss==1:
        plt.plot(history.history["loss"])
        plt.plot(history.history["val_loss"])
        plt.legend(["train", "validation"], loc="upper right")
        plt.savefig(f'{base_filename4}_b{batch_size}_a{alpha}_w{window_size}_L{LSTM_nodes}_{n}{extension_pdf}',format="pdf")
        plt.show()
    return model, val_loss_score
```

- Réseau encodeur-décodeur CNN avec quatre couches convolutionnelles et MaxPooling :

```
# Réseau encodeur-décodeur CNN avec 4 couches convolutionnelles et MaxPooling
base_filename2='training_B-B_phase_1-3_CNN-4Conv1D'
base_filename3='temps_B-B_phase_1-3_CNN-4Conv1D'
base_filename4='plot_B-B_phase_1-3_CNN-4Conv1D'
extension_csv='.csv'
extension_pdf='.pdf'

val_loss_score=[]

def model_encod_decod_CNN(configuration, X_train, y_train, X_validation, y_validation, trace_loss):
    n_features=1
    batch_size, alpha, window_size, nb_filters,kernel_size,pooling_size = configuration
    filepath='B-B_ph1-3_CNN-4Conv1D_after_{epoch:d}epoch-b%d_a%.4f_w%d_f%d_k%d_p%d.h5' % (batch_size, alpha, window_size,
                                                                                          nb_filters,kernel_size,pooling_size)
    optimizer= Adam(learning_rate=alpha)
    logger=tf.keras.callbacks.CSVLogger(f'{base_filename2}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_
                                        K{kernel_size}_P{pooling_size}_{n}{extension_csv}')
    # Define the Transformer model architecture
    input_layer = Input(shape=(window_size, n_features))
    x = Conv1D(nb_filters, kernel_size, activation='relu', padding='same')(input_layer)
    x = MaxPooling1D(pooling_size)(x)
    x = Conv1D(nb_filters, kernel_size, activation='relu', padding='same')(x)
    x = MaxPooling1D(pooling_size)(x)
    x = Conv1D(nb_filters, kernel_size, activation='relu', padding='same')(x)
    x = MaxPooling1D(pooling_size)(x)
    x = Conv1D(nb_filters, kernel_size, activation='relu', padding='same')(x)
    x = MaxPooling1D(pooling_size)(x)
    x = Flatten()(x)
```

```
    output_layer = Dense(window_size, activation='relu')(x)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(loss='mse', optimizer=optimizer)
    print(model.summary())
    callback_sans_enregistrement = tf.keras.callbacks.EarlyStopping(monitor='val_loss',verbose=1, patience=10)
    model_checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath,save_weights_only=False,save_freq='epoch',
                                                          period=5)
    history=model.fit(X_train, y_train,validation_data=(X_validation, y_validation), epochs=nb_epochs,
                      batch_size=batch_size,
                      callbacks=[callback_sans_enregistrement,logger,TimingCallback(f'{base_filename3}_
                                 b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_K{kernel_size}_P{pooling_size}_
                                 {n}{extension_csv}'),model_checkpoint], verbose=2, shuffle=False)
    val_loss_score=history.history['val_loss'][-1]

    if trace_loss==1:
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.legend(["train", "validation"], loc="upper right")
        plt.savefig(f'{base_filename4}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_K{kernel_size}_P{pooling_size}
                                                  _{n}{extension_pdf}',format="pdf")
        plt.show()
    return model, val_loss_score
```

```
# Teste les différentes combinaisons d'hyperparamètres pour trouver la meilleure configuration

RMSE_test=[]
best_val_loss=10000

base_filename='B-F_phase_1-3_LSTM'
extension='.h5'

for n in range(len(cfg)):
    batch_size, alpha, window_size, nodes, kernel_size,pooling_size = cfg[n]
    print("paramètres = ", cfg[n])
    X_train, y_train = create_sequences(scaled_data,scaled_filtre, debut_train, fin_train, window_size)
    X_validation, y_validation = create_sequences(scaled_data,scaled_filtre, debut_validation, fin_validation, window_size)
    X_test, y_test  = create_sequences(scaled_data,scaled_filtre, debut_test, fin_test, window_size)
    model, val_loss_score=model_CNN(cfg[n], X_train, y_train, X_validation, y_validation,1)
    model.save(f'{base_filename}_b{batch_size}_a{alpha}_w{window_size}_F{nb_filters}_K{kernel_size}_P{pooling_size}_
               {n}{extension}')
    print(val_loss_score)
    if val_loss_score< best_val_loss :
        best_cfg=cfg[n]
        best_val_loss=val_loss_score
        X_test_model, y_test_model= X_test, y_test

print('la meilleure configuration de paramètres est : ', best_cfg, "val_loss=", best_val_loss)
```

# ANNEXE F – CODES DE L'OPTIMISATION EN DEUX ÉTAPES : APPRENTISSAGE PAR RENFORCEMENT

L'environnement créé permet de simuler les cycles d'ensachage, comme lors d'une exécution en temps réel. Les données sont transmises au fur et à mesure à la commande (ou l'agent dans ce cas), puis elles sont filtrées et une action est réalisée en fonction de la masse atteinte.

```
# ACTION 1 = grand débit = regroupe les phases 1 et 3
# ACTION 2 = changement de débit = concerne uniquement la phase 4
# ACTION 3 = petit débit = regroupe les phases 5 et 6
# ACTION 4 = fermeture = concerne uniquement la phase 7
# ACTION 5 = stabilisation = concerne uniquement la phase 8
# ACTION 0 = évacuation du produit + remise à zéro = concerne uniquement la phase 0

#Appel des différentes bibliothèques
import pandas as pd
import queue
import threading
import time
import random
import matplotlib.pyplot as plt
import numpy as np
import csv
import re
import logging
import os
import math

import sys
print(sys.prefix)
```

```
def generate_complete_filename(folder_path, prefix, max_episode_number):
    """
    Génère une liste de chemins de fichiers complets uniques choisis aléatoirement qui commencent par un préfixe
    spécifié dans un dossier donné.
    La liste générée ne comprendra pas plus de files que le nombre maximum d'épisodes autorisé.

    Args:
    folder_path (str): Le chemin du dossier où rechercher les files.
    prefix (str): Le préfixe identifiant les files à prendre en compte pour la génération des noms.
    max_nombre_episode (int): Le nombre maximal d'épisodes, limitant ainsi le nombre de fichiers retournés.

    Returns:
    Iterator[str]: Un générateur qui retourne des chemins de fichiers complets uniques, limitée par le nombre maximum
    d'épisodes.
    """
    # Liste tous les fichiers dans le dossier qui commencent par le préfixe spécifié
    files = [f for f in os.listdir(folder_path) if f.startswith(prefix)]
    print(f"Nombre total de fichiers trouvés: {len(files)}")

    # Mélange la liste de fichiers pour aléatoire
    random.shuffle(files)

    # Limite la liste des fichiers au nombre maximal d'épisodes autorisé, après mélange
    files = files[:min(max_episode_number, len(files))]

    # Retourne le chemin complet de chaque fichier sélectionné
    for _file in files:
        yield os.path.join(folder_path, _file)
```

```
# Définir les paramètres de l'environnement
STATE_DIM = 1  # Nombre de dimensions de l'état de l'environnement
ACTION_DIM = 6  # Nombre d'actions possibles
MAX_EPISODES = 8000 # Nombre d'épisodes d'apprentissage
MEMORY_SIZE = 100000  # Taille de la mémoire de replay
BATCH_SIZE = 64  # Taille du batch pour l'apprentissage
GAMMA = 0.9  # Taux d'actualisation des récompenses
EPSILON = 1.0  # Probabilité d'exploration initiale
EPSILON_DECAY = 0.99885 # Taux de déclin de la probabilité d'exploration
EPSILON_MIN = 0.01  # Probabilité d'exploration minimale
MIN_EXPLORATION_STEPS = 20000 # Nombre minimum de pas à explorer
LEARNING_RATE = 0.001  # Taux d'apprentissage pour l'optimiseur
TARGET_UPDATE_FREQ = 100 # Fréquence de mise à jour du réseau cible

ACTION_BULK = 1
ACTION_BULK_TO_DRIBBLE = 2
ACTION_DRIBBLE = 3
ACTION_DRIBBLE_TO_CLOSE = 4
ACTION_STABILIZATION = 5
ACTION_INIT = 0

VALUE_FILE_BULK_TO_DRIBBLE = 4
VALUE_FILE_DRIBBLE_TO_CLOSE = 7
BULK_FEED_BLANKING_TIME = 100

simulator_data = {
    'time_step': [],
    'phase': [],
    'raw_weight': [],
    'extrapolation': []
}
```

```
DQNAgent_data = {
'time_step': [],
'phase': [],
'raw_weight': [],
'filtered_weight': [],
'reward': []
}


def reset_Dictionaries():
    global simulator_data
    global DQNAgent_data

    simulator_data = {
        'time_step': [],
        'phase': [],
        'raw_weight': [],
        'extrapolation': []
    }

    DQNAgent_data = {
    'time_step': [],
    'phase': [],
    'raw_weight': [],
    'filtered_weight': [],
    'reward': []
    }
```

