MPSI – PCSI

Sciences Industrielles de l'Ingénieur

Séquence 9

Analyser et modéliser pour décrire le comportement d'un système complexe

Pierre-Louis CHIAMBARETTO – Alexandre GARCIA – Sébastien GERGADIER

A3 Analyser l'organisation fonctionnelle et structurelle

A3-01 Associer les fonctions aux constituants.

A3-05 Caractériser un constituant de la chaine de puissance.

A3-07 Analyser un algorithme.

A4 Analyser les performances et les écarts

A4-01 Extraire un indicateur de performance pertinent à partir du cahier des charges ou de résultats issus de l'expérimentation ou de la simulation.

A4-02 Caractériser les écarts entre les performances.

A4-03 Interpréter et vérifier la cohérence des résultats obtenus expérimentalement, analytiquement ou numériquement.

B1 Choisir les grandeurs physiques et les caractériser

B1-01 Identifier les performances à prévoir ou à évaluer.

B1-02 Identifier les grandeurs d'entrée et de sortie d'un modèle.

B2 Proposer un modèle de connaissance et de comportement

B2-01 Choisir un modèle adapté aux performances à prévoir ou à évaluer.

B2-03 Associer un modèle aux composants des chaines fonctionnelles.

B2-10 Déterminer les caractéristiques d'un solide ou d'un ensemble de solides indéformables.

B2-12 Proposer une modélisation des liaisons avec leurs caractéristiques géométriques.

B2-13 Proposer un modèle cinématique paramétré à partir d'un système réel, d'une maquette numérique ou d'un plan d'ensemble.

B3 Valider un modèle

B3-01 Vérifier la cohérence du modèle choisi en confrontant les résultats analytiques et/ou numériques aux résultats expérimentaux.

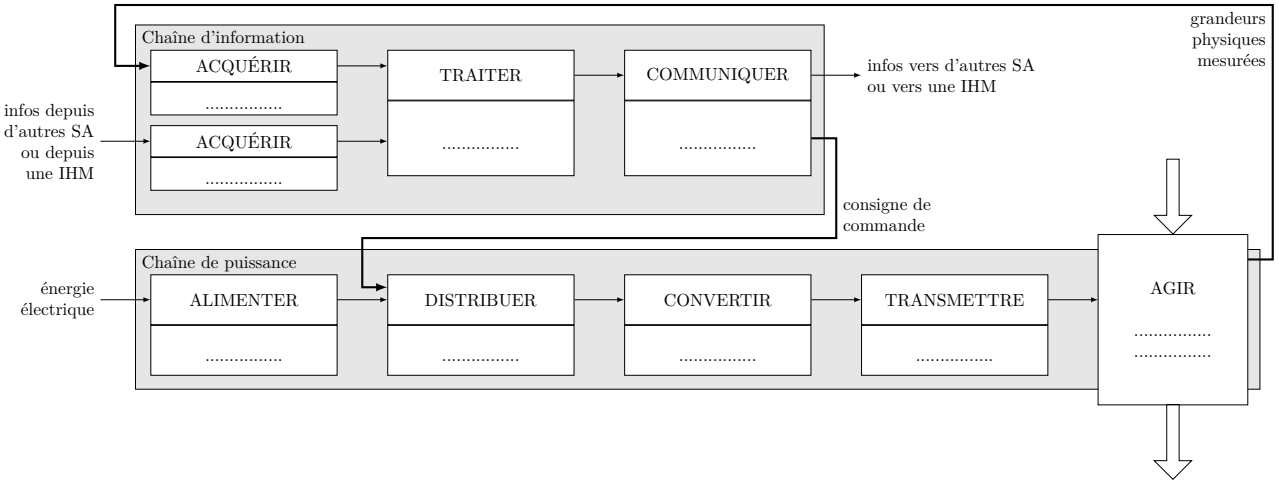
Table des matières

Première partie

Introduction

Outre le dimensionnement du système en termes de performances (conception de la chaîne d'énergie), l'ingénieur doit de plus en plus concevoir la chaîne d'information qui aujourd'hui prend une part très importante dans les systèmes rencontrés.

Les capteurs acquièrent des grandeurs physiques qui sont ensuite traitées par la partie commande qui va ensuite communiquer des ordres à la chaîne d'énergie ou envoyer des informations à des éléments extérieurs.



I Architecture des systèmes de commande

I.1 Rôle d'un système de commande

Depuis le début des années 2000, les progrès technologiques sans cesse croissants des actionneurs (moteurs, vérins, etc.), des protocoles de communications numériques et des capteurs dits « intelligents » ont facilité le développement de *systèmes numériques de contrôle-commande* (SNCC) de plus en plus puissants.

> Un système numérique de contrôle-commande (SNCC, ou DCS pour *distributed control system* en anglais) est une structure programmable utilisée pour le pilotage d'un procédé industriel. Un SNCC est doté d'une interface homme-machine pour la supervision et d'un réseau de communication numérique pour l'interface avec les différents éléments de son environnement tels que, par exemple, les capteurs ou les autres cartes de commande associées à des sous-systèmes.

I.2 Architecture

Les SNCC sont architecturés autour d'une carte de commande principale appelée « carte mère » à laquelle sont connectées plusieurs cartes secondaires, appelées « cartes filles », gérant de manière efficace un nombre limité de tâches, par exemple l'asservissement en vitesse et position de l'arbre de sortie d'un moteur électrique.

Cette organisation nodulaire est simple à concevoir et la division du traitement des informations sur plusieurs cartes présente de nombreux avantages parmi lesquels la fiabilité, les capacités de traitement et de stockage des données acquises par les capteurs ou transmises par les autres cartes, le faible temps de réponse à un événement d'entrée et la possibilité de gestion des communications par la carte mère comme par les cartes filles.

Exemple : Le robot à structure humanoïde NAO®, développé par l'entreprise française Aldebaran Robotics, est un exemple d'un système technique complexe. Pour gérer à la fois les fonctions

de détection de l'environnement, de déplacement fluide et de préhension d'objets, le robot dispose de nombreux capteurs (sonores, visuels et tactiles), de 25 axes asservis en position et en vitesse de rotation et d'un calculateur central gérant l'ensemble des fonctions.

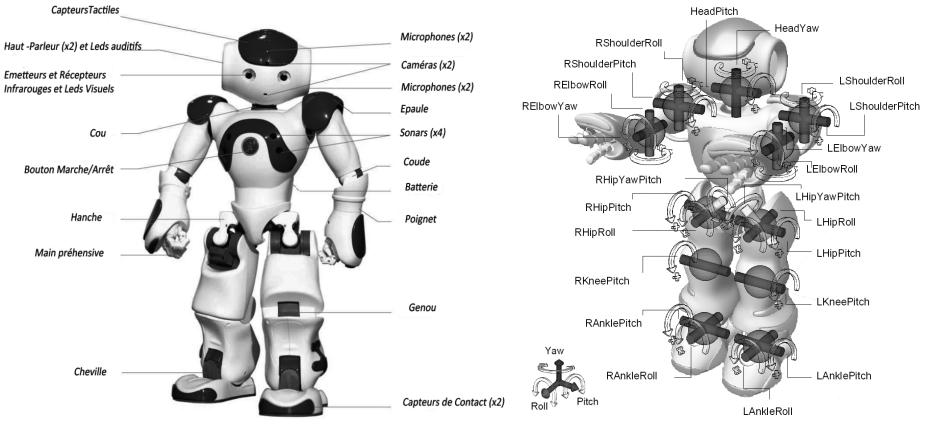


Figure 1 : Le robot NAO® de l'entreprise ALDEBARAN ROBOTICS.

Lors, par exemple, du déplacement en ligne droite de ce robot humanoïde, au moins 16 axes sont sollicités (les 6 de chaque jambe + les 2 de chaque épaule afin d'assurer l'équilibre) : il est donc impossible que chacun de ces axes ait un comportement individuel non contrôlé globalement par un calculateur central. Pour améliorer les capacités du robot et gérer efficacement les flux d'informations, les 25 axes ne peuvent être gérés par un seul et même calculateur : l'utilisation d'un SNCC est donc indispensable.
Dans le cas du robot NAO®, le SNCC est organisé autour d'une carte mère à processeur Intel ATOM® et embarquant un système d'exploitation Linux (noyau 2.6 temps réel). Chacun des 25 axes est géré par une carte microcontrôleur dédiée et connectée au calculateur central par un bus de données. La programmation du robot, connecté à un ordinateur par une connexion WiFi ou Ethernet, se fait par une interface graphique reproduisant la structure d'un diagramme d'activités (voir cours SysML). Le paramétrage des blocs de cette interface se fait en langage Python. La création des blocs et de nouvelles activités se fait dans de nombreux langages parmi lesquels C, C++, Python, Java et Matlab.

La réalisation d'un système de commande s'appuie sur des technologies numériques qui évoluent très rapidement. Les performances s'améliorent chaque jour et autorisent des applications toujours plus pointues.

La commande numérique est fondamentalement constituée de portes logiques et de mémoires, qui manipulent de l'information binaire. Cette information est matérialisée électriquement, le plus souvent par un potentiel nul (0 logique) et un potentiel de quelques volts (1 logique).

I.3 Nature des informations

Les systèmes numériques ne manipulent que des grandeurs binaires logiques donc de type « tout ou rien » : les entrées et les sorties de ses composants ne peuvent prendre que deux états. Une succession de données binaires sera nommée grandeur numérique (ou discrète) par opposition à une grandeur analogique ou continue.
Les capteurs sont toujours utilisées pour mesurer des grandeurs physiques continues (pression, température, position,...). Ces capteurs délivrent ensuite une grandeur qui doit être échantillonnée puis quantifiée pour être utilisée par un système de commande numérique.
En général, on convertit les signaux de nature non-électrique en signaux électriques, et vice-versa, à l'aide d'un transducteur comme un microphone, un haut-parleur, une caméra numérique, un écran d'ordinateur, une antenne, un thermomètre électronique, un indicateur de débit

ou un accéléromètre. Ensuite, on mesure l'intensité électrique correspondant au phénomène ou à la quantité physique pour en indiquer la valeur à un moment précis. Finalement, on convertit cette intensité électrique en un nombre pouvant être traité par le système numérique. Ce processus s'appelle la numérisation d'un signal.

I.4 Définitions

I.4.1 Variables binaires

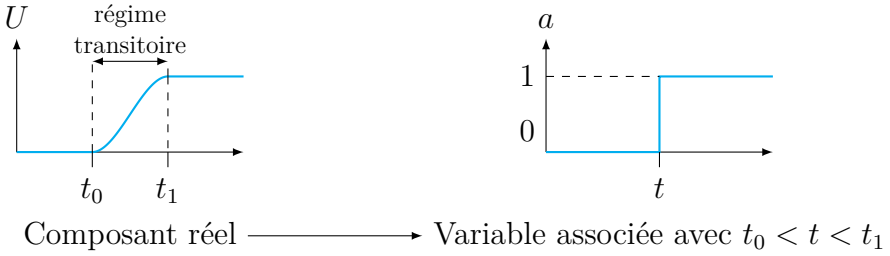
De nombreux composants utilisés en automatisme ne peuvent normalement prendre que deux états différents : lampe allumée ou éteinte, bouton-poussoir actionné ou relâché, moteur tournant ou à l'arrêt, vérin pneumatique sorti ou rentré...

A chacun de ces composants, on peut associer une **variable binaire** (ou logique, ou Tout Ou Rien) qui ne peut prendre que deux valeurs notées 0 ou 1 (vrai ou faux, oui ou non).

Une variable a est binaire si et seulement si elle ne peut prendre, à chaque instant, qu'une seule valeur parmi un ensemble de 2 valeurs possibles.

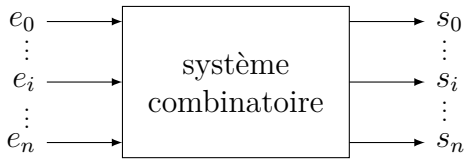
Remarque :

- le comportement « Tout ou Rien » (TOR) ne correspond qu'au comportement normalement prévu en régime stabilisé et en l'absence de tout dysfonctionnement.
- l'association d'une variable binaire à un composant ne peut pas rendre compte des états transitoires apparaissant entre deux états stables. C'est donc une simplification du comportement réel.



I.4.2 Système logique

Un **système** est dit **binaire** ou **logique** si les variables d'entrée et de sortie sont binaires.



I.4.3 Systèmes logique combinatoire et séquentiel

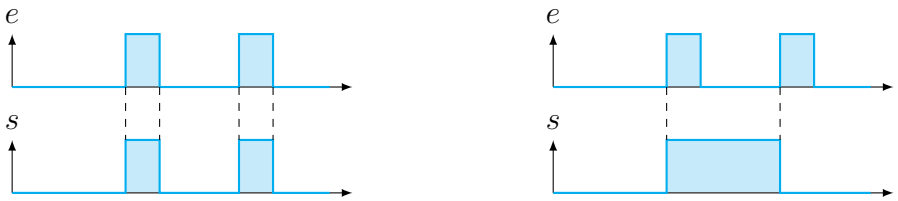
Un **système logique combinatoire** est un système binaire pour lequel à un état des variables d'entrée e_i correspond un unique état des variables de sortie s_j. (La réciproque n'est pas vraie)

En revanche, un **système logique** est dit **séquentiel** si les sorties s_j ne dépendent pas uniquement des entrées e_i mais aussi de l'évolution antérieure du système (historique).

La FIGURE 2(a) illustre l'appuie sur un interrupteur à deux positions :
- sur la position 0 (entrée à 0), la lumière est éteinte (sortie à 0) ;
- sur la position 1 (entrée à 1), la lumière est allumée (sortie à 1).

En effet, à un état des e_i peuvent alors correspondre plusieurs états des s_j. Il y a alors nécessité de faire apparaître de nouvelles variables internes (appelées mémoires).