```python
import queue  # Importation du module queue pour utiliser la classe Queue

class Buffer:
    def __init__(self):
        self.buffer = queue.Queue()  # Initialise une instance de Queue pour stocker les éléments

    def add(self, item):
        self.buffer.put(item)  # Ajoute un élément au tampon

    def remove(self):
        try:
            item = self.buffer.get(timeout=1)  # Retire et retourne un élément de la file
            self.buffer.task_done()  # Marque l'élément comme traité
            return item
        except queue.Empty:
            return None  # Retourne None si la file est vide

    def is_empty(self):
        return self.buffer.empty()  # Retourne True si la file est vide, sinon False
```

```python
from scipy.signal import butter, filtfilt  # Importation des fonctions nécessaires pour le filtre Butterworth

class ButterworthFilter:
    def __init__(self, order, cutoff, fs, btype='low'):
        self.order = order  # Ordre du filtre
        self.cutoff = cutoff  # Fréquence de coupure en Hz
        self.fs = fs  # Fréquence d'échantillonnage en Hz
        self.btype = btype  # Type de filtre : 'low', 'high', 'bandpass', 'bandstop'
        # Calcul des coefficients du filtre
        self.b, self.a = butter(self.order, self.cutoff / (0.5 * fs), btype=self.btype, analog=False)

    def filter_signal(self, signal):
        return filtfilt(self.b, self.a, signal)  # Application du filtre au signal
```

- L'environnement (simulateur) :

```python
class WeighingSimulator(threading.Thread):

    def __init__(self, buffer, window_size=25, toprint=True):
        super().__init__()
        self.buffer = buffer
        self.active_weighing = False
        self.window_size = window_size
        self.flow_meter = 0
        self.running = True

        self.toprint = toprint
        self.current_weight = 0.0
        self.time_bulk_to_dribble = 0
        self.time_dribble_to_close = 0
        self.time_stabilization = 0

    def stop_simulator(self):
        print("Arrêter le simulateur", flush=True)
        self.running = False

    def run(self):
        while self.running:
            if self.active_weighing:
                self.send_to_buffer()
                time.sleep(0.5) #time.sleep(1)
            else:
                time.sleep(5)

    def stop_weighing(self):
        print(f"Arrêt de la pesée {self.complete_weighing_name}", flush=True)
        self.active_weighing = False
```

```python
def start_weighing(self, complete_weighing_name):
    while (not self.buffer.is_empty()):
        time.sleep(1)
    print(f"Démarrage de la pesée {complete_weighing_name}...", flush=True)
    if (self.toprint):
        print(f"Buffer vide {self.buffer.is_empty()}", flush=True)
    file_loaded = self.load_excel_data(complete_weighing_name)
    if not file_loaded:
        return False
    self.phase = 1
    self.action = 1
    self.data_index = 0
    self.extrapolation = False
    self.last_filtered_weight = 0
    self.extrapolated_weight = 0
    self.extrapolation_index = 0
    self.extrapolation_flow = 0
    self.active_weighing = True
    self.weight_bulk = 0.0
    self.weight_dribble = 0.0
    self.stabilization = 0.0
    return True
```

```python
def load_excel_data(self,complete_weighing_name):
    # Déendir le chemin du dossier de logs dans le répertoire courant
    current_directory = os.getcwd()
    log_directory = os.path.join(current_directory, 'train', 'error')
    log_filename = 'unloaded_weighings.log'
    log_filepath = os.path.join(log_directory, log_filename)
    # Créer le dossier s'il n'existe pas
    if not os.path.exists(log_directory):
        os.makedirs(log_directory)

    # Configuration du logging
    logging.basicConfig(
        filename=log_filepath,  # Chemin du fichier de log
        filemode='a',           # Mode d'écriture 'a' pour ajouter, 'w' pour écraser
        format='%(asctime)s - %(levelname)s - %(message)s',  # Format du message
        level=logging.DEBUG     # Niveau de log
    )
    try:
        self.complete_weighing_name = complete_weighing_name
        excel_data = pd.read_excel(self.complete_weighing_name)
        self.raw_data = excel_data.poids_brut
        if (self.raw_data == 0).all():
            raise ValueError("Toutes les valeurs de la colonne 'poids_brut' sont zéro.")
        self.phases_data = excel_data.phase
        self.time_bulk_to_dribble = (self.phases_data == VALUE_FILE_BULK_TO_DRIBBLE).sum()
        self.time_dribble_to_close = (self.phases_data == VALUE_FILE_DRIBBLE_TO_CLOSE).sum()
        if (self.toprint):
            print(f"time bulk to dribble : {self.time_bulk_to_dribble}", flush=True)
            print(f"time dribble to close : {self.time_dribble_to_close}", flush=True)
        _filter = ButterworthFilter(1, 4, 627, btype='low')
        filter_signal = _filter.filter_signal(self.raw_data)
        self.start_1, self.end_1 = self.index_values(self.phases_data, 1, 0, len(self.phases_data))
        self.start_3, self.end_3 = self.index_values(self.phases_data, 3, 0, len(self.phases_data))
        self.start_4, self.end_4 = self.index_values(self.phases_data, 4, 0, len(self.phases_data))
        self.start_5, self.end_5 = self.index_values(self.phases_data, 5, 0, len(self.phases_data))
```

```
            self.start_6, self.end_6 = self.index_values(self.phases_data, 6, 0, len(self.phases_data))
            self.start_7, self.end_7 = self.index_values(self.phases_data, 7, 0, len(self.phases_data))
            self.start_8, self.end_8 = self.index_values(self.phases_data, 8, 0, len(self.phases_data))
            self.start_0, self.end_0 = self.index_values(self.phases_data, 0, 0, len(self.phases_data))
            self.grand_debit = filter_signal[self.start_1[0]:self.end_3[0]]
            self.changement = filter_signal[self.start_4[0]:self.end_4[0]]
            self.petit_debit = filter_signal[self.start_5[0]:self.end_6[0]]
            self.fermeture = filter_signal[self.start_7[0]:self.end_7[0]]
            self.stabilisation = filter_signal[self.start_8[0]:self.end_8[0]]
            self.evacuation = filter_signal[self.start_0[0]:self.end_0[0]]

            self.bulk_flow_rates = self.compute_flow(list(self.grand_debit), interval=self.window_size)
            self.flow_change_rates = self.compute_flow(list(self.changement), interval=self.window_size)
            self.dribble_flow_rates = self.compute_flow(list(self.petit_debit), interval=self.window_size)
            self.flow_closure_rates = self.compute_flow(list(self.fermeture), interval=self.window_size)
            self.stabilization_flow_rates = self.compute_flow(list(self.stabilisation), interval=self.window_size)
            return True
        except Exception as e:
            # Log exceptions avec le niveau ERROR
            logging.error(f"Erreur de lecture de la pesée {complete_weighing_name}. Erreur: {e}")
            return False
```

```
    # fonction pour identifier le début et la _end de chaque phase 1 par exemple, dans tout le fichier
    def index_values(self,dataset_phase,phase_number,a,b):
        start_phase,end_phase = [],[]
        index= [i for i in range(a,b) if dataset_phase[i] == phase_number]
        length = len(index)
        i = 0
        while (i< length):
            low = index[i]
            while i <length-1 and index[i]+1 == index[i + 1]:
                i += 1
            high = index[i]
            if (high - low >= 1):
                start_phase.append(low)
                end_phase.append(high)
            else:
                start_phase.append(high)
                end_phase.append(low)
            i += 1
        return start_phase,end_phase
```

```
    def compute_flow(self,signal, interval):
        flow_per_interval = []
        i = 0
        while i < len(signal):
            # Ajuste pour le dernier intervalle s'il y a moins de données restantes
            if len(signal) - i < interval and i != 0:
                interval_start = len(signal) - interval
            else:
                interval_start = i
            end_interval = min(i + interval, len(signal))
            current_interval = signal[interval_start:end_interval]
            # Calcule le débit comme la somme des différences entre chaque point consécutif
            somme_differences = sum(current_interval[j+1] - current_interval[j] for j in range(len(current_interval)-1))
            debit = somme_differences / (len(current_interval)-1) if len(current_interval) > 1 else 0
            flow_per_interval.append(debit)
            i += interval
        return flow_per_interval
```

```
    def values_action_index(self, action):
        if action == 1:
            start = self.start_1
            _end =  self.end_3
        elif action == 2:
            start = self.start_4
            _end =  self.end_4
        elif action == 3:
            start = self.start_5
            _end =  self.end_6
        elif action == 4:
            start = self.start_7
            _end =  self.end_7
        elif action == 5:
            start = self.start_8
            _end =  self.end_8
        else:
            start = self.start_0
            _end =  self.end_0
        return start, _end
```