La FIGURE 2(b) illustre l'appuie sur un interrupteur de type bouton poussoir :
- sur la position 0 (entrée à 0), la lumière est éteinte (sortie à 0) ;
- sur la position 1 (entrée à 1, appui sur le bouton), la lumière s'allume (sortie à 1) ;
- sur la position 0 (entrée à 0, relâchement du bouton), la lumière reste allumée (sortie à 0) ;
- sur la position 1 (entrée à 1, appui sur le bouton), la lumière s'éteint (sortie à 0).



(a) Système logique combinatoire

(b) Système logique séquentielle

FIGURE 2 : Différence entre système logique combinatoire et séquentiel.

Deuxième partie

Modélisation du comportement séquentiel des systèmes par diagrammes d'états

I Systèmes séquentiels, à évènements discrets et machines d'états

I.1 Introduction

Les systèmes à évènements discrets (SED) permettent de modéliser la succession des différentes étapes d'évolution d'un système. Ce sont des systèmes séquentiels dans lesquels une même cause peut produire des effets différents (par opposition aux systèmes combinatoires).

Les variables de sorties d'un **système combinatoire** s'écrivent mathématiquement par la relation suivante :

$$S(t) = f\Big(E(t)\Big)$$

Les variables de sorties d'un **système séquentiel** s'écrivent mathématiquement par la relation suivante :

$$S(t) = f\Big(E(t), S(t - \Delta t)\Big)$$

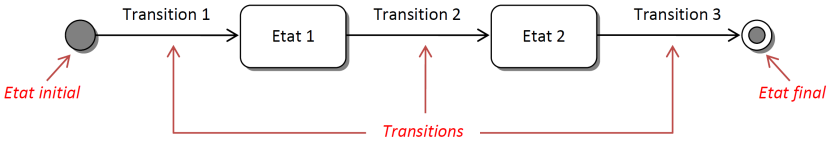
I.2 Machines d'états (`stm`) et diagrammes d'états SysML

Pour modéliser le fonctionnement des SED, nous allons utiliser le concept de machines d'états (ou `stm` de l'anglais STate Machine). L'outil de modélisation SysML permet, grâce au diagramme d'états, de représenter l'évolution d'un des blocs d'un diagramme de définition de blocs (`bdd`) ou d'un diagramme de blocs internes (`ibd`).

> Un **état** modélise une phase de fonctionnement du système. Pendant cette phase, le système accomplit une ou plusieurs actions et/ou activités, ou attend un évènement. Il peut être actif ou non.

> Un **diagramme d'états** du langage SysML donne une représentation graphique des transitions possibles entre les différents états d'une machine d'états.

L'objectif de ce diagramme est de décrire les différents états pris par un bloc en fonction des évènements qui lui arrivent. Le passage d'un état à un autre se fait en franchissant une transition.



> Les éléments de base sont donc : l'**état** avec ses **actions** et **activités**, la **transition** et les **évènements**.

Les éléments graphiques utilisés dans ce diagramme sont principalement des rectangles aux coins arrondis pour les états et des flèches pour les transitions. Les flèches sont orientées à partir d'un état source et vers un état cible. La transition est franchie (passage de l'état **source** à l'état **cible**) lors de l'apparition de l'événement rattaché à la transition.

I.3 Règles d'évolution

— Le point de départ de la machine d'états est indiqué par l'**état initial** ;
— Les transitions sont franchies en fonction de l'activité des états sources et des transitions ;
— Un seul état par machine d'états ou sous-machine d'états est actif à la fois. Différentes structures de machines d'états (voir plus loin) permettent d'avoir plusieurs états actifs dans des machines d'états distinctes.

II Les différents types d'états et de structures

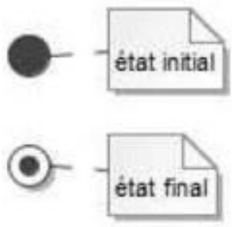
II.1 États usuels

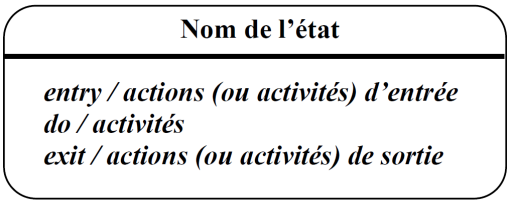
Dans les diagrammes d'états, on trouve principalement trois sortes d'états :
— état initial – pseudo-état qui indique un point d'entrée dans un graphe. Il n'y en a qu'un seul par machine d'états ;
— état final – pseudo état non obligatoire, il indique que le système décrit n'a plus d'état actif. Il peut y en avoir plusieurs dans un diagramme d'états. En effet, plusieurs scénarios peuvent être possibles pour mettre fin à un comportement ;
— état générique – état courant, on peut principalement lui rattacher des **activités**, des **actions** ou **activités en entrée et en sortie** d'un état.

> Une **activité** prend du temps et peut être interrompue par un événement. On la trouve à l'intérieur des états par le mot « `do` ». Une activité est donc dite interruptible. Elle peut se finir naturellement seule (dans ce cas elle est dite finie) ou nécessiter un événement pour se terminer.

Par exemple `do / émettre son musique` est une activité.

> Une **action** ne prend pas de temps et ne peut pas être interrompue. Son exécution peut par exemple provoquer un changement d'état, l'émission d'un ordre ou un retour de valeur. Ces actions ne sont appelées que par les mots clé « `entry` » ou « `exit` ».





Le nom de l'état est indiqué dans la partie haute du rectangle. La partie basse est réservée aux activités et actions.
Par exemple, `entry / C:=C+1` et `exit / eclairage:= 0` sont des actions.

`entry`	Ex : `entry / activite1`	Le front `entry` apparaît à l'activation de l'état.
`do`	Ex : `do / activite2`	L'`activite2` s'exécute tant que l'état est actif après la terminaison de l'`activite1`.
`exit`	Ex : `exit / activite3`	Le front `exit` apparaît à désactivation de l'état. L'`activite2` est interrompue, et l'`activite3` est exécutée.

Remarque : Il est possible de ne définir aucun comportement dans l'état, cette structure vide pouvant indiquer par exemple un état d'attente.

II.2 Transitions et événements associés usuels

II.2.1 Structure de base d'une transition

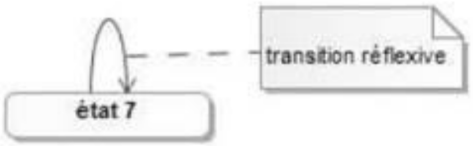
Les possibilités d'évolution entre états sont représentées par des flèches qui indiquent le sens d'évolution.

Ces flèches portent (ou non) un message sous la forme `événement[garde]/effet` qui définit la transition avec son évènement déclencheur, sa condition de garde et l'effet associé (action), ces termes étant détaillés un peu plus loin.

L'état de départ est appelé l'**état source** et l'état d'arrivée est appelé l'**état cible**. L'activité de l'état source de la transition est nécessaire au franchissement de la transition.

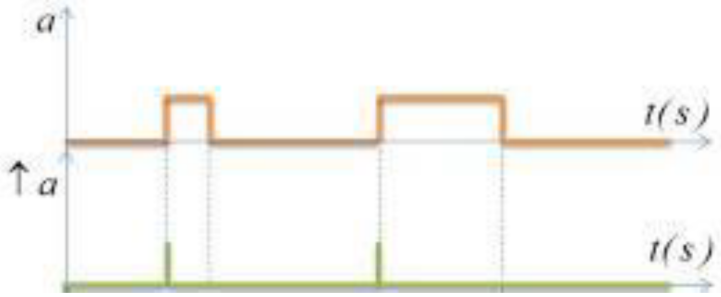
Une transition réflexive entraîne une sortie d'état puis un retour dans ce même état. Lors de l'événement E1 (voir figure ci-dessous), l'état 7 reste actif et :

— L'activité en cours (associée au comportement do) est interrompue,
— L'action associée au comportement `exit` est exécutée,
— L'action associée au comportement `entry` est exécutée,
— L'activité associée au comportement `do` est lancée à nouveau.



II.2.2 Notion d'événement

Un **évènement** est un changement qui doit être pris en compte dans la description par diagramme d'états. Un évènement se produit à une date précise qui est l'occurrence de cet évènement. **La durée d'un évènement est nulle et il ne peut pas être mémorisé**. Un événement est le passage de 0 à 1 d'une variable. On peut lui associer la notion de front montant (noté ci-dessous ↑a).

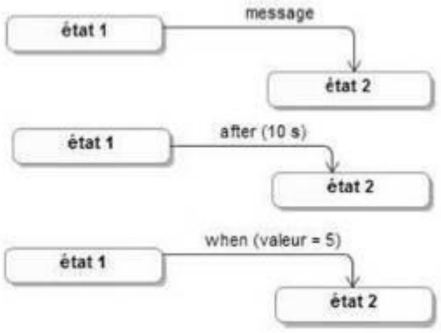


L'occurrence de l'événement est une condition nécessaire au franchissement.

Remarque : [Front descendant]Pour désigner un événement correspondant au front **descendant** d'une variable `a`, on utilisera l'évènement $\bar{a}$.

Il existe trois types principaux d'évènements associés à une transition :

- l'événement de signal (`signal event`) : un message asynchrone (cela signifie que le destinataire ne l'attend pas forcément. Par exemple appui sur un bouton) est arrivé ;
- l'évènement temporel (`time event`) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « `after` ») ou un temps absolu a été atteint (mot clé « `at` ») ;
- l'évènement de changement (`change event`) : une valeur a changé de telle sorte que la transition est franchie (mot clé « `when` »).

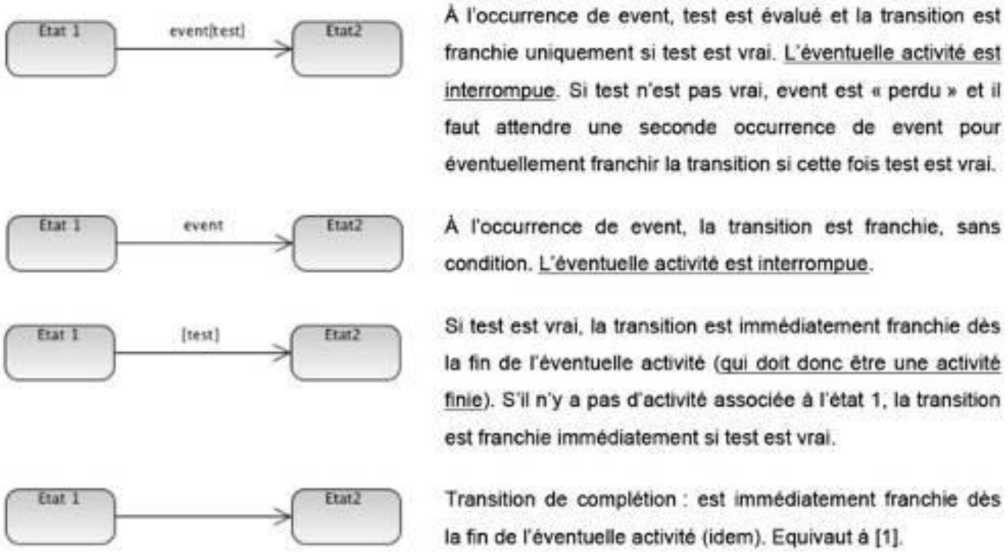


II.2.3 La condition de garde

La condition de garde est la deuxième condition de franchissement.

> Une **condition de garde** est une expression booléenne encadrée de crochets, évaluée lorsque l'état précédant la transition est actif ET que l'évènement déclencheur se produit. Si la condition de garde est vraie, la transition est alors franchie, sinon elle ne l'est pas et l'événement est perdu.

Remarque : [Différence fondamentale entre évènement et condition de garde]Un événement est parfaitement daté dans le temps (cf front montant). Une condition de garde n'est pas datée, elle doit être vraie (niveau logique 1) à l'instant où l'événement survient pour que la transition soit franchie. Si la condition de garde est fausse lors de l'évènement, il faudra attendre le prochain évènement pour espérer évoluer.
Exemples de transition :



II.2.4 Notion d'effet

> Un **effet** est un comportement (action ou activité) accompli lorsque la transition est franchie.

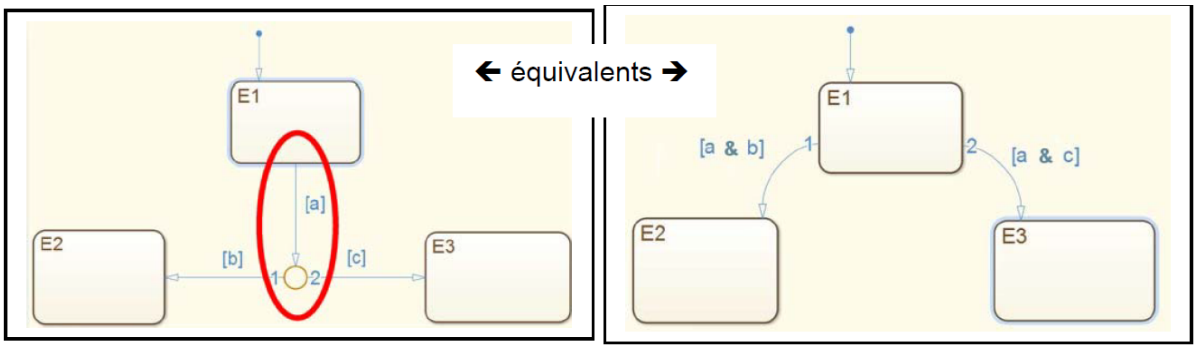
L'action (ou activité) associée à l'effet est exécutée AVANT d'exécuter les actions (ou activités) associées à l'état cible.