```
    def flow_value(self, action):
        flow_index = self.flow_meter
        if (self.toprint):
            print(f"Compteur débit : {flow_index}", flush=True)
            print(f" Débits grand débit {self.bulk_flow_rates}", flush=True)
            print(f" Débits chagement {self.flow_change_rates}", flush=True)
            print(f" Débits petit début {self.dribble_flow_rates}", flush=True)
            print(f" Débits fermeture {self.flow_closure_rates}", flush=True)
            self.flow_change_rates
        if action == 1:
            flow_index = self.data_index // self.window_size
            if flow_index >= len(self.bulk_flow_rates):
                debit = self.bulk_flow_rates[-1]
            else:
                debit = self.bulk_flow_rates[flow_index]
        elif action == 2:
            if flow_index >= len(self.flow_change_rates):
                debit = self.flow_change_rates[-1]
            else:
                debit = self.flow_change_rates[flow_index]
        elif action == 3:
            if flow_index >= len(self.dribble_flow_rates):
                debit = self.dribble_flow_rates[-1]
            else:
                debit = self.dribble_flow_rates[flow_index]
        elif action == 4:
            if flow_index >= len(self.flow_closure_rates):
                debit = self.flow_closure_rates[-1]
            else:
                debit = self.flow_closure_rates[flow_index]
        else :
            debit = 0
        self.flow_meter+=1
        return debit
```

```
def read_weight_fixed_duration_phase(self):
    _time = 0
    fixed_duration_weight = []
    duration = self.time_bulk_to_dribble if self.action == ACTION_BULK_TO_DRIBBLE else
        (self.time_dribble_to_close if self.action == ACTION_DRIBBLE_TO_CLOSE else 0)
    while (_time < duration):
        weight = self.read_weight()
        fixed_duration_weight = pd.concat([pd.Series(fixed_duration_weight, dtype=float),pd.Series(weight, dtype=float)]
        _time += self.window_size
    return fixed_duration_weight

def compute_moving_averages(self, _values):
    moving_averages = []
    for i in range(len(_values)):
        window = _values[0:i]
        average = sum(window) / (i+1)
        moving_averages.append(average)
    return moving_averages

def compute_stabilization_phase_duration(self):
    #print(f"Before calculer moyennes mobiles débits stab: {self.stabilization_flow_rates}", flush=True)
    moving_values = self.compute_moving_averages(self.stabilization_flow_rates)
    #print(f"After calculer moyennes mobiles valeurs mobiles: {moving_values} len: {len(moving_values)}", flush=True)

    found_stable = False
    for i in range(4, len(moving_values)):
        #if (-0.0001 < moving_values[i] < 0.0001):
        if (-0.0004 < moving_values[i] < 0.0004):
            duration = i * self.window_size
            found_stable = True
            break
    if not found_stable:
        duration = -1  # if no stabilization is found
    if self.toprint:
        print("Durée de la stabilisation=", duration, flush=True)
    return duration
```

```
def read_weight_stabilization_phase(self):
    _time = 0
    weight_phase_stabilisation = []


    self.time_stabilization = self.compute_stabilization_phase_duration()
    while (_time < self.time_stabilization):
        weight = self.read_weight()
        weight_phase_stabilisation  = pd.concat([pd.Series(weight_phase_stabilisation, dtype=float ),
                                                 pd.Series(weight, dtype=float)])
        _time += self.window_size
    return weight_phase_stabilisation

def read_weight_phase_init(self):
    _time = 0
    weight = pd.Series(dtype='float64')
    for i in range(1,self.window_size+1):
            weight = weight._append(pd.Series([self.current_weight], index=[self.extrapolation_index], dtype=float))
            simulator_data['time_step'].append(self.extrapolation_index)
            simulator_data['phase'].append(self.action)
            simulator_data['raw_weight'].append(self.extrapolated_weight)
            self.extrapolation_index += 1
    simulator_data['extrapolation'].extend(([self.extrapolation for _ in range(0,self.window_size)]))
    return weight
```

```
def read_weight(self):
    weight = []
    _filter = ButterworthFilter(1, 4, 627, btype='low')
    start, _end = self.values_action_index(self.action)
    start = start[0]
    _end = _end[0]
    if (self.toprint):
        print(f"extrapolation : {self.extrapolation}", flush=True)
    if (self.extrapolation == False):
        if self.window_size <= (_end - (start + self.data_index)  + 1):
            weight = self.raw_data[start+self.data_index:start+self.data_index+self.window_size]
            self.data_index += self.window_size
            last_index = weight.index[-1]
            filter_signal = _filter.filter_signal(weight)
            self.last_filtered_weight = filter_signal[-1]
            self.extrapolated_weight = self.last_filtered_weight
            self.extrapolation_index = last_index + 1
            simulator_data['phase'].extend([self.action for _ in range(0,self.window_size)])
            simulator_data['time_step'].extend(weight.index[0:self.window_size])
            simulator_data['raw_weight'].extend(weight.iloc[0:self.window_size])
        else:
            weight = self.raw_data[start+self.data_index:_end+1]
            filter_signal = _filter.filter_signal(self.raw_data[_end-self.window_size:_end+1])
```

```
            # Determine the range
            n = (_end + 1) - (start + self.data_index)
            # Extend phase with repeated actions
            simulator_data['phase'].extend([self.action] * n)
            # Extend raw_weight with values from weight within the range
            simulator_data['raw_weight'].extend(weight.iloc[0:n])
            # Extend time_step with indices from weight within the range
            simulator_data['time_step'].extend(weight.index[0:n])
            self.extrapolation_flow = self.flow_value(self.action)
            self.last_filtered_weight = filter_signal[-1]
            self.extrapolated_weight = self.last_filtered_weight
            for i in range(1, self.window_size - (_end - (start + self.data_index))):
                self.extrapolated_weight = float("{:.8f}".format(self.extrapolated_weight + self.extrapolation_flow))
                last_index = weight.index[-1] if len(weight) > 0 else -1
                new_index = last_index + 1
                weight = weight._append(pd.Series([self.extrapolated_weight], index=[new_index], dtype='float64'))
                simulator_data['phase'].append(self.action)
                simulator_data['raw_weight'].append(self.extrapolated_weight)
                simulator_data['time_step'].append(self.extrapolation_index)
            self.extrapolation_index = new_index + 1
            self.extrapolation = True
```

```
    else:
        # extrapoler
        if (self.toprint):
            print(f"\n J'extrapole action: {self.action}", flush=True)
        self.extrapolation_flow = self.flow_value(self.action)
        if isinstance(weight, list):
            weight = pd.Series(weight, dtype='float64')
        for i in range(1,self.window_size+1):
            self.extrapolated_weight = float("{:.8f}".format(self.extrapolated_weight + self.extrapolation_flow))
            weight = weight._append(pd.Series([self.extrapolated_weight], index=[self.extrapolation_index], dtype=float)
            simulator_data['time_step'].append(self.extrapolation_index)
            simulator_data['phase'].append(self.action)
            simulator_data['raw_weight'].append(self.extrapolated_weight)
            self.extrapolation_index += 1
    simulator_data['extrapolation'].extend(([self.extrapolation for _ in range(0,self.window_size)]))
    return weight
```

```
    def send_to_buffer(self):
        if self.action == ACTION_INIT:
            weight = self.read_weight_phase_init()
            self.stop_weighing()
        else:
            weight = self.read_weight()
        if (self.toprint):
            print(f"\n weight lu : {weight}", flush=True)
            print(f"\n action : {self.action} extrapolation: {self.extrapolation}", flush=True)
            print("\n après weight lu", flush=True)

        weight_serie = pd.Series(weight, dtype=float)
        # Ajouter chaque élément de la série au buffer
        for idx, valeur in weight_serie.items():
            self.current_weight = valeur
            self.buffer.add((idx, valeur, self.action))
        if self.action == ACTION_INIT:
            weight = self.read_weight_phase_init()
            self.stop_weighing()
```

```
    def change_phase(self, action):
        start, _end = self.values_action_index(self.action)
        if self.data_index < _end[0]:
            self.extrapolation = True
        self.action = action
        self.data_index = 0  # Réinitialise l'index pour la nouvelle action
        self.flow_meter = 0
        print(f"\n Changement d'action {action}.", flush=True)
```

```
class ReplayBuffer:
    def __init__(self, capacity):  # Constructeur de la classe qui initialise le tampon de replay
        # La capacité maximale du tampon, soit le nombre maximum d'expériences (états, actions, récompenses, etc.)
        # que le tampon peut contenir.
        # Une fois cette capacité atteinte, les expériences les plus anciennes seront écrasées.
        self.capacity = capacity
        self.buffer = []  # Une liste qui stocke les expériences.
        self.position = 0  # Un indice pour suivre le point d'insertion de manière circulaire.

    def push(self, state, action, reward, next_state, done):
        # Méthode pour ajouter une expérience au tampon
        if len(self.buffer) < self.capacity:
            self.buffer.append(None)  # Ajoute un espace réservé si le tampon n'est pas plein
        # Insère la nouvelle expérience à la position courante
        self.buffer[self.position] = (state, action, reward, next_state, done)
        # Met à jour la position pour l'insertion suivante, en bouclant avec l'opération modulo sur capacity
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        # Méthode pour échantillonner aléatoirement un lot d'expériences du tampon
        batch = random.sample(self.buffer, batch_size)
        # Empile chaque composant (états, actions, etc.) dans des tableaux séparés et les retourne
        state, action, reward, next_state, done = map(np.stack, zip(*batch))
        return state, action, reward, next_state, done

    def __len__(self):
        return len(self.buffer)  # Retourne le nombre actuel d'expériences stockées dans le tampon.
```

- L’agent :

```
import random
import time
import queue
import numpy as np
import tensorflow as tf
import math

class DQNAgent:
    def __init__(self, explore, buffer, window_size, simulator, expected_weight = 25.0, expected_weight_min = 24.9,
                 expected_weight_max = 25.1, weight_begin_close = 24, weight_begin_dribble = 20, k7=0.5, toprint=False):
        self.explore = explore # Indicateur pour activer ou désactiver l'exploration
        self.buffer = buffer # Tampon de replay pour stocker les expériences
        self.window_size = window_size # Taille de la fenêtre pour l'observation
        self.simulator = simulator  # Instance du simulateur de pesée
        self.model = self.create_model()   # Création du modèle principal pour l'agent
        self.target_model = self.create_model() # Création du modèle cible pour l'agent
        self.replay_buffer = ReplayBuffer(MEMORY_SIZE) # Initialisation du tampon de replay avec une taille définie
        self.epsilon = EPSILON # Paramètre d'exploration pour le dilemme exploration-exploitation
        self.toprint = toprint # Indicateur pour contrôler l'affichage des informations
        self.action_actuelle=ACTION_BULK # Action initiale de l'agent
        self.fixed_phase_duration = 0  # Remise à zéro de la durée de la phase
        self.expected_weight = expected_weight  # Poids attendu dans l'environnement
        self.weight_begin_close = weight_begin_close  # Seuil de poids à partir duquel l'agent peut commencer la fermeture
        self.weight_begin_dribble = weight_begin_dribble # Seuil de poids à partir duquel l'agent peut
                                                         # commencer le petit débit
        self.expected_weight_min = expected_weight_min  # Poids minimum accepté
        self.expected_weight_max = expected_weight_max # Poids maximum accepté
        self.k7 = k7  # Paramètre spécifique pour l'agent
        self.total_training_steps = 0  # Compteur du nombre total de pas d'entraînement
        self.total_reward = 0.0 # Récompense mise à zéro
        self.actions_sequence = [] # Séquence des actions prises par l'agent
```

```
    # Créer le réseau de neurones pour la fonction Q
    def create_model(self):
        model = tf.keras.models.Sequential([
            tf.keras.layers.Dense(32, input_dim=STATE_DIM, activation='relu'),
            tf.keras.layers.Dense(32, activation='relu'),
            tf.keras.layers.Dense(ACTION_DIM, activation='linear')
        ])
        model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE))
        return model

    def save(self, filename):
        self.model.save(filename)

    def load(self, filename):
        self.model = tf.keras.models.load_model(filename)
```

```python
# Choisir une action à partir de l'état actuel
def choose_action(self, action_in_state, state, _time_since_beginning): #(self, state, _time_since_beginning):
    if (_time_since_beginning <= BULK_FEED_BLANKING_TIME):
        return ACTION_BULK
    if action_in_state == ACTION_INIT:
        return ACTION_INIT
    duration = (
        self.simulator.time_bulk_to_dribble if action_in_state == ACTION_BULK_TO_DRIBBLE else
        self.simulator.time_dribble_to_close if action_in_state == ACTION_DRIBBLE_TO_CLOSE else
        self.simulator.compute_stabilization_phase_duration() if action_in_state == ACTION_STABILIZATION else
        0
    )
    if action_in_state in {ACTION_BULK_TO_DRIBBLE, ACTION_DRIBBLE_TO_CLOSE, ACTION_STABILIZATION}:
        if self.fixed_phase_duration < duration:
            return action_in_state
        else:
            self.fixed_phase_duration = 0
            if action_in_state == ACTION_BULK_TO_DRIBBLE:
                return ACTION_DRIBBLE
            elif action_in_state == ACTION_DRIBBLE_TO_CLOSE:
                return ACTION_STABILIZATION
            elif action_in_state == ACTION_STABILIZATION:
                return ACTION_INIT
    random_value = np.random.rand()
    print(f"Random value: {random_value}, Epsilon: {self.epsilon}")
    if (self.toprint):
        print(f"action in state : {action_in_state}", flush=True)
    if self.total_training_steps < MIN_EXPLORATION_STEPS or (self.explore and random_value < self.epsilon):
        if (action_in_state == ACTION_BULK):
            if (_time_since_beginning > BULK_FEED_BLANKING_TIME) and (state[0, 0] > self.weight_begin_dribble):
                action = random.randint(ACTION_BULK, ACTION_BULK_TO_DRIBBLE)
            else:
                action = ACTION_BULK
        else:
            if (state[0, 0] > self.weight_begin_close):
                action = random.randint(ACTION_DRIBBLE, ACTION_DRIBBLE_TO_CLOSE)
            else:
                action = ACTION_DRIBBLE
        if (self.toprint):
            print("random action = ",action, flush=True)

    else:
        q_values = self.model.predict(state, verbose=0)
        print("qvalues = ", q_values)
        action = int(np.argmax(q_values[0]))
        print("qvalue action = ", action)

        # Should follow valid sequence
        if action_in_state == ACTION_STABILIZATION and action != ACTION_STABILIZATION:
            action = ACTION_INIT
        elif action != ACTION_INIT and action_in_state != ACTION_INIT and action != action_in_state:
            action = action_in_state + 1

    if (self.toprint):
        print("current action in state = ", action_in_state)
        print("qvalue or random action = ", action)

    return action
```

```python
#Calcule la récompense pour la précision avec une plage de tolérance.
def reward_precision_with_tolerance(self, current_weight, k7 = 0.5):
    if not (self.expected_weight_min <= current_weight <= self.expected_weight_max):
        return -1 # Pénalité si le poids final est hors de la plage

    # Ajouter un bonus ou ajuster la récompense basée sur la précision finale
    final_precision_bonus = 1.0 - self.k7 * abs(current_weight - self.expected_weight)
    return final_precision_bonus

#Calcule la récompense pour l'efficacité temporelle.
def reward_time(self, current_weight, time_step):
    # Récompense basée sur le poids et le temps à chaque pas
    epsilon = 0.1
    time_adjusted_reward = current_weight / (time_step + epsilon)
    return time_adjusted_reward
```

```python
#Vérifie si la séquence donnée est une sous-séquence valide de full_sequence et gère une terminaison spéciale.
def is_sequence_valid(self, sequence, full_sequence, special_end=0):
    """
    :param sequence: La séquence à vérifier.
    :param full_sequence: La séquence complète attendue.
    :param special_end: La terminaison spéciale qui peut être valide si la séquence complète est suivie.
    :return: True si la séquence est valide, False sinon.
    """
    if not sequence:
        return False

    # Gérer la terminaison spéciale
    if sequence[-1] == special_end:
        if sequence[:-1] == full_sequence:
            return True
        else:
            return False

    # Vérifier si la séquence est une sous-séquence valide de full_sequence
    full_length = len(full_sequence)
    for i in range(min(len(sequence), full_length)):
        if sequence[i] != full_sequence[i]:
            return False
    return True
```

```python
# Ajoute une action à la séquence si elle n'est pas un doublon de la dernière action dans la séquence.
def add_action_to_sequence(self, action):
    """
    :param action: La nouvelle action à ajouter.
    """
    print(f"before add action to sequence: {self.actions_sequence}", flush=True)
    if not self.actions_sequence or (self.actions_sequence and action != self.actions_sequence[-1]):
        self.actions_sequence.append(action)
    print(f"after add action to sequence: {self.actions_sequence}", flush=True)

# Calcule la récompense pour la séquence actuelle
def compute_sequence_reward(self, actions_sequence, full_sequence=[ACTION_BULK, ACTION_BULK_TO_DRIBBLE, ACTION_DRIBBLE,
                                                                    ACTION_DRIBBLE_TO_CLOSE, ACTION_STABILIZATION], special_end=ACTION_INIT):

    if self.is_sequence_valid(actions_sequence, full_sequence, special_end):
        return 1  # Récompense pour une séquence valide
    return -1  # Aucune récompense si la séquence est invalide

# Combinaison des récompenses pour calculer la récompense totale
def compute_reward(self, current_weight, time_step, k7=0.5, is_final_step=False):
    if is_final_step:
        r_precision = self.reward_precision_with_tolerance(current_weight, k7)
        r_time = self.reward_time(current_weight, time_step)
        r_sequence = self.compute_sequence_reward(self.actions_sequence)

        # Calcul de la récompense totale comme produit des composantes
        if r_sequence == -1 or r_precision == -1:
            total_r = -1
        else:
            total_r =  r_precision + r_time
    else:
        total_r = 0
    return total_r
```