II.2.5 Transition basée sur un état interne

On peut utiliser l'activité d'un état comme variable dans une condition de garde. Par exemple, la condition de garde `[in Etat1]` est vraie (`[in Etat1]=1`) si l'état intitulé « `Etat1` » est actif, et vaut 0 sinon.

II.3 Autre symbole utile

On peut trouver dans les graphes d'états les points de décision  , qui doivent être suivis de deux transitions sous forme de condition de garde complémentaires, donc non compatibles. Il est d'ailleurs souvent utile d'utiliser la condition de garde `[else]`. La durée d'exécution est nulle et aucune activité ni action n'y est associé.



III Structures et fonctions avancées

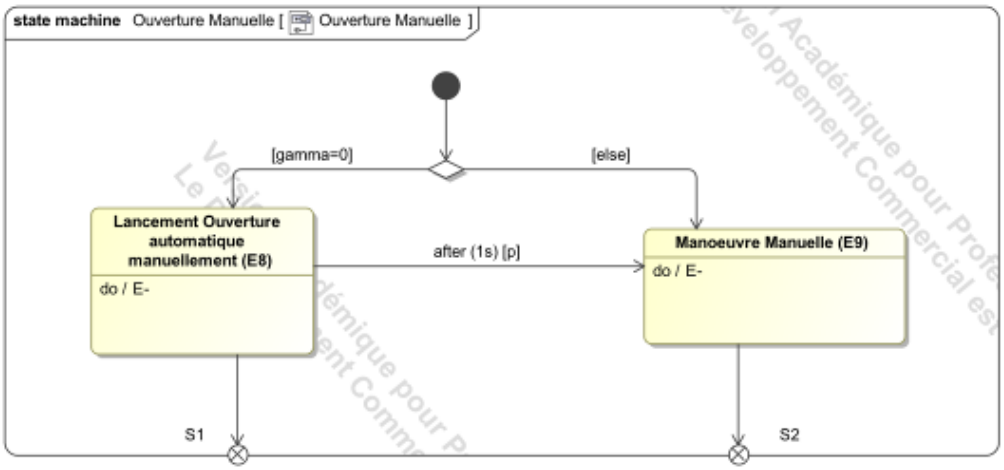
III.1 État composite

Lorsque le comportement à décrire :

— risque d'être peu lisible car trop complexe ;

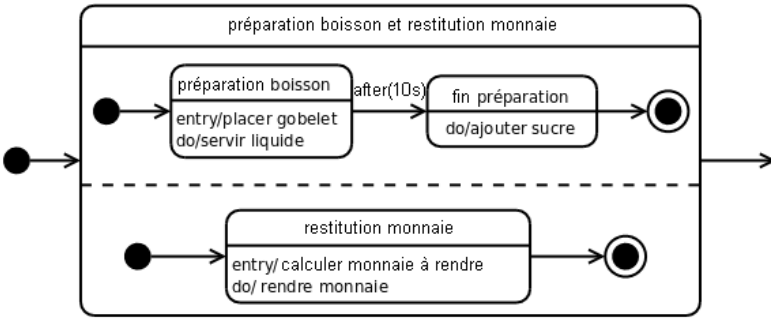
— fait apparaître des parties "séparables" de l'ensemble.

Alors on extrait ces parties pour en faire des sous-graphes séparés, hiérarchiquement inférieurs au graphe principal. On englobe ces sous-graphes dans un état appelé **état composite**. Un état composite est composé de plusieurs sous-états internes. Un état composite possède un état initial.



III.2 Notion d'état orthogonal

Un **état orthogonal** possède plusieurs **régions**, séparées par des **pointillés** et possédant chacune sa propre description d'états. Les différentes régions d'un état orthogonal fonctionnent **toutes en parallèle** sans aucune influence les unes sur les autres.



Remarque : Ainsi, avec un état orthogonal, il est possible d'avoir plusieurs états actifs à un instant t donné. Cela permet de réaliser plusieurs tâches en parallèle. Chaque région doit **obligatoirement** avoir un état initial (qui sont activé lors de l'activation de l'état orthogonal), et peut avoir un état final ou non.

La désactivation de l'état orthogonal entraîne la désactivation de l'ensemble des états de chaque région. Cette première est obtenue, par exemple, lorsque l'un des états finaux d'une région est actif, et que l'éventuelle transition de sortie est vraie. Ainsi, l'utilisation d'une transition basée sur les états internes de chaque région peut-être pertinente, afin de synchroniser la sortie de l'état orthogonal.

Troisième partie

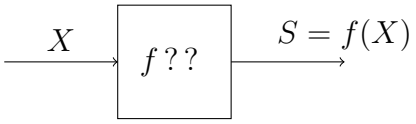
Modélisation du comportement des systèmes par Intelligence Artificielle

I Pourquoi et comment une IA ?

I.1 Introduction

Objectif

L'objectif de l'Intelligence Artificielle (IA) est de prévoir la (ou les) grandeur(s) de sortie S à partir d'une (ou plusieurs) entrée(s) X et d'une fonction f inconnue initialement.



Pour que cette prévision soit possible, il est nécessaire de disposer de la fonction f, appelée **modèle de prévision**. Pour cela, on suppose disposer d'une base de données d'apprentissage, qui permettra de déterminer la fonction f.

I.2 Les différentes phases liées à l'IA

Comme expliqué, la première phase dans l'IA est la **phase d'apprentissage**. L'objectif de celle-ci est d'obtenir **un** modèle de prévision, autrement dit de la fonction f. Elle est obtenue à partir d'une partie de la base de données d'apprentissage (paires $S-X$ fournies par l'utilisateur par exemple). Cette partie est généralement appelée **données d'entraînement**.

La seconde phase, appelée **phase de test**, a pour objectif de valider (ou non) le modèle, en utilisant un ou plusieurs indices de performances. Cette validation est réalisée généralement sur une partie de la base de données d'apprentissage, appelée **données de test**.

Enfin, lorsque le modèle obtenu est satisfaisant, il est possible de procéder à la **phase d'inférence**, qui correspond à l'utilisation du modèle sur de nouvelles données.

I.3 Les différents apprentissages

On distingue principalement 3 types d'apprentissage :

- l'apprentissage **supervisé**. Cet apprentissage est basé sur le fait que les données d'entraînement sont étiquetées ou annotés. C'est-à-dire, dans le cadre de la reconnaissance d'animaux, toutes les photos de la base de données sont étiquetées par le nom de l'animal. Le but est alors de construire une fonction f qui donne le nom d'un animal S à partir de sa photo, par exemple, X ;
- l'apprentissage **non supervisé**. Cet apprentissage est basé sur le fait que les données d'entraînement ne sont pas étiquetées ou annotées. C'est-à-dire, dans le cadre de la reconnaissance d'animaux, la base de données contient des photos d'animaux (en un nombre fini d'espèces d'animaux) mais pour chaque photo la connaissance du type d'animal n'est pas connu. Le but est de construire une fonction f qui attribue une catégorie S à chaque photo X, par exemple ;
- l'apprentissage **par renforcement**. Cet apprentissage est basé sur le fait de prédisposer d'un modèle de prévision, et que celui-ci va évoluer potentiellement à chaque nouvelle prédiction dans la phase d'inférence. Ce type d'approche est adaptée à la construction d'algorithmes qui doivent prendre des successions de décisions dans un environnement changeant.

I.4 Problématiques liées aux données d'apprentissage

Les problématiques liées aux données d'apprentissage sont principalement au nombre de 2.

— La **quantité de données**. Afin de déterminer la fonction de prédiction durant la phase d'apprentissage, il est nécessaire de disposer d'une quantité importante de données d'entraînement. En effet, plus il y aura de données d'entraînement, plus le modèle sera capable, a priori, de fournir la « bonne » grandeur de sortie. Toutefois, plus le nombre de données d'entraînement est grand, plus la phase d'apprentissage durera longtemps (sauf pour l'algorithme k-NN, voir section III). Il n'est pas inhabituel de lancer des processus d'apprentissage qui durent plusieurs heures, voire même plusieurs jours ;

— La **qualité des données**. Indépendamment de la quantité des données, il faut s'assurer que celles-ci soient de « bonnes » qualités. En effet, il faut éviter tout **biais** dans les données d'entraînement.
Les biais peuvent être multiples, à titre d'exemples, dans le cas de la classification d'animaux, il faut prendre garde à ce que le nombre de données d'entraînement associées à chaque type d'animal soit identique.
Un autre exemple, dans la catégorie des chiens, il faut fournir des données associées à chaque type (classe) de chiens, et pas que des épagneuls bretons, ou que des chiens blancs...
Un dernier exemple, dans le cas de la classification, concerne la normalisation des données. Supposons que les données d'apprentissage contiennent une colonne avec des valeurs allant de 0 à 1 et une autre colonne avec des valeurs allant de 10 000 à 100 000. Une petite variation, par exemple de 0.1, pourra avoir une grande influence vis-à-vis de la première colonne, et quasiment aucune sur la seconde colonne.

Remarque : Il est donc très important de pré-traiter les données d'apprentissage afin d'obtenir un modèle de prévision pertinent et ainsi réduire les biais d'apprentissage. Et souvent, pour l'humain, c'est la phase la plus chronophage de la phase d'apprentissage.

II Classification ou régression ?

L'Intelligence Artificielle a pour objectif de prévoir (ou prédire) la grandeur de sortie S, à partir d'un modèle de prévision (fonction f) et d'une nouvelle donnée d'entrée X. Cette prévision peut être de 2 types, de la classification ou de la régression.

II.1 La classification

La **classification** consiste à déterminer la classe d'appartenance la plus probable parmi les différentes classes possibles d'une donnée d'entrée. Par exemple, pour la reconnaissance des panneaux de signalisation, la classification consistera à déterminer s'il s'agit « a priori » d'un panneau STOP, Cédez-le-passage, Sens interdit, Sens unique, Voie sans issue ... Le modèle de prévision est appelé généralement **classifieur**.

II.2 La régression

La **régression** consiste à déterminer une (ou plusieurs) grandeur(s) de sortie continue(s) à partir d'une ou plusieurs données d'entrée. Par exemple, la régression peut consister à évaluer le prix d'un bien immobilier en fonction de sa surface, de sa situation géographique, du nombre de pièces, du nombre de salles de bain, des revenus financiers de ses habitants voisins ...

II.2.1 La régression linéaire monovariable

La grandeur d'entrée X est un scalaire (monovariable), et la grandeur de sortie S est le résultat d'une application affine de l'entrée. Ainsi, la fonction de prévision s'écrit sous la forme

$$S = f(X) = a \cdot X + b$$

L'algorithme associé à la régression linéaire monovariable consiste à déterminer les coefficients a et b optimaux.

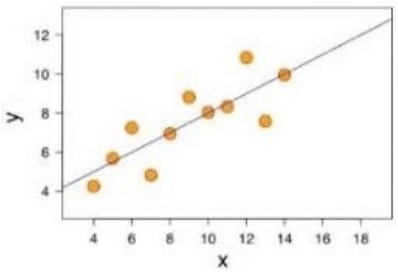


FIGURE 3 : Régression linéaire monovariable

II.2.2 La régression linéaire multivariable

La grandeur d'entrée X est un vecteur qui est composé de n variables $x_1, x_2 \dots x_n$ (multivariable), et la grandeur de sortie S est le résultat d'une application affine de l'entrée. Ainsi, la fonction de prévision s'écrit sous la forme

$$S = f(X) = a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n + b$$

L'algorithme associé à la régression linéaire multivariable consiste à déterminer les coefficients a_i pour $i \in [\![1, n]\!]$ et b optimaux.

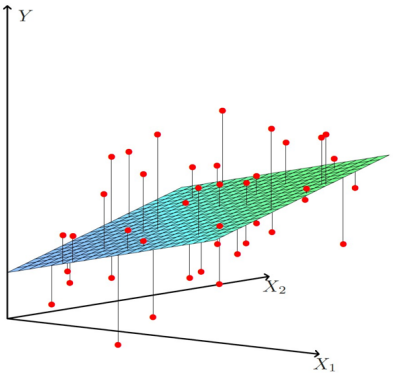


FIGURE 4 : Régression linéaire multivariable

II.2.3 La régression non-linéaire monovariable

La grandeur d'entrée X est un scalaire (monovariable), et la grandeur de sortie S est une application non-linéaire de l'entrée. Généralement, la non linéarité est du type polynomiale (avec n préalablement défini). Ainsi, la fonction de prévision s'écrit sous la forme

$$S = f(X) = a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n + b$$

L'algorithme associé à la régression non-linéaire (polynomiale) monovariable consiste à déterminer les coefficients a_i pour $i \in [\![1, n]\!]$ et b optimaux.

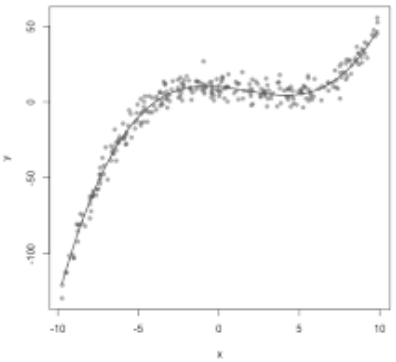


FIGURE 5 : Régression non-linéaire (polynomiale) monovariable

II.2.4 La régression non-linéaire multivariable

La grandeur d'entrée X est un vecteur composé de n variables $x_1, x_2 \dots x_n$ (multivariée), et la grandeur de sortie S est une application non-linéaire des entrées. Généralement, la non linéarité est du type polynomiale (avec n préalablement défini) et des termes éventuellement croisés ($x_i \times x_j$ avec $i \neq j$). Ainsi, la fonction de prévision s'écrit sous la forme

$$\begin{aligned} S = f(X) = & a_{11} \cdot x_1 + a_{12} \cdot x_1^2 + \ldots + a_{1n} \cdot x_1^n \\ & + a_{21} \cdot x_2 + a_{22} \cdot x_2^2 + \ldots + a_{2n} \cdot x_2^n + \\ & \ldots \\ & + b_{12} \cdot x_1 x_2 + b_{13} \cdot x_1 x_3 + \ldots + a_{1n} \cdot x_1 x_n + \\ & \ldots \\ & + b \end{aligned}$$

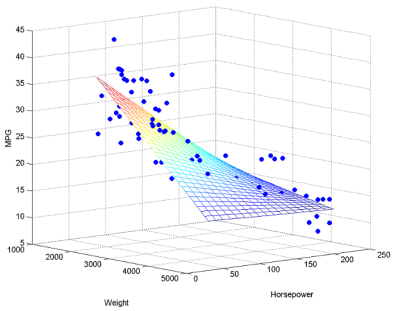


FIGURE 6 : Régression non-linéaire (polynomiale) multivariable

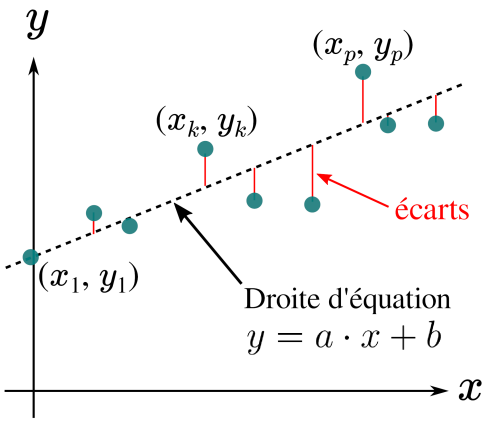
L'algorithme associé à la régression non-linéaire (polynomiale) multivariable consiste à déterminer les coefficients a_{ij}, b_{ij} et b optimaux.

Remarque : [Optimalité des paramètres]Quel que soit la régression utilisée, l'objectif des algorithmes est des fournir les coefficients optimaux. Ceux-ci sont généralement ceux qui minimisent une fonction score dont la valeur est notée J. Cette fonction score représente généralement la somme des carrés des écarts entre les valeurs prédites par le modèle de prévision ($f(X)$) et les données d'apprentissage (y_i).

$$\text{score} = J = \sum_{i=1}^{i=n} \Big(y_i - f(X) \Big)^2$$

II.2.5 Exemple de détermination des coefficients optimaux pour une régression linéaire monovariable

Soit le résultat d'essais expérimentaux, qui à une entrée x, fournit la grandeur de sortie y.



On souhaite modéliser, par une fonction affine (régression linéaire monovariable), son comportement. Ainsi, le modèle proposé est la droite d'équation $y = a\,x + b$ qui minimise les écarts avec un ensemble de points (x_k, y_k), $\forall\, k \in \{1, \ldots, N\}$.

La fonction score associée est la somme des erreurs de prédiction au carré. Cette méthode est aussi appelée la méthode des moindres carrés. Donc :

$$J = \sum_{k=1}^{N} (y_k - (a\,x_k + b))^2$$

J étant différentiable, ses extremas (on admet que ceux-ci sont des minimas) étant ceux pour lesquels son gradient s'annule. Donc, les coefficients optimaux a et b sont les solutions du système suivant :

$$\begin{cases} \dfrac{\partial J}{\partial a} &= -2 \sum\limits_{k=1}^{N} (y_k - a\,x_k - b) = 0 \\ \\ \dfrac{\partial J}{\partial b} &= -2 \sum\limits_{k=1}^{N} x_k\,(y_k - a\,x_k - b) = 0 \end{cases}$$

Soit donc à résoudre le système suivant :

$$\begin{cases} \sum\limits_{k=1}^{N} y_k - a \sum\limits_{k=1}^{N} x_k - N\,b &= 0 \\ \\ \sum\limits_{k=1}^{N} x_k\,y_k - a \sum\limits_{k=1}^{N} x_k^2 - b \sum\limits_{k=1}^{N} x_k &= 0 \end{cases}$$

En notant $\overline{x} = \dfrac{1}{N} \sum\limits_{k=1}^{N} x_k$ la moyenne empirique des x_k (et $\overline{y}$ celle des y_k), on obtient :

$$\begin{cases} a &= \dfrac{\sum x_k \sum y_k - N \sum x_k\,y_k}{\left(\sum x_k\right)^2 - N \sum x_k^2} = \dfrac{\overline{x} \cdot \overline{y} - \overline{x\,y}}{\overline{x}^2 - \overline{x^2}} \\ \\ b &= \dfrac{1}{N} \sum y_k - a \dfrac{1}{N} \sum x_k = \overline{y} - a\,\overline{x} \end{cases}$$

En notant :

— $Var(x)$ la variance définie par

$$Var(x) = \frac{1}{N} \sum_{k=1}^{N} \left(x_k - \overline{x}\right)^2 = \frac{\sum\limits_{k=1}^{N} x_k^2}{N} - \overline{x}^2$$

— $Cov(x, y)$ la covariance définie par

$$Cov(x, y) = \frac{1}{N} \sum_{k=1}^{N} (x_k - \overline{x}) \cdot (y_k - \overline{y}) = \frac{1}{N} \sum_{k=1}^{N} x_k y_k - \overline{x} \cdot \overline{y}$$

On en déduit que l'équation de la droite de régression linéaire est :

$$y = \underbrace{\frac{Cov(x, y)}{Var(x)}}_{a} \cdot x + \underbrace{\overline{y} - \frac{Cov(x, y)}{Var(x)} \cdot \overline{x}}_{b}$$

II.3 Validation d'un modèle de classification

Un des outils généralement utilisés pour la validation des performances d'un classifieur est la matrice de confusion. Elle permet de représenter sous forme synthétique (tableau) les classes prédites par le modèle de prévision (*Predicted Class*) et les labels des données de test (*True Class*).

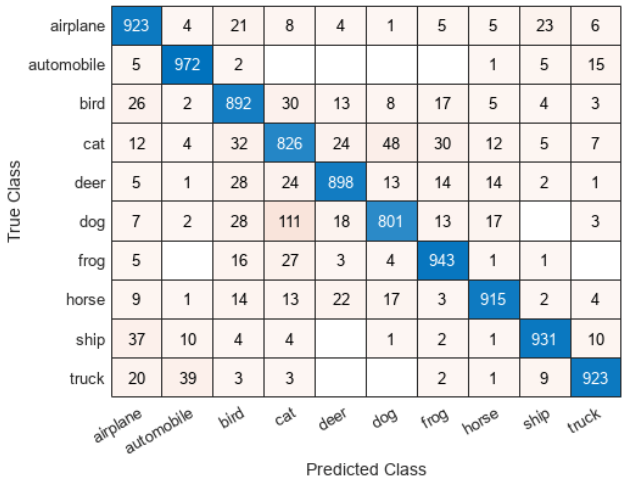


FIGURE 7 : Matrice de confusion

Remarque : Dans le cas d'un classifieur idéal, la matrice de confusion est diagonale, **ET** chaque valeur de la diagonale correspond au nombre d'éléments de la classe concernée dans les données de test.

Remarque : La somme totale des valeurs dans la matrice de confusion est égale au nombre de données de test.

Remarque : Pour une ligne donnée (classe réelle) de la matrice de confusion, la somme des valeurs correspond au nombre de jeu de test de la classe considérée.

Afin de valider (ou non) un modèle de classification, il est donc nécessaire de définir des critères de performances (appelés parfois métriques). Ces critères sont définis pour chaque classe.

Une **prédiction est dite positive** pour la classe i si le modèle de prévision prévoit la classe i pour une entrée X. Elle est **dite négative** pour la classe i si le modèle prévoit toute autre classe de sortie.

II.3.1 Vrais positifs et négatifs

Les **vrais positifs** (VP) indiquent les cas où les prédictions et les valeurs réelles sont effectivement positives.

Les **vrais négatifs** (VN) indiquent par contre les cas où les prédictions et les valeurs réelles sont toutes les deux négatives.

II.3.2 Faux positifs et négatifs

Les **faux positifs** ou FP (*False Positive*) indiquent quant à eux une prédiction positive contraire à la valeur réelle qui est négative.

Les **faux négatifs** ou FN (*False Negative*) font référence aux cas où les prédictions sont négatives alors que les valeurs réelles sont positives.

II.3.3 Le taux d'erreur et Accuracy

Le **taux d'erreur** ou ERR (*ERror Rate*) est une métrique qui est calculée en faisant la somme de toutes les prédictions incorrectes sur le nombre total de données (positives et négatives). Plus il est bas, meilleure est la classification. Le meilleur taux d'erreur possible est de 0, mais il est rarement atteint par un classifieur dans la pratique.

$$ERR = \frac{FP + FN}{VP + VN + FP + FN}$$

L'**exactitude** (*accuracy*) fait la somme de tous les vrais positifs et vrais négatifs qu'il divise par le nombre total d'instances. Il permet d'apporter une réponse à la question suivante : de toutes les classes positives et négatives, combien parmi elles ont été prédites correctement ? Des valeurs élevées de ce paramètre sont souvent souhaitables. Il peut également être calculé avec la formule suivante :

$$Accuracy = 1 - ERR = \frac{VP + VN}{VP + VN + FP + FN}$$

II.3.4 La sensibilité et la spécificité

La **sensibilité** (appelée parfois rappel) indique le rapport entre les prévisions positives correctes et le nombre total de prévisions positives. Ce paramètre répond donc à la question suivante : sur tous les enregistrements positifs prédits, combien sont réellement positifs ?

La **spécificité** (appelée parfois précision) ou TNR (*True Negative Rate*) caractérise le ratio entre le nombre de prédictions négatives correctes et le nombre total de cas réels négatifs.

$$\text{Sensibilité} = \frac{VP}{VP + FN} \qquad \text{et} \qquad \text{Spécificité } = \frac{VN}{VN + FP}$$

Remarque : Sur un problème de classification à 2 classes ($S \in \{0, 1\}$) :

- le modèle qui prévoit toujours la classe 1 possède une sensibilité de 1 (car FN=0) et une spécificité de 0 (car VN=0) ;
- le modèle qui prévoit toujours la classe 0 possède une sensibilité de 0 (car VP=0) et une spécificité de 1 (car FP=0).

Concrètement, et de façon opérationnelle, généralement, on définit dans un cahier des charges une valeur minimale à atteindre pour la spécificité ou la sensibilité, et on cherche à rendre maximale l'autre.

II.4 Validation d'un modèle de régression

II.4.1 Racine carrée de l'erreur quadratique moyenne

La **Racine carrée de l'erreur quadratique moyenne** ou RMSE (*Root Mean Squared Error*) consiste à calculer pour chaque point x_i des données de test la distance entre la valeur prédite ($f(x_i)$) et sa valeur vraie y_i (dans les données de test) et en faire la somme. Pour se ramener à l'unité de y, on la normalise par le nombre de données de test, et on en prend la racine carrée.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\Big(f(x_i) - y_i\Big)^2}$$

II.4.2 Coefficient de détermination R^2

Le coefficient de détermination R^2 indique à quel point les valeurs prédites sont corrélées aux vraies valeurs.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2}{\sum_{i=1}^{n} \left(y_i - \overline{y}\right)^2} \qquad \text{avec} \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Ce coefficient, compris entre 0 et 1, permet ainsi de **caractériser la corrélation entre les valeurs prédites par le modèle et les valeurs réelles des données de test**, comme l'illustre la FIGURE 8).

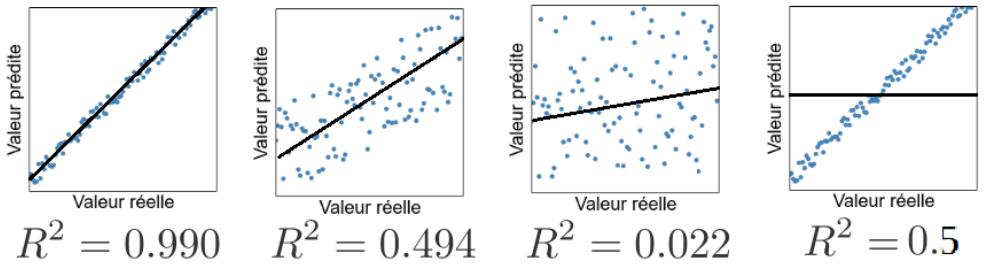


FIGURE 8 : Illustration du coefficient de détermination

II.5 Le sur-apprentissage, qu'est-ce-donc ?

En Intelligence Artificielle, on parle de sur-apprentissage (*overfitting*) quand un modèle a trop bien appris les particularités de chacun des exemples des données d'entraînement. Il présente alors un taux de succès très important sur les données d'entraînement (pouvant atteindre jusqu'à 100%), au détriment de ses performances générales réelles.

On illustre le phénomène de sur-apprentissage sur la FIGURE 9. La ligne noire a vocation à séparer les cercles des triangles. Les données comportent quelques exceptions, qui ne doivent pas être considérées comme des nouvelles règles.

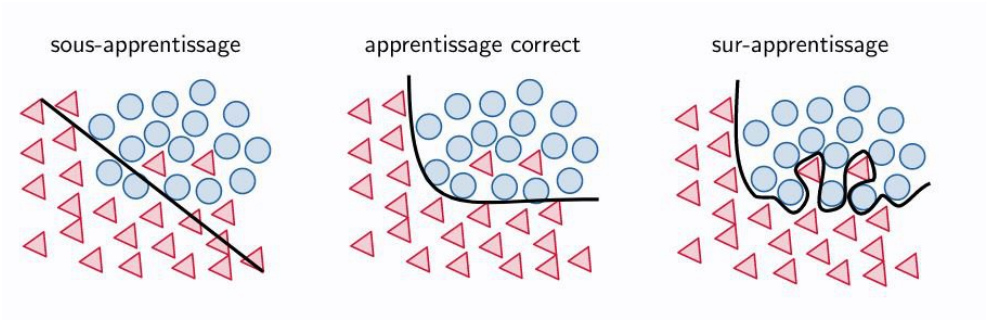


FIGURE 9 : Illustration du sur-apprentissage et du sous-apprentissage

III Algorithme des k plus proches voisins – *kNN*

L'algorithme des k plus proches voisins (*k-Nearest Neighbors* : kNN) est un algorithme qui peut être utilisé pour de la classification ou de la régression. Il est considéré comme étant un algorithme à apprentissage supervisé. C'est effectivement un algorithme utilisant des données étiquetées (il est donc bien supervisé) mais il n'y a en rien une phase d'apprentissage.