```
def process_window(self, window_data, action, _time_since_beginning):
    if (self.toprint):
        print(f"type {type(window_data)}", flush=True)
    _filter = ButterworthFilter(1, 4, 627, btype='low') # Crée un filtre Butterworth
    filtered_signal = _filter.filter_signal(window_data) # Applique le filre au signal de la fenêtre
    last_weight = filtered_signal[-1] # Obtient le dernier poids du signal filtré
    signal = filtered_signal
    if (self.toprint):
        print("\n agent signal filtré", flush=True)
        print(filtered_signal, flush=True)
    # Crée l'état à partir du dernier poids du signal filtré
    state = np.array([last_weight])
    state = np.reshape(state, [1, STATE_DIM])
    # Vérifie si l'action est l'initialisation pour déterminer si c'est la dernière étape
    is_final_step = action == ACTION_INIT
    # Calcule la récompense en fonction du dernier poids et du temps écoulé
    _reward = self.compute_reward(last_weight, _time_since_beginning, self.k7, is_final_step)
    if (self.toprint):
        print(f"recompenses: {_reward}", flush=True)
    DQNAgent_data['filtered_weight'].extend(filtered_signal[0:self.window_size])
    if (self.toprint):
        print("after 1.1", flush=True)
    DQNAgent_data['phase'].extend([action for _ in range(0,self.window_size)])
    if (self.toprint):
        print("after 1.2", flush=True)
    DQNAgent_data['time_step'].extend([i for i in range(_time_since_beginning, _time_since_beginning + len(window_data))
    if (self.toprint):
        print("after 1.3", flush=True)
    DQNAgent_data['raw_weight'].extend(window_data)
    if (self.toprint):
        print("after 1.4", flush=True)
    DQNAgent_data['reward'].extend([_reward for _ in range(0,len(window_data))])
    if (self.toprint):
        print("after 1.5", flush=True)
    return last_weight, state, _reward
```

```
def process_weighing(self):
    state = self.reset()
    if (self.toprint):
            print(f"state {state} dim {state.shape}", flush=True)
    last_filtered_weight = 0.0
    #previous_weight = 0.0
    time_since_beginning = 0
    last_action = ACTION_BULK
    while self.simulator.active_weighing or not self.buffer.is_empty():

        if last_action == ACTION_BULK:
            self.weight_bulk = last_filtered_weight

        action = self.choose_action(last_action, state, time_since_beginning)
        self.add_action_to_sequence(action)
        if last_action != action:
            print(f"last action: {last_action} action: {action}", flush=True)
            self.simulator.change_phase(action)
        last_action = action
        window_dict = {}  # Utiliser un dictionnaire pour collecter les données
        # Remplir la fenêtre jusqu'à la taille désirée ou jusqu'à ce que le buffer soit vide
        _action = action
        while len(window_dict) < self.window_size:
            item = self.buffer.remove()
            if item is not None:
                index, value, _action = item
                window_dict[index] = value  # Ajouter directement les données au dictionnaire
            else:
                if not self.simulator.active_weighing and self.buffer.is_empty():
                    break
```

```
                if (self.toprint):
                    print("Attente de plus de données...", flush=True)
                    print(f"dans attente ....pesée active : {self.simulator.active_weighing } buffer vide :
                          {self.buffer.is_empty()}", flush=True)
                time.sleep(0.1)
            # Créer la série pandas directement à partir du dictionnaire
            if window_dict:
                window_serie = pd.Series(window_dict, name='raw_weight', dtype=float)
                if (self.toprint):
                    print(f"Agent traite la fenêtre : {window_serie}", flush=True)
                    print(f"\n taille fenêtre : {len(window_serie)}", flush=True)
                last_filtered_weight, next_state, _reward = self.process_window(window_serie, _action, time_since_beginning)
                self.total_reward = -1 if _reward == -1 else self.total_reward + _reward
                if (self.toprint):
                    print(f"Après traiter fenêtre dernier weight: {last_filtered_weight} nextstate: {next_state}
                                reward: {_reward}", flush=True)
                time_since_beginning += self.window_size
                if action == ACTION_BULK_TO_DRIBBLE or action == ACTION_DRIBBLE_TO_CLOSE or action == ACTION_STABILIZATION:
                    self.fixed_phase_duration += self.window_size
                done = not (self.simulator.active_weighing or not self.buffer.is_empty())
                if (self.toprint):
                    print(f"pesée active : {self.simulator.active_weighing } buffer vide : {self.buffer.is_empty()}
                                        done : {done}", flush=True)
                    print(f"before remember state: {state} action: {action} reward: {_reward}
                          total reward: {self.total_reward} nextstate: {next_state} done: {done}", flush=True)
                print(f"action de l'état: {action}", flush=True)
                self.remember(state, action, self.total_reward, next_state, done)
                state = next_state
                self.learn()
                if done:
                    print("Dernière récompense =", _reward, flush=True)
                    print("État =", state, flush=True)
                    break
```

```
            else:
                if (self.toprint):
                    print(f"Else pesée active : {self.simulator.active_weighing } buffer vide : {self.buffer.is_empty()}",
                          flush=True)

        DQNAgent_data['filtered_weight'].append(last_filtered_weight)
        DQNAgent_data['phase'].append(ACTION_INIT)
        DQNAgent_data['time_step'].append(-2)
        DQNAgent_data['raw_weight'].append(self.actions_sequence)
        DQNAgent_data['reward'].append(self.total_reward)

    # Enregistrer l'expérience dans la mémoire de replay
    def remember(self, state, action, reward, next_state, done):
        self.replay_buffer.push(state, action, reward, next_state, done)
```

```python
def learn(self):
    # Réduire la probabilité d'exploration au fil du temps
    if self.epsilon > EPSILON_MIN:
        self.epsilon *= EPSILON_DECAY
    self.total_training_steps += 1

    if len(self.replay_buffer) < BATCH_SIZE:
        return
    # Prélever un échantillon aléatoire de la mémoire de replay
    state_batch, action_batch, reward_batch, next_state_batch, done_batch = self.replay_buffer.sample(BATCH_SIZE)
    state_batch= state_batch.reshape((BATCH_SIZE,STATE_DIM))
    # Calculer les valeurs Q pour l'état actuel
    q_values = self.model.predict(state_batch, verbose=0)

    next_state_batch= next_state_batch.reshape((BATCH_SIZE,STATE_DIM))
    # Calculer les valeurs Q pour l'état suivant avec le réseau de neurones cible
    next_q_values = self.target_model.predict(next_state_batch,verbose=0)
    # Trouver la valeur Q maximale à chaque état suivant
    max_next_q = np.max(next_q_values, axis=1)

    # Calculer les cibles d'apprentissage pour les valeurs Q
    y_batch = reward_batch + (1 - done_batch) * GAMMA * max_next_q

    if (self.toprint):
        print(f"batch size: {BATCH_SIZE} action batch: {action_batch} qvalues shape: {q_values.shape}", flush=True)

    action_batch = action_batch.astype(int) # Convertir action_batch en entiers
    q_values[np.arange(BATCH_SIZE), action_batch] = y_batch # Mettre à jour les valeurs Q pour les actions choisies

    # Entraîner le réseau de neurones avec les états et les valeurs Q calculés
    self.model.train_on_batch(state_batch, q_values)

    # Mettre à jour le réseau de neurones cible avec les poids du réseau de neurones
    if self.total_training_steps % TARGET_UPDATE_FREQ == 0:
        self.target_model.set_weights(self.model.get_weights())
```

```python
# Entraîner l'agent dans l'environnement
def train(self):
    WeighingSimulator # Référence à la classe du simulateur de pesée
    current_directory = os.getcwd()
    folder_path = os.path.join(current_directory, 'train')  # Le dossier d'entrainement
    prefix = 'cycle'  # Le préfixe des fichiers de pesées
    dqn_model_file_name = os.path.join(current_directory, 'dqn_weighing_agent_model.h5') # Nom du fichier du modèle DQN
    # Charger le modèle s'il existe
    try:
        self.load(dqn_model_file_name) # Charger le modèle existant
        print("Weighing agent model loaded successfully.")
    except Exception as e:
        print("No existing model found. Training a new model for Weighing agent.")
    # Créer le générateur de noms de fichiers de pesées
    max_episode_number = MAX_EPISODES # Nombre maximum d'épisodes pour l'entraînement
    generateur_pesee = generate_complete_filename(folder_path, prefix, max_episode_number)
    num_episodes = 0
    for complete_weighing_name in generateur_pesee:
        reset_Dictionaries() # Réinitialise les dictionnaires de données pour chaque épisode
        print(f"Nom de pesée unique sélectionné (épisode): {complete_weighing_name}", flush=True)

        weighing_started = self.simulator.start_weighing(complete_weighing_name)  # Démarre la pesée avec le simulateur
        if not weighing_started:
            continue
        print(f"Traitement de la pesée {complete_weighing_name}...", flush=True)
        self.process_weighing() # Traite la pesée
        num_episodes += 1
        df_DQNAgent = pd.DataFrame(DQNAgent_data) # Crée un DataFrame à partir des données de l'agent DQN
        # Extrait le numéro de l'épisode du nom de fichier à l'aide d'une expression régulière
        match = re.search(r'cycle_(\d+)', complete_weighing_name)
```