C'est certainement un des algorithmes d'Intelligence Artificielle des plus simples à comprendre.

III.1 Un exemple en classification

Dans cet exemple, on considère un jeu de rôle disposant de 2 types (classes) nommées `chevalier` et de `fantassin`. Chaque personnage possède 2 caractéristiques, une note de courage et une note de force. On peut ainsi représenter les données sur la FIGURE 10.

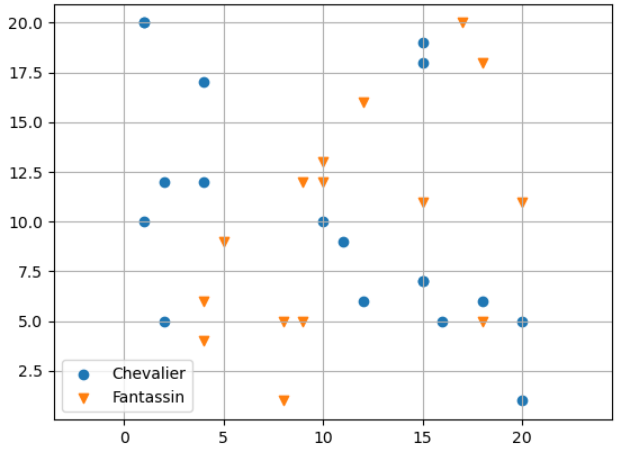


FIGURE 10 : Illustration du jeu de rôle

L'objectif est de classer un nouveau personnage (connaissant ses 2 caractéristiques, notes de courage et de force), soit dans la classe des `chevalier`, soit dans la classe des `fantassin`.

Dans le cas des 7-NN (k=7), on obtient la FIGURE 11.

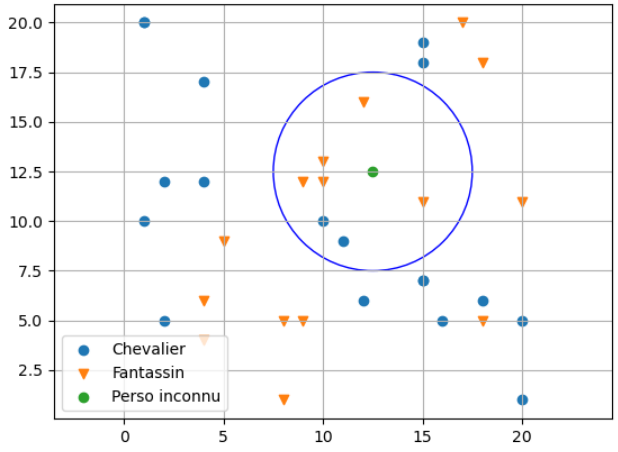


FIGURE 11 : Illustration du jeu de rôle avec nouveau personnage à classer pour k=7

Alors, l'algorithme des k plus proches voisins, pour k=7, renverra la classe `fantassin`, car parmi les 7 plus proches voisins du personnage à classer, 5 sont de la classe `fantassin` et 2 de la classe `chevalier`.

III.2 Algorithmes des k plus proches voisins

On présente sous forme de pseudo-code les deux algorithmes des k plus proches voisins dans le cas de la classification et de la régression.

III.2.1 Algorithmes des k plus proches voisins pour la classification

Données : $\mathcal{D} = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \cdots, (\mathbf{x_p}, y_p)\}$;
$-$ p observations en n dimensions : $\mathbf{x_i} \in \mathbb{R}^n$;
$-$ p étiquettes : y_i ;
Objectif : Déterminer l'étiquette associée à $\mathbf{x}^*$;
pour *i variant de* 1 *à* p **faire**
    Établir le tableau de taille p, des distances entre $\mathbf{x}^*$ et $\mathbf{x_i}$ (généralement euclidienne) ;
fin
Déterminer les k points $\mathbf{x_i}$ les plus proches de $\mathbf{x}^*$;
Par le vote majoritaire parmi les k plus proches voisins, on détermine l'étiquette (notée y^*) de $\mathbf{x}^*$;
return y^*

Algorithme 1 : Algorithme des k plus proches voisins : Classification

III.2.2 Algorithmes des k plus proches voisins pour la régression

Données : $\mathcal{D} = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \cdots, (\mathbf{x_p}, y_p)\}$;
$-$ p observations en n dimensions : $\mathbf{x_i} \in \mathbb{R}^n$;
$-$ p étiquettes : y_i ;
Objectif : Estimer la valeur de $f(\mathbf{x}^*)$;
pour *i variant de* 1 *à* p **faire**
    Établir le tableau de taille p, des distances entre $\mathbf{x}^*$ et $\mathbf{x_i}$ (généralement euclidienne) ;
fin
Déterminer les k points $\mathbf{x_{ki}}$ les plus proches de $\mathbf{x}^*$ avec $\mathbf{x_{ki}}$ le $i^{\text{ème}}$ point le plus proche de $\mathbf{x}^*$ parmi les k plus proches ;
Estimer la moyenne $y = f(\mathbf{x}^*) = \dfrac{1}{\mathbf{k}} \displaystyle\sum_{\mathbf{i}=1}^{\mathbf{k}} \mathbf{y_{ki}}$;
return y

Algorithme 2 : Algorithme des k plus proches voisins : Régression

III.3 Choix de la valeur de k

Le paramètre k est un **hyper-paramètre**. C'est l'utilisateur de l'algorithme qui le fixe. Il n'est pas calculé par l'algorithme mais fixé en avance. Pour chercher la valeur optimale de k pour un problème, il faut tester différentes valeurs de k pour le problème donné ! Cela peut donc prendre du temps. Et le paramètre k optimal trouvé ne sera valable que sur le problème traité !

III.4 Et un exemple de classification en Python avec algorithme k-NN

III.4.1 Présentation de l'exemple

L'algorithme des k plus proches voisins est ici présenté pour la reconnaissance de caractères. On utilise pour cela la base de données `mnist` disponible sous Python.

FIGURE 12 : Échantillon de la base de données `mnist`

III.4.2 Séparation des données d'entraînement et de test

La base de données est composée de 70 000 images de taille 28×28 pixels. On n'utilisera ici que 50% des images, soit 35 000, pour limiter les temps de calculs. Pour réaliser le code des k plus proches voisins, on utilise les fonctions de la bibliothèque `sklearn`, déjà pré-implémentée. On ne va pas à chaque fois réinventer la roue...

```
## Importation de la bibliothèque mnist
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784',version=1) # Cela peut prendre
    du temps...
## Importation d'une fonction permettant de séparer les données
    en deux groupes
from sklearn.model_selection import train_test_split
## Importation du classifieur kNN
from sklearn import neighbors

vect_x_utile,vect_x_non_utile,vect_y_utile,vect_y_non_utile=
    train_test_split(mnist.data, mnist.target,train_size=0.5) #
    50% des données seront utilisées et stockées dans vect_x_utile
     (28x28 pixels) et vect_y_utile le label de l'image
```

La fonction `train_test_split` permet de séparer les données en deux groupes. Les images et leurs étiquettes utiles sont stockées dans les variables `vect_x_utile` et `vect_y_utile`. `vect_x_utile` représente les pixels de l'image. Il s'agit donc d'une matrice de taille 28×28 dont les valeurs sont comprises entre 0 et 255 (images en niveaux de gris). `vect_y_utile` est l'étiquette associée à l'image. Ici, il y a 10 possibilités : 0, 1, 2, $\cdots$, 9.

Pour tester la qualité du classifieur par la méthode des k plus proches voisins, 80% des données sont utilisées comme données d'entraînement. Cela signifie tout simplement que ce sont les données $\mathbf{x_i}$ connues pour la réalisation de notre algorithme des k plus proches voisins. Les 20% des données restantes seront les données de tests ($\mathbf{x}^*$), c'est-à-dire celles dont on va chercher à déterminer l'étiquette. Or de ces données, on connait leur "vraie" étiquette et on pourra comparer les "vraies" étiquettes et les étiquettes prédites et ainsi tester les performances du modèle de prévisions. On a donc au total $35\,000 \times 0.8 = 28\,000$ images connues et $35\,000 \times 0.2 = 7000$ images dont on va chercher à prédire le chiffre écrit.

```
# vect_x_train représente les caractéristiques des données d'
    entraînement
```

```
# vect_x_test représente les caractéristiques des données de test
# etiquettes_train représente les labels (étiquettes) des données d'entraînement
# etiquettes_test représente les labels (étiquettes) des données de test
vect_x_train,vect_x_test,etiquettes_train,etiquettes_test=train_test_split(vect_x_utile,vect_y_utile,train_size=0.8)
```

III.4.3 Classification par les k plus proches voisins sous `sklearn`, pour k fixé

```
# On commence par créer le modèle de classification pour k=3
knn = neighbors.KNeighborsClassifier(3)
# On définit le modèle sur les données d'entraînement
knn.fit(vect_x_train,etiquettes_train)
# On exécute notre modèle de classification pour les données de test
predictions=knn.predict(vect_x_test)

# Détermination des performances de notre classifieur
liste_prediction_correcte=[predictions[i]==etiquettes_test.values[i] for i in range(len(predictions))] # liste
pourcentage_erreur=(1-sum(liste_prediction_correcte)/len(liste_prediction_correcte))*100 # Pourcentage d'erreur dans la prédiction
```

Analyse du code ci-dessus :

- À la ligne 2, on crée le modèle que l'on souhaite utiliser, sous le nom `knn`. Ici, on va utiliser l'algorithme pour $k = 3$. Cette ligne permet d'initialiser l'algorithme ;
- À la ligne 4, l'algorithme ici n'effectue aucune optimisation mais va juste sauvegarder toutes les données en mémoire. C'est sa manière d'apprendre en quelque sorte ;
- À la ligne 6, on effectue les prédictions pour chaque image de `vect_x_test` grâce à la fonction `predict` ;
- À la ligne 9, on construit la liste des prédictions qui sont correctes. On effectue la création de cette liste par compréhension en vérifiant que la prédiction `prediction[i]` est bien égale à la valeur de l'étiquette `etiquettes_test.values[i]`. La liste `liste_prediction_correcte` est donc une liste de booléens `True` et `False` ;
- À la ligne 10, on calcule le pourcentage d'erreur dans les prédictions faites à la ligne précédente. Pour cela, on calcule la moyenne des réussites à l'aide de la commande `sum(liste_prediction_correcte)/len(liste_prediction_correcte)`. On rappelle que `True+True=2`, le typage est automatiquement forcé, c'est pour cela que l'on peut utiliser la fonction `sum`.

Le résultat obtenu est un pourcentage d'erreur de l'ordre de 3.6%.

IV Algorithme des k moyennes – *k-means*

IV.1 Objectif

Le but est de partitionner un ensemble de données d'entraînement (non étiquetées, donc l'apprentissage est non supervisé) en k groupes (classes) regroupant les données les plus « proches » ou « similaires ».

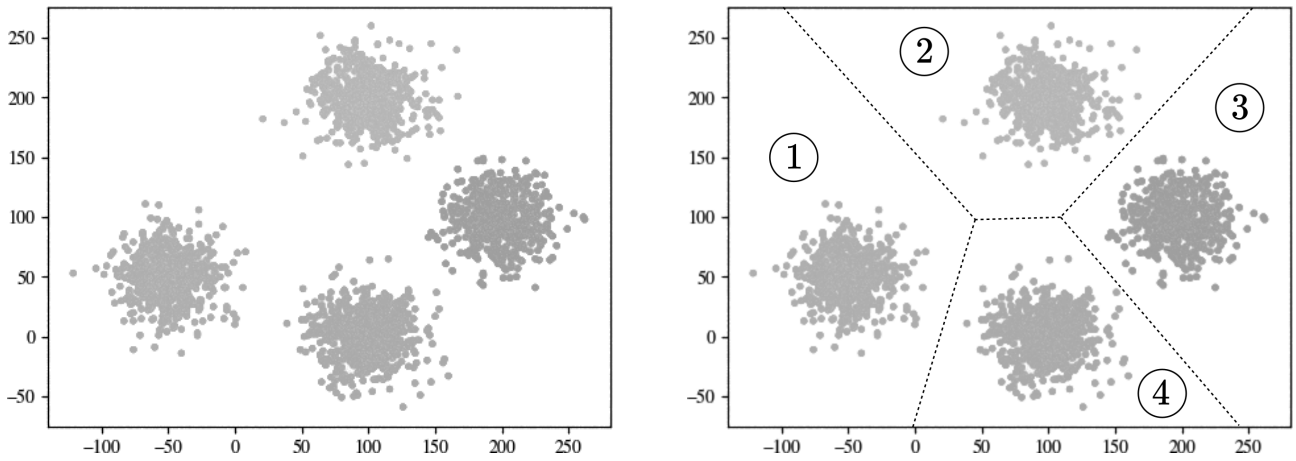


FIGURE 13 : Exemple de données à partitionner, où il est évident que 4 groupes distincts se détachent. Un partitionnement « manuel » peut être celui donné à droite.