```
            # Répertoire des enregistrements d'entraînement
            train_records_directory = os.path.join(current_directory, 'train_records')
            if not os.path.exists(train_records_directory):
                os.makedirs(train_records_directory)  # Crée le répertoire s'il n'existe pas
            # Enregistre les données de l'agent DQN dans un fichier Excel nommé en fonction de l'épisode
            agent_records_file_name = os.path.join(train_records_directory, f'DQNAgent_episode_{match.group(1)}.xlsx')
            df_DQNAgent.to_excel(agent_records_file_name, index=False)
            # Sauvegarde le modèle tous les 20 épisodes
            if (num_episodes) % 20 == 0:
                self.save(dqn_model_file_name)
                print(f"Model saved at episode {num_episodes}.")
        self.simulator.stop_simulator() # Arrête le simulateur après l'entraînement
        self.save(dqn_model_file_name)  # Sauvegarde le modèle à la fin de l'entraînement
        print(f"Model saved at the end of training.")
        print("L'entrainement est terminée.", flush=True) # Indique que l'entraînement est terminé
```

```
    # Tester l'agent dans l'environnement
    def test(self):
        folder_path = os.path.join(os.getcwd(), 'test')  # Le dossier pour les tests
        prefix = 'cycle'  # Le préfixe des fichiers de pesées

        # Créer le générateur de noms de fichiers de pesées
        max_nombre_episode = MAX_EPISODES
        generateur_pesee = generate_complete_filename(folder_path, prefix, max_nombre_episode)
        for complete_weighing_name in generateur_pesee:
            reset_Dictionaries() # Réinitialise les dictionnaires de données
            print(f"Nom de pesée unique sélectionné (épisode): {complete_weighing_name}", flush=True)

            # Démarre la pesée avec le simulateur
            weighing_started = self.simulator.start_weighing(complete_weighing_name)
            if not weighing_started:
                continue
            print(f"Test de la pesée {complete_weighing_name}...", flush=True)
            self.process_weighing()  # Traite la pesée

            df_DQNAgent = pd.DataFrame(DQNAgent_data)  # Crée un DataFrame à partir des données de l'agent DQN

            match = re.search(r'cycle_(\d+)', complete_weighing_name) # Extrait le numéro de l'épisode du nom de fichier

            # Répertoire des enregistrements de test
            current_directory = os.getcwd()
            test_records_directory = os.path.join(current_directory, 'test_records')
            if not os.path.exists(test_records_directory):
                os.makedirs(test_records_directory)
            # Enregistre les données de l'agent DQN dans un fichier Excel
            agent_records_file_name = os.path.join(test_records_directory, f'DQNAgent_episode_{match.group(1)}.xlsx')
            df_DQNAgent.to_excel(agent_records_file_name, index=False)

        self.simulator.stop_simulator() # Arrête le simulateur
        print("Le test est terminée.", flush=True)
```

```
    def reset(self):
        self.weight_bulk = 0.0 # Réinitialise le poids en vrac
        self.current_weight = 0.0 # Réinitialise le poids actuel
        self.total_reward = 0.0  # Réinitialise la récompense totale
        self.stabilization = 0.0  # Réinitialise la stabilisation
        self.fixed_phase_duration = 0  # Réinitialise la durée de la phase fixe

        state = np.array([self.current_weight]) # État initial basé sur le poids actuel
        state = np.reshape(state, [1, STATE_DIM])  # Redimensionne l'état pour l'entrée du modèle
        self.actions_sequence = []  # Réinitialise la séquence des actions
        return state  # Retourne l'état initial
```

```python
# Entraîner l'agent dans l'environnement
from threading import Thread  # Importation pour gérer les threads
import tensorflow as tf  # Importation de TensorFlow pour la création et l'entraînement du modèle

if __name__ == "__main__":
    window_size = 25  # Taille de la fenêtre

    # Création du buffer et des acteurs
    buffer = Buffer()  # Initialisation du tampon pour stocker les expériences
    toprint = False  # Indicateur pour contrôler l'affichage des informations
    simulator = WeighingSimulator(buffer, window_size, toprint)  # Création du simulateur avec les paramètres donnés
    expected_weight = 25.0  # Poids attendu dans l'environnement
    weight_begin_close = 24  # Seuil de poids à partir duquel l'agent peut envisager la fermeture
    weight_begin_dribble = 20  # Seuil de poids à partir duquel l'agent peut envisager le petit débit
    expected_weight_min = 24.5  # Poids minimum attendu
    expected_weight_max = 25.5  # Poids maximum attendu

    explore = True  # Indicateur pour activer l'exploration

    k7 = 0.5  # Paramètre spécifique pour l'agent
    # Création de l'agent DQN avec les paramètres donnés
    agent = DQNAgent(explore, buffer, window_size, simulator, expected_weight, expected_weight_min, expected_weight_max,
                     weight_begin_close, weight_begin_dribble, k7, toprint)