IV.2 Principe de l'algorithme des k moyennes

Le principe est basé sur la représentation de chaque classe par une donnée centrale, appelée **centroïde** (ou centre de masse), qui est tout simplement le point minimisant la somme des distances de chaque donnée de la classe à ce point : lorsque ce point est atteint, il est alors équivalent au barycentre de ces données (si la distance est la norme euclidienne). La recherche de ce point, pour chaque classe parmi k, est déterminé de manière itérative en effectuant la moyenne des données appartenant à la classe, d'où le nom d'algorithme des k-moyennes.

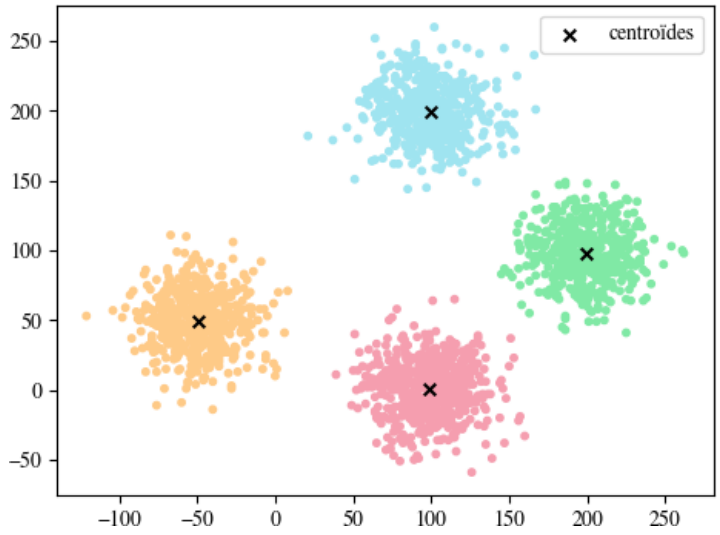


FIGURE 14 : Centroïdes représentatifs des 4 classes à discriminer.

IV.3 Algorithme des k moyennes

```
1  Choix du paramètre k;
2  Initialisation des positions des k centroïdes (aléatoirement, donc k éléments tirés au
     hasard) ;
3  Processus ;
4  tant que Position des centroïdes est variable faire
5  |  pour chaque point du jeu de données faire
6  |  |  Déterminer quel est le centroïde le plus proche (à l'aide de la distance
   |  |    euclidienne). On constitue alors un ensemble de points attachés à chaque
   |  |    centroïde : les clusters.;
7  |  fin
8  |  pour chaque centroïde faire
9  |  |  Déplacer chaque centroïde au barycentre des points qui lui sont attachés. ;
10 |  fin
11 fin
```

Algorithme 3 : Algorithme des k-moyennes

IV.4 Choix du paramètre k

k est un hyper-paramètre que l'on va choisir pour identifier un nombre de classes voulu. C'est donc un hyper-paramètre choisi *a priori*. Il a une importance fondamentale dans la classification (voir FIGURE 15).

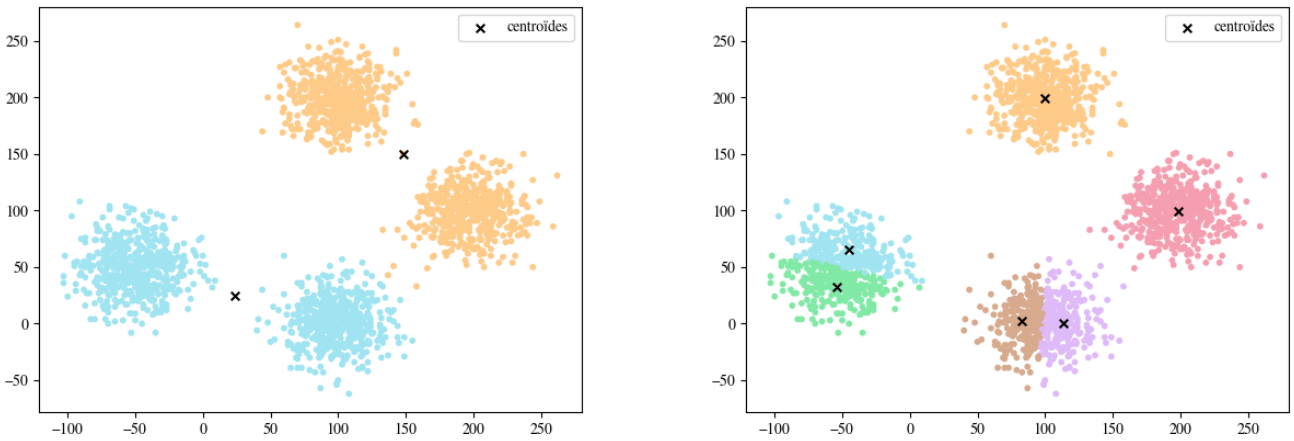


FIGURE 15 : Centroïdes obtenus dans le cas où k=2 et k=6.

IV.5 Et un exemple d'application en Python avec l'algorithme k-moyennes

IV.5.1 Présentation de l'exemple

Le jeu de données Iris a été initialement publié à l'UCI Machine Learning Repository. Ce jeu de données date de 1936 et est souvent utilisé pour tester des algorithmes d'apprentissage automatique et des visualisations.

Il contient trois classes (familles) de la fleur Iris (`Iris-Setosa`, `Iris-Versicolor` et `Iris-Virginica`). Il contient 150 données d'entraînement (ligne du jeu de donnée), 50 pour chaque classe. Chaque donnée est composée de quatre attributs pour décrire une fleur d'Iris. Ces attributs sont `Sepal_Length`, `Sepal_width`, `Petal_Length` et `Petal_width`.

Iris Versicolor Iris Setosa Iris Virginica

Bien que le jeu de données soit déjà étiqueté par le type d'Iris, nous supposerons que nous ne disposons pas de cette information. L'objectif de cet exemple par l'algorithme des k-*means*, est justement de trouver des regroupements de données cohérentes pour former les k=3 classes ! ! !

IV.5.2 Chargement de la base de données

La base de données est déjà incluse dans Python, dans le module `sklearn.datasets`.

```python
# Importation des modules
from sklearn.cluster import KMeans
from sklearn import datasets

# Chargement de base de données iris
iris=datasets.load_iris()
data=iris.data # data contient les caractéristiques
```

IV.5.3 Apprentissage par l'algorithme des k-*means* pour k=3 classes

```python
# Création du modèle de classification pour 3 classes
modele=KMeans(3)
# Lancement de l'apprentissage
modele.fit(data)
# Affichage des données classées
classes_predites=modele.labels_
print(classes_predites)
```

L'exécution de code permet d'obtenir l'affichage suivant : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 ***2*** 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ***2*** 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 ***1*** 2 2 2 2 ***1*** 2 2 2 2 2 2 ***1*** ***1*** 2 2 2 2 ***1*** 2 ***1*** 2 ***1*** 2 2 ***1*** ***1*** 2 2 2 2 2 ***1*** 2 2 2 2 ***1*** 2 2 2 ***1*** 2 2 2 ***1*** 2 2 ***1***].

Remarque : Les numéros des classes obtenues peuvent varier en fonction de l'exécution du code précédent. En effet, compte-tenue de l'initialisation aléatoire des k centroïdes, la première donnée peut être affectée à la classe 0, 1 ou 2.

Remarque : La base de données étant organisée par groupe de 50 données, les 50 premières données correspondent à la classe `Iris-setosa`, les 50 suivantes à la classe `Iris-versicolor` et les 50 dernières à la classe `Iris-virginica`.

IV.5.4 « Validation » de la classification

Comme annoncé en présentation de cet exemple, nous disposons des classes d'appartenance réelles pour chaque donnée de la base de données. Ainsi, nous pouvons vérifier les performances de notre classifieur. Dans un cas général, bien évidemment, cette vérification n'est pas possible.

Les « erreurs de prédiction » sont identifiées dans le paragraphe 4.5.3 ***en gras, rouge et en italique***. On en dénombre, manuellement, 15 sur 150. Le résultat obtenu, pour cet essai, est un pourcentage d'erreur de l'ordre de 10%. Globalement satisfaisant. . ., mais nous n'avions que 50 données pour chaque classe d'Iris. . .

V Les réseaux de neurones

V.1 Une inspiration biologique

On sait depuis longtemps que le cerveau humain est constitué d'environs 10^{11} neurones, inter-connectés par des synapses (environs 10^4 par neurone). La modélisation du cerveau passe donc par la modélisation des neurones et de leurs synapses.

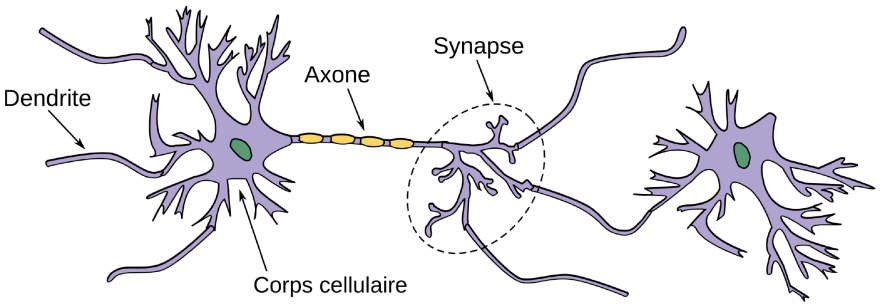


FIGURE 16 : Interconnexions de neurones biologiques

Le modèle biologique le plus simple est celui du neurone à impulsion, où un neurone génère un potentiel électrique (appelé potentiel d'action) selon les stimuli d'entrée, qui circule le long de l'axone jusqu'aux synapses pour être transmis à de nombreux autres neurones via des dendrites.

V.2 Neurone artificiel - Le perceptron

Le **neurone artificiel**, appelé **perceptron**, est l'élément de base des réseaux de neurones. Il prend en entrées les grandeurs x_i pour $i \in [\![1, n]\!]$ et génère en sortie la grandeur y, et est décrit sur la FIGURE 17.

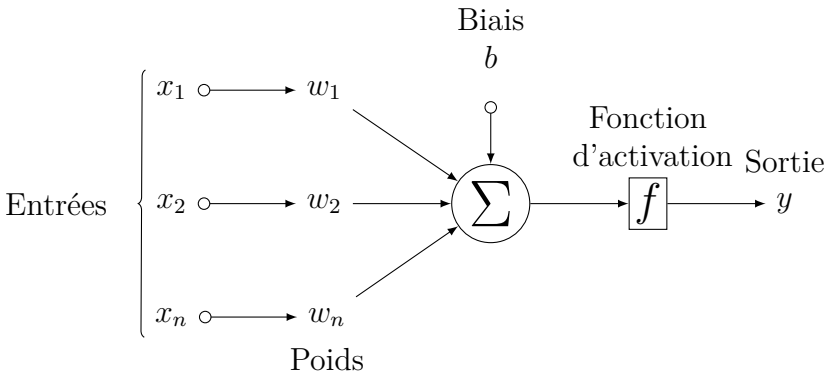


FIGURE 17 : Modélisation d'un neurone artificiel

Ainsi, la sortie y d'un neurone artificiel est telle que :

$$y = f\Big(\sum_{i=1}^{n} w_i \cdot x_i + b\Big)$$

Le coefficient w_i est appelé **poids** associé à l'entrée x_i, et b le **biais**. La fonction f, généralement non linéaire, est appelée **fonction d'activation**.

V.2.1 Objectif de l'IA avec un neurone artificiel

Dans le cadre de la classification ou de la régression, l'objectif de l'algorithme est de déterminer les poids w_i et le biais b du neurone artificiel, pour une fonction d'activation imposée, afin d'optimiser une métrique calculée sur les données d'entraînement.

V.2.2 Les fonctions d'activation usuelles

La fonction d'activation sert avant tout à modifier de manière non-linéaire les données d'entrée. C'est au concepteur du modèle de prévision (classification ou régression) de faire le choix de la fonction d'activation.

Fonction ReLU La fonction ReLU *Rectified Linear Unit* est la fonction d'activation la plus simple et la plus utilisée. Pour une valeur θ, elle renvoie θ si est supérieur à 0, 0 sinon. Autrement dit, c'est le maximum entre θ et 0.

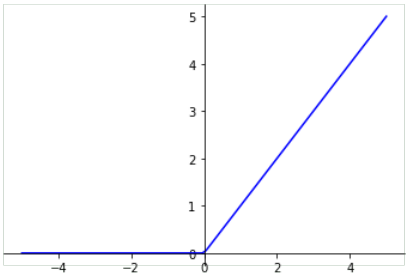


FIGURE 18 : Fonction d'activation ReLU

Fonction Sigmoïde La fonction Sigmoïde donne une valeur comprise entre 0 et 1, autrement dit, comme une probabilité. Si θ est la grandeur d'entrée de la fonction d'activation, et y sa sortie, elle est définie par :

$$y = \frac{1}{1 + e^{-\theta}}$$

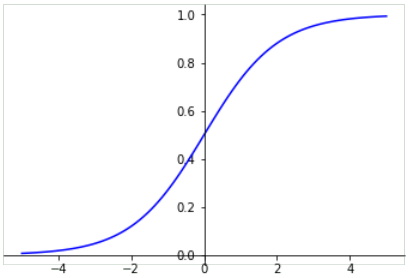


FIGURE 19 : Fonction d'activation sigmoide

Elle est donc très utilisée pour la classification binaire – 2 classes (associée à une fonction *argmax*), lorsqu'un modèle de classification doit déterminer seulement deux labels.

Fonction Tangente hyperbolique La fonction `tanh` est simplement la fonction de la tangente hyperbolique. Il s'agit en fait d'une version mathématiquement décalée de la fonction sigmoïde, notamment. En effet, la sigmoïde donne un résultat entre 0 et 1, alors que la fonction `tanh` donne un résultat entre -1 et 1.

Cette fonction d'activation est, comme la sigmoïde, utilisée dans la classification binaire.