    # Lancer la simulation
    simulator.start()  # Démarrer le simulateur dans un thread séparé
    agent.train()  # Entraîner l'agent avec les expériences du tampon
```

## RÉFÉRENCES BIBLIOGRAPHIQUES


"Bridgestone introduces new state-of-the-art tire assembling system news | bridgestone." Bridgestone Global Website.

(2020). SpeedAc iQ Net Weighing Baggin System. Operating and Maintenance Manual, Premier Tech L.

Akundi, A., D. Euresti, S. Luna, W. Ankobiah, A. Lopes and I. Edinbarough (2022). "State of Industry 5.0 : Analysis and Identification of Current Research Trends." Applied System Innovation **5**(1): 27.

AlHinai, N. (2020). Chapter 1 - Introduction to biomedical signal processing and artificial intelligence. Biomedical Signal Processing and Artificial Intelligence in Healthcare. W. Zgallai, Academic Press**:** 1-28.

Ambardar, A. (1995). Analog and digital signal processing, PWS Boston, MA, USA.

Arsene, C. T., R. Hankins and H. Yin (2019). Deep learning models for denoising ECG signals. 2019 27th European signal processing conference (EUSIPCO), IEEE.

Awad, M. and R. Khanna (2015). Support Vector Regression. Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. M. Awad and R. Khanna. Berkeley, CA, Apress**:** 67-80.

Bahar, H. B. and D. H. Horrocks (1998). "Dynamic weight estimation using an artificial neural network." Artificial Intelligence in Engineering **12**(1): 135-139.

Bank, D., N. Koenigstein and R. Giryes (2023). "Autoencoders." Machine learning for data science handbook: data mining and knowledge discovery handbook: 353-374.

Bellman, R. (1966). "Dynamic programming." Science **153**(3731): 34-37.

Bertsekas, D. (2012). Dynamic programming and optimal control: Volume I, Athena scientific.

Bhattacharya, B., A. H. Lobbrecht and D. P. Solomatine (2003). "Neural Networks and Reinforcement Learning in Control of Water Systems." Journal of Water Resources Planning and Management **129**(6): 458-465.

Booch, G., J. Rumbaugh and I. Jacobson (2005). Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series), Addison-Wesley Professional.

Boschetti, G., R. Caracciolo, D. Richiedei and A. Trevisani (2013). "Model-based dynamic compensation of load cell response in weighing machines affected by environmental vibrations." Mechanical Systems and Signal Processing **34**(1-2): 116-130.

Cai, W., J. Wang, P. Jiang, L. Cao, G. Mi and Q. Zhou (2020). "Application of sensing techniques and artificial intelligence-based methods to laser welding real-time monitoring: A critical review of recent literature." Journal of Manufacturing Systems **57**: 1-18.

Carpenter, G. A. (1989). "Neural network models for pattern recognition and associative memory." Neural Networks **2**(4): 243-257.
Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.
Chiang, H. T., Y. Y. Hsieh, S. W. Fu, K. H. Hung, Y. Tsao and S. Y. Chien (2019). "Noise Reduction in ECG Signals Using Fully Convolutional Denoising Autoencoders." IEEE Access **7**: 60806-60813.
Choi, K. N. (2017). "Noise in load cell signal in an automatic weighing system based on a belt conveyor." Journal of Sensors **2017**.
Clijsters, S., T. Craeghs, S. Buls, K. Kempen and J. P. Kruth (2014). "In situ quality control of the selective laser melting process using a high-speed, real-time melt pool monitoring system." The International Journal of Advanced Manufacturing Technology **75**(5): 1089-1101.
Craeghs, T., F. Bechmann, S. Berumen and J.-P. Kruth (2010). "Feedback control of Layerwise Laser Melting using optical sensors." Physics Procedia **5**: 505-514.
Deisenroth, M. and C. E. Rasmussen (2011). PILCO: A model-based and data-efficient approach to policy search. Proceedings of the 28th International Conference on machine learning (ICML-11).
Del Moral, P. (1997). "Nonlinear filtering: Interacting particle resolution." Comptes Rendus de l'Académie des Sciences-Series I-Mathematics **325**(6): 653-658.
Di Lorenzo, R., G. Ingarao and F. Micari (2006). "On the use of artificial intelligence tools for fracture forecast in cold forming operations." Journal of Materials Processing Technology **177**(1): 315-318.
Diniz, P. S. (1997). Adaptive filtering, Springer.
Fang, X., G. Gong, G. Li, L. Chun, P. Peng, W. Li, X. Shi and X. Chen (2022). "Deep reinforcement learning optimal control strategy for temperature setpoint real-time reset in multi-zone building HVAC system." Applied Thermal Engineering **212**: 118552.
Forslund, R., A. Snis and S. Larsson (2021). "A greedy algorithm for optimal heating in powder-bed-based additive manufacturing." Journal of Mathematics in Industry **11**(1): 14.
Ganguli, R., I. Chopra and D. J. Haas (1998). "Helicopter rotor system fault detection using physics-based model and neural networks." AIAA journal **36**(6): 1078-1086.
Garsten, E. "Amazon chooses uveye drive-through tech to prevent van breakdowns." Forbes.
Goldberg, D. E. and J. H. Holland (1988). "Genetic Algorithms and Machine Learning." Machine Learning **3**(2): 95-99.
Goodfellow, I., Y. Bengio and A. Courville (2016). Deep learning, MIT press.
Gope, D., P. C. Gope, A. Thakur and A. Yadav (2015). "Application of artificial neural network for predicting crack growth direction in multiple cracks geometry." Applied Soft Computing **30**: 514-528.
Hajgató, G., G. Paál and B. Gyires-Tóth (2020). "Deep reinforcement learning for real-time optimization of pumps in water distribution systems." Journal of Water Resources Planning and Management **146**(11): 04020079.
He, Z., Q. Li, M. Chu and G. Liu (2023). "Dynamic weighing algorithm for dairy cows based on time domain features and error compensation." Computers and Electronics in Agriculture **212**: 108077.

He, Z., K.-P. Tran, S. Thomassey, X. Zeng, J. Xu and C. Yi (2021). "A deep reinforcement learning based multi-criteria decision support system for optimizing textile chemical process." Computers in Industry **125**: 103373.
Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." Neural computation **9**(8): 1735-1780.
Huang, J., M. Algahtani and S. Kaewunruen (2022). "Energy forecasting in a public building: a benchmarking analysis on long short-term memory (LSTM), support vector regression (SVR), and extreme gradient boosting (XGBoost) networks." Applied Sciences **12**(19): 9788.
Hurst, J., A. Abate, M. Bernards, A. Nguyen and H. M. Nguyen (2020). Bipedal robot, Google Patents.
Hyndman, R. J. and G. Athanasopoulos (2018). Forecasting: principles and practice, OTexts.
Jhang, Y.-S., S.-T. Wang, M.-H. Sheu, S.-H. Wang and S.-C. Lai (2022). "Channel-Wise Average Pooling and 1D Pixel-Shuffle Denoising Autoencoder for Electrode Motion Artifact Removal in ECG." Applied Sciences **12**(14): 6957.
Jinhuan, W., A. L. Weay and S. Palaniappan "Research on Application of Deep Learning Algorithm in Earthquake Noise Reduction." Academic Journal of Engineering and Technology Science **5**(10): 44-48.
Julier, S. J. and J. K. Uhlmann (1997). New extension of the Kalman filter to nonlinear systems. Signal processing, sensor fusion, and target recognition VI, Spie.
Justusson, B. I. (1981). Median Filtering: Statistical Properties. Two-Dimensional Digital Signal Prcessing II: Transforms and Median Filters. Berlin, Heidelberg, Springer Berlin Heidelberg**:** 161-196.
Kaelbling, L. P., M. L. Littman and A. W. Moore (1996). "Reinforcement learning: A survey." Journal of artificial intelligence research **4**: 237-285.
Kagermann, H., W.-D. Lukas and W. Wahlster (2011). "Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution." VDI nachrichten **13**(1): 2-3.
Kalman, R. E. (1960). "A new approach to linear filtering and prediction problems."
Keshavarzi, S. (2022). "Comparison of Two Algorithms for ECG Signal Denoising: A Recurrent Neural Network and A Support Vector Regression." International Journal of Simulation--Systems, Science & Technology **23**(1).
Kim, J., J. Ko, H. Choi and H. Kim (2021). "Printed Circuit Board Defect Detection Using Deep Learning via A Skip-Connected Convolutional Autoencoder." Sensors **21**(15): 4968.
Kim, S. W., J. H. Kong, S. W. Lee and S. Lee (2022). "Recent advances of artificial intelligence in manufacturing industrial sectors: A review." International Journal of Precision Engineering and Manufacturing: 1-19.
Kohne, T., H. Ranzau, N. Panten and M. Weigold (2020). "Comparative study of algorithms for optimized control of industrial energy supply systems." Energy Informatics **3**(1): 12.
Kondili, E., C. C. Pantelides and R. W. Sargent (1993). "A general algorithm for short-term scheduling of batch operations—I. MILP formulation." Computers & Chemical Engineering **17**(2): 211-227.
Kumar, S. V. (2017). "Traffic flow prediction using Kalman filtering technique." Procedia Engineering **187**: 582-587.

Lehtomaki, N., N. Sandell and M. Athans (1981). "Robustness results in linear-quadratic Gaussian based multivariable control designs." IEEE Transactions on Automatic Control **26**(1): 75-93.
Li, J., R. Jin and H. Z. Yu (2018). "Integration of physically-based and data-driven approaches for thermal field prediction in additive manufacturing." Materials & Design **139**: 473-485.
Liang, F. and O. Qingli (2021). "Design of weighing system based on improved moving average filter algorithm." Journal of Artificial Intelligence Practice **4**(2): 74-77.
Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra (2015). "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971.
Lindner, T., A. Milecki and D. Wyrwał (2021). "Positioning of the Robotic Arm Using Different Reinforcement Learning Algorithms." International Journal of Control, Automation and Systems **19**(4): 1661-1676.
Liu, R., S. Liu and X. Zhang (2021). "A physics-informed machine learning model for porosity analysis in laser powder bed fusion additive manufacturing." The International Journal of Advanced Manufacturing Technology **113**(7-8): 1943-1958.
Liu, S. and G. P. Henze (2006). "Evaluation of Reinforcement Learning for Optimal Control of Building Active and Passive Thermal Storage Inventory." Journal of Solar Energy Engineering **129**(2): 215-225.
Louka, P., G. Galanis, N. Siebert, G. Kariniotakis, P. Katsafados, I. Pytharoulis and G. Kallos (2008). "Improvements in wind speed forecasts for wind power prediction purposes using Kalman filtering." Journal of Wind Engineering and Industrial Aerodynamics **96**(12): 2348-2362.
Ma, W., Q. Li, J. Li, L. Ding and Q. Yu (2021). "A method for weighing broiler chickens using improved amplitude-limiting filtering algorithm and BP neural networks." Information Processing in Agriculture **8**(2): 299-309.
Manju, B. and M. Sneha (2020). "ECG denoising using wiener filter and kalman filter." Procedia Computer Science **171**: 273-281.
Manring, L. H. and B. P. Mann (2022). "Modeling and Reinforcement Learning Control of an Autonomous Vehicle to Get Unstuck From a Ditch." Journal of Autonomous Vehicles and Systems **2**(1).
Markovsky, I. (2015). "Comparison of adaptive and model-free methods for dynamic measurement." IEEE Signal processing letters **22**(8): 1094-1097.
Martin, A. and T. C. Molteno (2015). Automated weighing by sequential inference in dynamic environments. 2015 6th International Conference on Automation, Robotics and Applications (ICARA), IEEE.
McCarthy, J., M. L. Minsky, N. Rochester and C. E. Shannon (2006). "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955." AI magazine **27**(4): 12-12.
Mehrpouya, M., A. Dehghanghadikolaei, B. Fotovvati, A. Vosooghnia, S. S. Emamian and A. Gisario (2019). "The Potential of Additive Manufacturing in the Smart Factory Industrial 4.0: A Review." Applied Sciences **9**(18): 3865.

Meller, M., M. Niedźwiecki and P. Pietrzak (2014). "Adaptive filtering approach to dynamic weighing: a checkweigher case study." IFAC Proceedings Volumes **47**(3): 5927-5932.
Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski (2015). "Human-level control through deep reinforcement learning." nature **518**(7540): 529-533.
Mohanty, S., C. C. Tutum and J. H. Hattel (2013). Cellular scanning strategy for selective laser melting: evolution of optimal grid-based scanning path and parametric approach to thermal homogeneity. Laser-based Micro-and Nanopackaging and Assembly VII, SPIE.
Muller, I., R. M. d. Brito, C. E. Pereira and V. Brusamarello (2010). "Load cells in force sensing analysis -- theory and a novel application." IEEE Instrumentation & Measurement Magazine **13**(1): 15-19.
Niedźwiecki, M., M. Meller and P. Pietrzak (2016). "System identification based approach to dynamic weighing revisited." Mechanical Systems and Signal Processing **80**: 582-599.
Niedźwiecki, M. and P. Pietrzak (2016). "High-precision FIR-model-based dynamic weighing system." IEEE Transactions on Instrumentation and Measurement **65**(10): 2349-2359.
Niedźwiecki, M. and A. Wasilewski (1996). "Application of adaptive filtering to dynamic weighing of vehicles." Control Engineering Practice **4**(5): 635-644.
Ogoke, F. and A. B. Farimani (2021). "Thermal control of laser powder bed fusion using deep reinforcement learning." Additive Manufacturing **46**: 102033.
Paraskevoudis, K., P. Karayannis and E. P. Koumoulos (2020). "Real-Time 3D Printing Remote Defect Detection (Stringing) with Computer Vision and Artificial Intelligence." Processes **8**(11): 1464.
Phate, V. R., R. Malmathanraj and P. Palanisamy (2021). "Classification and indirect weighing of sweet lime fruit through machine learning and meta-heuristic approach." International Journal of Fruit Science **21**(1): 528-545.
Pietrzak, P., M. Meller and M. Niedźwiecki (2014). "Dynamic mass measurement in checkweighers using a discrete time-variant low-pass filter." Mechanical Systems and Signal Processing **48**(1): 67-76.
Pimenov, D. Y., A. Bustillo and T. Mikolajczyk (2018). "Artificial intelligence for automatic prediction of required surface roughness by monitoring wear on face mill teeth." Journal of Intelligent Manufacturing **29**(5): 1045-1061.
Postalcioglu, S., K. Erkan and E. D. Bolat (2005). Comparison of Kalman filter and wavelet filter for denoising. 2005 International Conference on Neural Networks and Brain, IEEE.
Postma, S., R. G. K. M. Aarts, J. Meijer and J. B. Jonker (2002). "Penetration control in laser welding of sheet metal." Journal of Laser Applications **14**(4): 210-214.
Poungponsri, S. and X.-H. Yu (2013). "An adaptive filtering approach for electrocardiogram (ECG) signal noise reduction using neural networks." Neurocomputing **117**: 206-213.
Prieto, M. D., J. Roura and J. R. Martínez (2011). "Bearings fault detection using inference tools." Vibration Analysis and Control-New Trends and Developments: 263-280.
Privitera, D., S. Bellissima and S. Bartolini (2023). "Adaptive Dosing Control System Through ARIMA Model for Peristaltic Pumps." IEEE Access.

Qi, X., Y. Luo, G. Wu, K. Boriboonsomsin and M. Barth (2019). "Deep reinforcement learning enabled self-learning control for energy efficient driving." Transportation Research Part C: Emerging Technologies **99**: 67-81.
Rabiner, L. R. and B. Gold (1975). "Theory and application of digital signal processing." Englewood Cliffs: Prentice-Hall.
Renken, V., L. Lübbert, H. Blom, A. von Freyberg and A. Fischer (2018). "Model assisted closed-loop control strategy for selective laser melting." Procedia CIRP **74**: 659-663.
Ribeiro, J., R. Lima, T. Eckhardt and S. Paiva (2021). "Robotic Process Automation and Artificial Intelligence in Industry 4.0 – A Literature review." Procedia Computer Science **181**: 51-58.
Rich, S. H. and G. J. Prokopakis (1986). "Scheduling and sequencing of batch operations in a multipurpose plant." Industrial & Engineering Chemistry Process Design and Development **25**(4): 979-988.
Richalet, J., A. Rault, J. Testud and J. Papon (1978). "Model predictive heuristic control." Automatica (journal of IFAC) **14**(5): 413-428.
Roques, P. (2009). SysML par l'exemple : un langage de modélisation pour systèmes complexes. Paris, Eyrolles.
Rutan, S. C. (1991). "Adaptive kalman filtering." Analytical Chemistry **63**(22): 1103A-1109A.
Sayadi, O., R. Sameni and M. B. Shamsollahi (2007). ECG denoising using parameters of ECG dynamical model as the states of an extended Kalman filter. 2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE.
Schafer, R. W. (2011). "What Is a Savitzky-Golay Filter?".
Schaul, T., J. Quan, I. Antonoglou and D. Silver (2015). "Prioritized experience replay." arXiv preprint arXiv:1511.05952.
Schulman, J., F. Wolski, P. Dhariwal, A. Radford and O. Klimov (2017). "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347.
Schwalbach, E. J., S. P. Donegan, M. G. Chapman, K. J. Chaput and M. A. Groeber (2019). "A discrete source model of powder bed fusion additive manufacturing thermal history." Additive Manufacturing **25**: 485-498.
Schwendemann, S., Z. Amjad and A. Sikora (2021). "A survey of machine-learning techniques for condition monitoring and predictive maintenance of bearings in grinding machines." Computers in Industry **125**: 103380.
Seidel, C. (2020). "From press shop to validation: BMW Group Plant Munich builds on artificial intelligence and smart use of data."
Selesnick, I. W. and C. S. Burrus (1998). "Generalized digital Butterworth filter design." IEEE Transactions on signal processing **46**(6): 1688-1694.
Shankar, S., T. Mohanraj and R. Rajasekar (2019). "Prediction of cutting tool wear during milling process using artificial intelligence techniques." International Journal of Computer Integrated Manufacturing **32**(2): 174-182.
Siami-Namini, S., N. Tavakoli and A. S. Namin (2018). A comparison of ARIMA and LSTM in forecasting time series. 2018 17th IEEE international conference on machine learning and applications (ICMLA), IEEE.

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller (2014). Deterministic policy gradient algorithms. International conference on machine learning, Pmlr.
Smith, S. (2003). Digital signal processing: a practical guide for engineers and scientists, Newnes.
Smola, A. J. and B. Schölkopf (2004). "A tutorial on support vector regression." Statistics and Computing **14**(3): 199-222.
Song, D., A. M. C. Baek and N. Kim (2021). "Forecasting stock market indices using padding-based fourier transform denoising and time series deep learning models." IEEE Access **9**: 83786-83796.
Sutton, R. S. (1988). "Learning to predict by the methods of temporal differences." Machine learning **3**: 9-44.
Sutton, R. S. and A. G. Barto (2018). Reinforcement learning: An introduction, MIT press.
Sutton, R. S., D. McAllester, S. Singh and Y. Mansour (1999). "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems **12**.
Tasaki, R., T. Yamazaki, H. Ohnishi, M. Kobayashi and S. Kurosu (2007). "Continuous weighing on a multi-stage conveyor belt with FIR filter." Measurement **40**(7-8): 791-796.
Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin (2017). "Attention is all you need." Advances in neural information processing systems **30**.
Visioli, A. (2006). Practical PID control, Springer Science & Business Media.
Wan, E. A. and R. Van Der Merwe (2000). The unscented Kalman filter for nonlinear estimation. Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373), Ieee.
Wang, Y., P. Zheng, T. Peng, H. Yang and J. Zou (2020). "Smart additive manufacturing: Current artificial intelligence-enabled methods and future perspectives." Science China Technological Sciences **63**(9): 1600-1611.
Watkins, C. J. and P. Dayan (1992). "Q-learning." Machine learning **8**: 279-292.
Wei, Z., T. Zhang, B. Yue, Y. Ding, R. Xiao, R. Wang and X. Zhai (2021). "Prediction of residential district heating load based on machine learning: A case study." Energy **231**: 120950.
Welch, G. and G. Bishop (1995). "An introduction to the Kalman filter."
Widrow, B. and M. E. Hoff (1960). Adaptive switching circuits. IRE WESCON convention record, New York.
Wiener, N. (1949). Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications, The MIT press.
Williams, B. M. and L. A. Hoel (2003). "Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results." Journal of transportation engineering **129**(6): 664-672.
Xu, Y., Y. Sun, J. Wan, X. Liu and Z. Song (2017). "Industrial Big Data for Fault Diagnosis: Taxonomy, Review, and Applications." IEEE Access **5**: 17368-17380.
Yabanova, I. (2016). "Dynamic mass measurement and appropriate filter analysis." IU-Journal of Electrical & Electronics Engineering **16**(2): 3033-3036.

Yabanova, I. (2017). "Digital Signal Processing–based Dynamic Mass Measurement System for Egg Weighing Process." Measurement and Control **50**(4): 97-102.
Yamakawa, Y. and T. Yamazaki (2015). "Modeling and control for checkweigher on floor vibration." Proceedings of the XXI IMEKO.
Yamakawa, Y., T. Yamazaki, J. Tamura and O. Tanaka (2009). "Dynamic behaviors of a checkweigher with electromagnetic force compensation." Proceedings of the XIX IMEKO: 208-211.
Yamazaki, T., Y. Sakurai, H. Ohnishi, M. Kobayashi and S. Kurosu (2002). Continuous mass measurement in checkweighers and conveyor belt scales. Proceedings of the 41st SICE Annual Conference. SICE 2002., IEEE.
Yang, Z., Y. Lu, H. Yeung and S. Krishnamurty (2019). Investigation of Deep Learning for Real-Time Melt Pool Classification in Additive Manufacturing.
Yeung, H., B. Lane and J. Fox (2019). "Part geometry and conduction-based laser power control for powder bed fusion additive manufacturing." Additive manufacturing **30**: 100844.
Zarei, J., M. A. Tajeddini and H. R. Karimi (2014). "Vibration analysis for bearing fault detection and classification using an intelligent filter." Mechatronics **24**(2): 151-157.
Zeng, A., S. Song, J. Lee, A. Rodriguez and T. Funkhouser (2020). "Tossingbot: Learning to throw arbitrary objects with residual physics." IEEE Transactions on Robotics **36**(4): 1307-1319.
Zhu, M., X. Wang and Y. Wang (2018). "Human-like autonomous car-following model with deep reinforcement learning." Transportation Research Part C: Emerging Technologies **97**: 348-368.