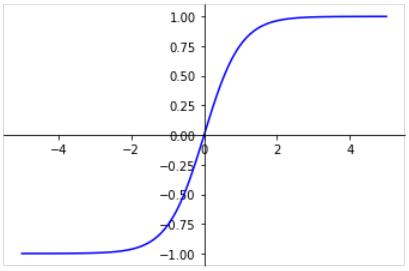


FIGURE 20 : Fonction d'activation tangente hyperbolique

V.2.3 Choix de la fonction d'activation

À ce jour, aucune théorie générale ne permet de procéder au choix le plus pertinent d'une fonction d'activation pour un problème donné. Ainsi, de nombreux tests sont mis en œuvre, et le meilleur est retenu... On nous avait vendu de la science...

V.3 Les réseaux de neurones

Les réseaux de neurones (FIGURE 21) sont une association organisée de neurones artificiels. Ainsi, les entrées de certains neurones sont les sorties d'autres...

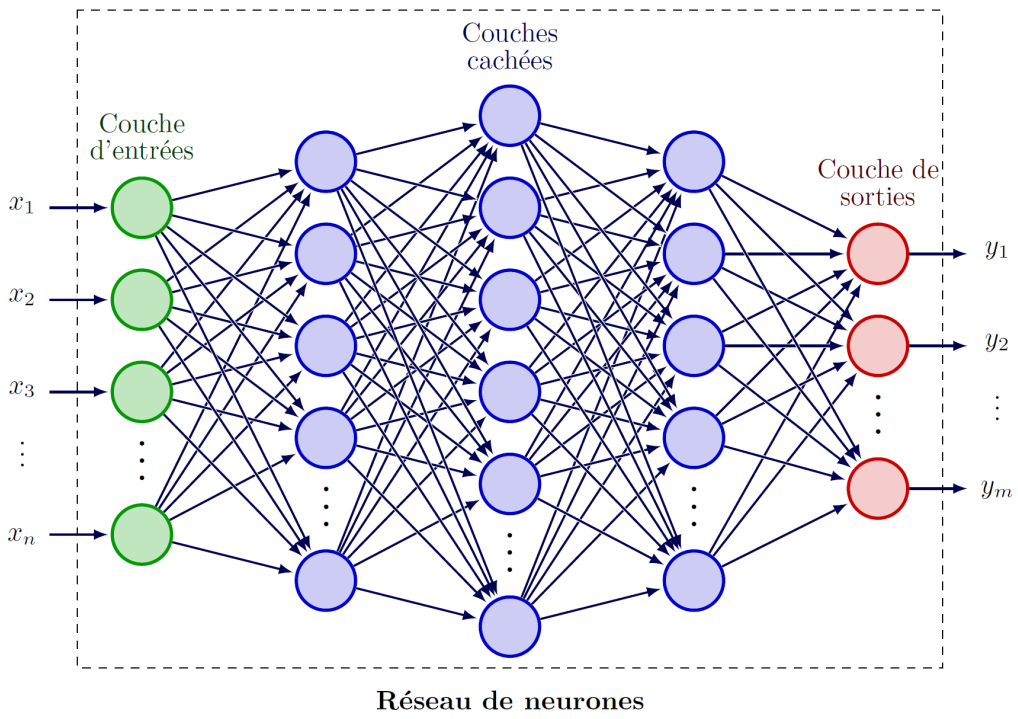


FIGURE 21 : Réseau de neurones

V.3.1 Structures des réseaux de neurones

Un réseau de neurones peut-être décomposé graphiquement en 5 zones (FIGURE 21) :

- la zone des entrées (x_i). Elle correspond au nombre de caractéristiques. Dans le cas de la reconnaissance d'objet dans une image, elle correspond à chaque pixel de l'image ;
- la **couche d'entrée**. Il s'agit de la première couche qui traite directement les entrées. Cette couche prend en entrée une seule caractéristique (x_i), et utilise la fonction d'activation Identité ($y = f(x) = x$) ;
- la **couche de sortie**. Il s'agit de la dernière couche du réseau de neurones. C'est elle qui génère les sorties y_i ;

— la zone des sorties (y_i). Ce sont les grandeurs prédites par le modèle ;
— toutes les autres couches (si elles existent) sont appelées **couches cachées**.

Chaque neurone (dans les couches cachées ou de sorties) possède ses propres entrées, et donc ses propres poids et biais, et sa propre fonction d'activation.

Remarque : Le nombre de neurones par couches cachées n'est pas obligatoirement le même.

V.3.2 Objectif de l'IA par réseau de neurones

L'objectif de l'IA est de déterminer les poids et les biais associés à chaque neurone artificiel, afin de répondre de manière optimale à l'objectif de classification ou de régression. Nous ne rentrerons pas dans le détail du processus algorithmique permettant la détermination de ces poids et biais, car cela dépasse largement le cadre de ce cours.

Bien évidemment, cette optimisation dépend du choix des fonctions d'activation de chaque neurone, et là encore une fois, il n'y a pas de règles simples... La seule solution, tester, tester, tester et encore tester... et prendre le meilleur !!!

TD 1 – Télescope constitué de deux satellites[1]

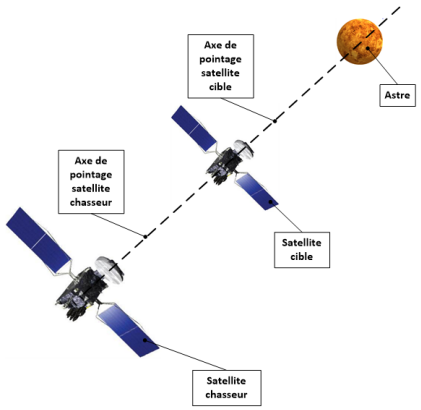


FIGURE 1 : Pointage d'un astre par le couple de satellites

Fiabilisation de la mesure

Lors de la phase d'intégration, des défaillances du capteur permettant de détecter la position linéaire du chariot ont perturbé la phase de test. Afin de résoudre ce problème, le choix a été fait de doubler ce capteur. Un module doit être programmé afin d'accroître la fiabilité de la mesure. Le bilan des entrées et des sorties de ce module est représenté sur la Figure 2.

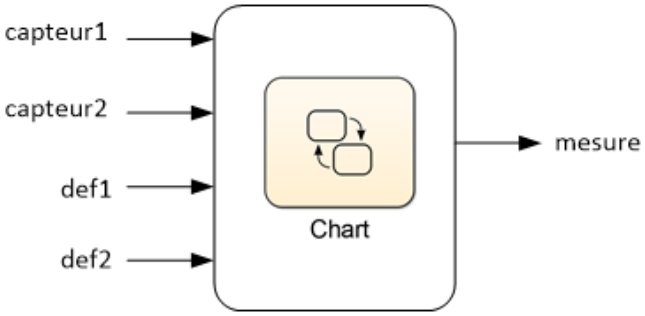


FIGURE 2 : Bilan des entrées et des sorties du système de fiabilisation de la mesure

La nécessité d'implanter ce graphe d'états dans une cible impose de prendre en compte le pas de calcul choisi pour la programmation.

— Le logiciel met à jour les variables du graphe d'état à chaque pas de calcul, réglé ici à 0.01 s ;

— Une mesure est effectuée tous les centièmes de seconde.

La Figure 3 présente la programmation de cette logique à l'aide d'une machine à états.

Éléments de syntaxe :

entry / var=0 : l'affectation var=0 s'effectue à l'activation de l'état ;

do / var=var+1 : l'incrémentation de var s'effectue à chaque pas de calcul, tant que l'état est actif.

1. Extrait X-ENS - PSI - 2015

Entrées	
capteur1	Signal de mesure issu du **capteur1**
capteur2	Signal de mesure issu du **capteur2**
def1	Signal logique indiquant que le capteur1 est en défaut. Si **def1**=1 le signal présente un défaut. Si **def1**=0 le signal est correct
def2	Signal logique indiquant que le capteur2 est en défaut. Si **def2**=1 le signal présente un défaut. Si **def2**=0 le signal est correct
Sortie	
mesure	Signal de mesure calculé en vue de l'exploitation par le processus

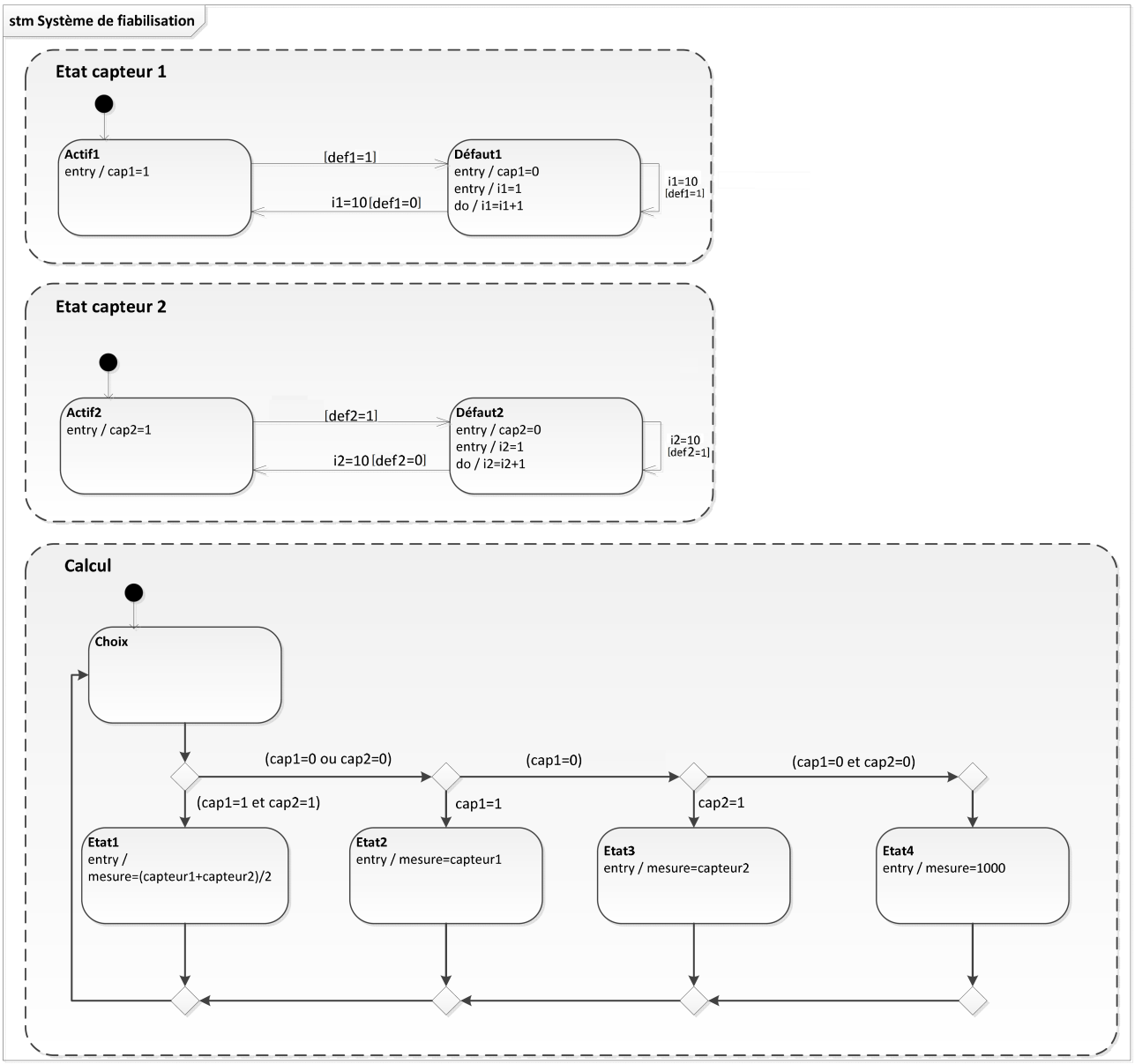


FIGURE 3 : Machine à états du système de fiabilisation

Q1. Que représentent les valeurs 0 et 1 que peuvent prendre les variables internes **cap1** et **cap2** ?

Q2. Préciser le rôle des variables internes **i1** et **i2** ?

Q3. Donner l'expression de la variable **mesure** dans les cas suivants :
- Si les deux capteurs ne sont pas en défaut ;
- Si le capteur 1 présente un défaut ;
- Si le capteur 2 présente un défaut ;
- Si les deux capteurs sont en défaut.

Q4. Quelle durée s'écoule entre l'instant où **def1** passe de la valeur 1 à la valeur 0 et l'instant où la mesure issue du capteur 1 est exploitée ?

TD 2 – Canne motorisée[1]

Étude de l'exigence 3.1.6 « Commande des axes asservis »

Cette partie a pour objectif d'analyser le mode de distinction des différentes phases de fonctionnement de la canne robotisée. Lors de la marche avec une canne conventionnelle, il est possible de constater une synchronisation du mouvement de la canne conventionnelle avec celui de la jambe qu'elle assiste. Ainsi, une forte corrélation est observée entre l'angle de la canne et celui de la hanche de la jambe invalide.

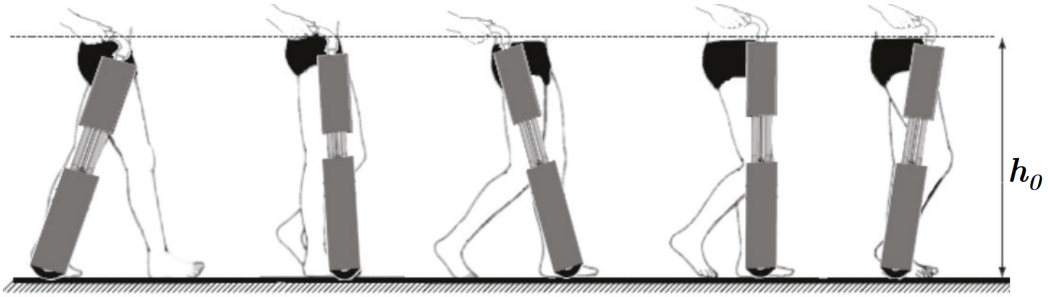


FIGURE 1 : Synchronisation souhaitée du prototype de canne robotisée avec la marche

Selon la figure 1, le mode de commande suivant a été retenu pour contrôler le mouvement du prototype de canne robotisée.

- Lors de la phase de balancement de la jambe invalide, l'angle de la canne active par rapport à la verticale est asservi sur l'angle de la hanche de la jambe invalide. Cette tâche est accomplie en gardant la hauteur de la poignée h_0 constante afin de ne pas perturber la position de la main de l'utilisateur ;
- Lors de la phase d'appui de la jambe invalide, la roue est asservie à une vitesse nulle afin d'offrir un point d'appui immobile pour le patient. La longueur de l'axe télescopique est asservie pour garder la hauteur de la poignée h_0 constante de la canne.

Il est donc nécessaire de maintenir la hauteur de poignée h_0 constante pour les deux phases. Ceci impose une relation entre l'inclinaison de la canne et sa longueur.

La détection des phases d'appui et de balancement de la jambe invalide est basée sur une mesure du gyromètre inclus dans la centrale inertielle attachée à la jambe invalide. Lorsque la jambe est en balancement, la rotation de la cuisse s'effectue dans le sens trigonométrique, la rotation a lieu en sens inverse lors de la phase d'appui.

La figure 2 donne l'évolution de la vitesse angulaire fournie par le gyromètre (facteur $\times$ 100) et l'évolution de la variable S identifiant les phases de balancement ($S = 0$) et d'appui ($S = 1$) au cours d'un cycle de marche. Pour éviter la prise en compte des bruits de mesure du gyromètre dans le changements d'état de S, on introduit une valeur réglable de détection appelée *Seuil*.

La stratégie retenue pour la commande des deux degrés de liberté du prototype de canne au cours d'un cycle de marche peut être décrite par le diagramme d'états partiel fourni ci-dessous. Le tableau 1 récapitule les variables, états et activités associées, utilisés dans ce diagramme.

1. Extrait CCINP - PSI - 2018

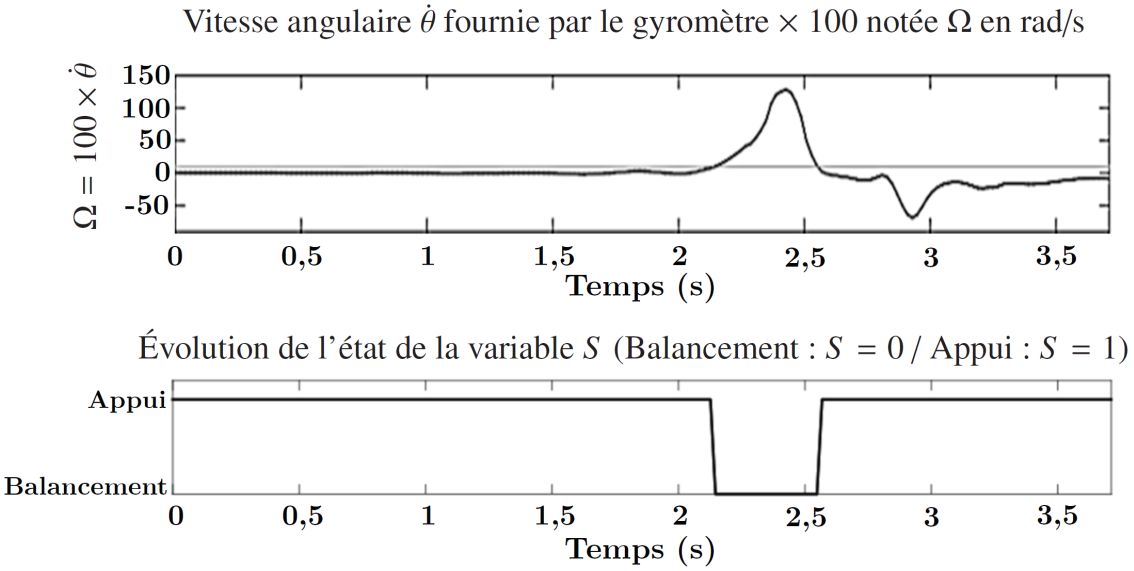
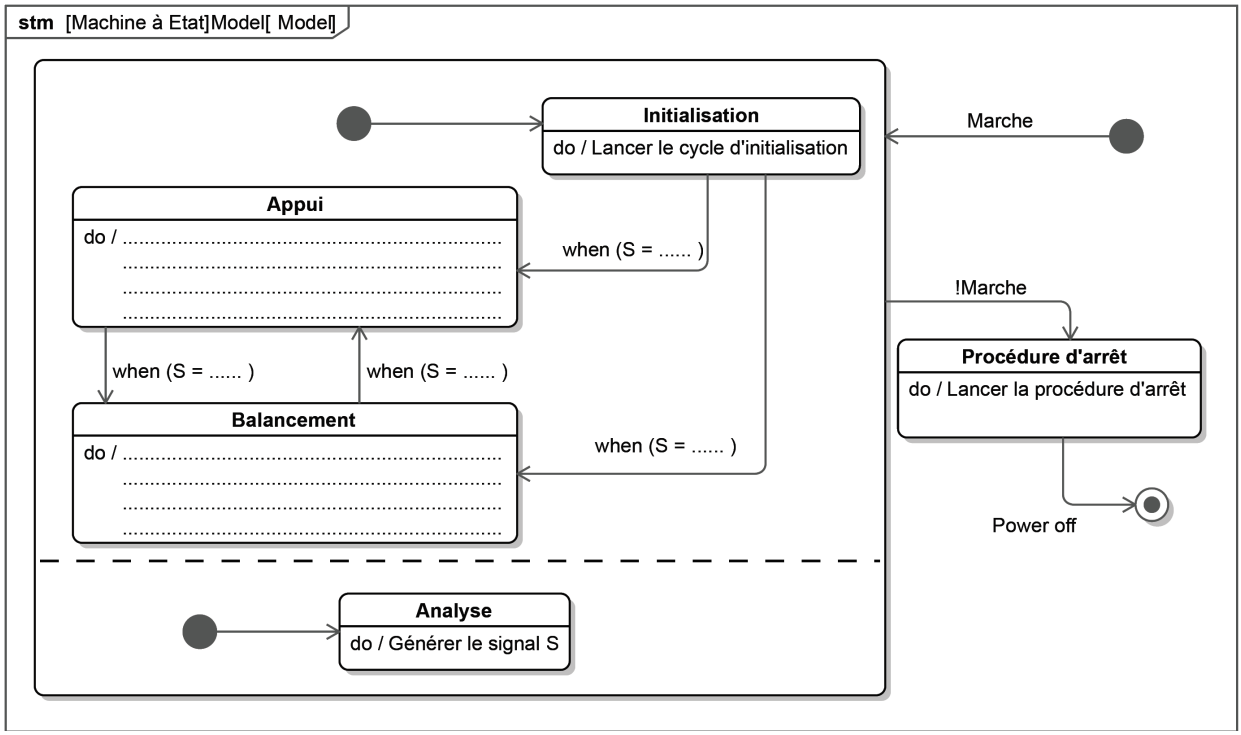


FIGURE 2 : Évolution de la vitesse angulaire fournie par le gyromètre (facteur × 100) et évolution de la variable S au cours d'un cycle de marche

Variables	États	Activités associées
• S : balancement $S = 0$ appui $S = 1$ • *Seuil* : valeur réglable de détection • *Marche* : mise en route du système • *Power off* : arrêt de l'alimentation	• Initialisation • Appui • Balancement • Analyse • Procédure d'arrêt	• Lancer le cycle d'initialisation • Asservir la hauteur de la canne • Maintenir une vitesse de roue nulle • Asservir la vitesse de roue • Générer le signal S • Lancer la procédure d'arrêt

TABLEAU 1: Variables, états et activités du diagramme d'états de la commande du prototype de canne

Q1. Compléter le diagramme d'états fourni en page suivante afin de gérer les asservissements en phase de balancement et d'appui. Pour cela, préciser les valeurs (0 ou 1) de la variable S et les activités associées aux états des phases d'appui et de balancement.


stm [Machine à Etat]Model[Model]
Initialisation
do / Lancer le cycle d'initialisation
Marche
Appui
do / .....
.....
.....
.....
when (S = ...)
!Marche
Procédure d'arrêt
do / Lancer la procédure d'arrêt
when (S = ...)
when (S = ...)
Balancement
do / .....
.....
.....
.....
when (S = ...)
Power off
Analyse
do / Générer le signal S

TD 3 – Quille pendulaire[1]

Gérer les cycles préprogrammés

La commande des manœuvres de la quille s'effectue via un pupitre (voir figure 1) placé à proximité du poste de barre à partir duquel le navigateur peut demander à l'automate de réaliser :

- Le déplacement de la quille d'un bord ou de l'autre selon une valeur de consigne ;
- Des cycles préprogrammés comme celui de « virement de bord » et celui de « Relâcher ». Le cycle de virement de bord permet de placer la quille de façon symétrique à la position qu'elle occupait précédemment. Ce cycle est utilisé lorsque le navigateur change l'orientation du navire par rapport au vent lors d'un virement de bord. L'automate prend alors en charge intégralement la séquence de manœuvres de la quille, laissant le navigateur disponible pour les autres tâches.

Le cycle « Relâcher » permet de déplacer la quille sous le seul effet de la pesanteur. La quille est ainsi manœuvrée sans utiliser l'énergie de la centrale hydraulique. L'automate est également interfacé via le réseau du navire à une base de données où sont stockés les paramètres des navigations précédentes (conditions météorologiques, performances du navire et angle de quille). Le navigateur peut ainsi intégrer les paramètres de la quille à l'ensemble des paramètres décisionnels qui lui permettent d'élaborer sa stratégie de navigation. L'automate gère également la centrale hydraulique qui met en pression l'huile utilisée dans les vérins de manœuvre de la quille.

Deux capteurs renseignent l'automate :

- Un inclinomètre mesure l'angle de gîte du navire, information notée α ;
- Le capteur de position angulaire mesure l'angle d'inclinaison de la quille, grandeur notée $mes(\theta)$.

Le pupitre de commande est doté de quatre boutons poussoirs :

- tr – Demande d'inclinaison sur tribord ;
- b – Demande d'inclinaison sur bâbord.

Un appui « bref » sur l'un ou l'autre de ces boutons provoque une évolution de l'angle de consigne de 1°, un appui « long » une évolution de 10°.

- vir – Demande du cycle « virement de bord » ;
- rel – Demande du cycle « Relâcher ».

1. Extrait CCMP - PSI - 2014

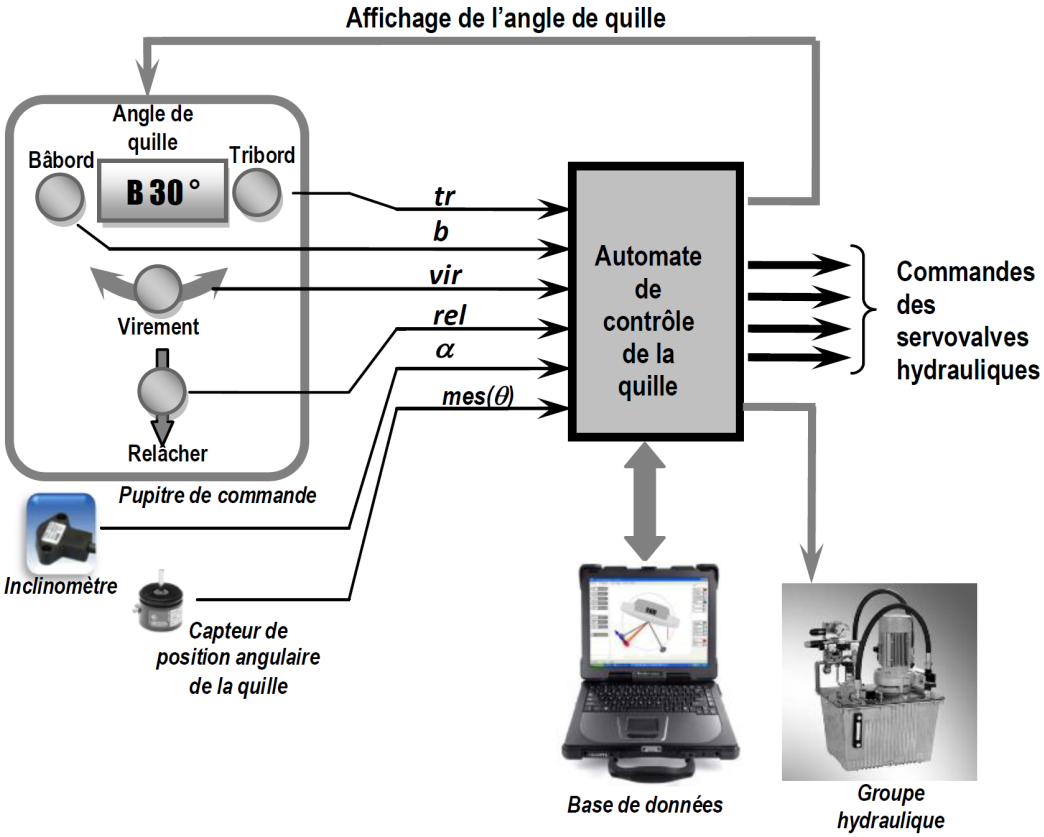


FIGURE 1 : Structure de la commande

Il comporte également un afficheur numérique permettant de visualiser soit l'angle d'inclinaison de la quille, soit la valeur de consigne lorsque le barreur agit sur « bâbord » ou « tribord » (variables b ou tr). Le modèle de commande implanté dans l'automate est défini par le diagramme d'états sur la figure 5.

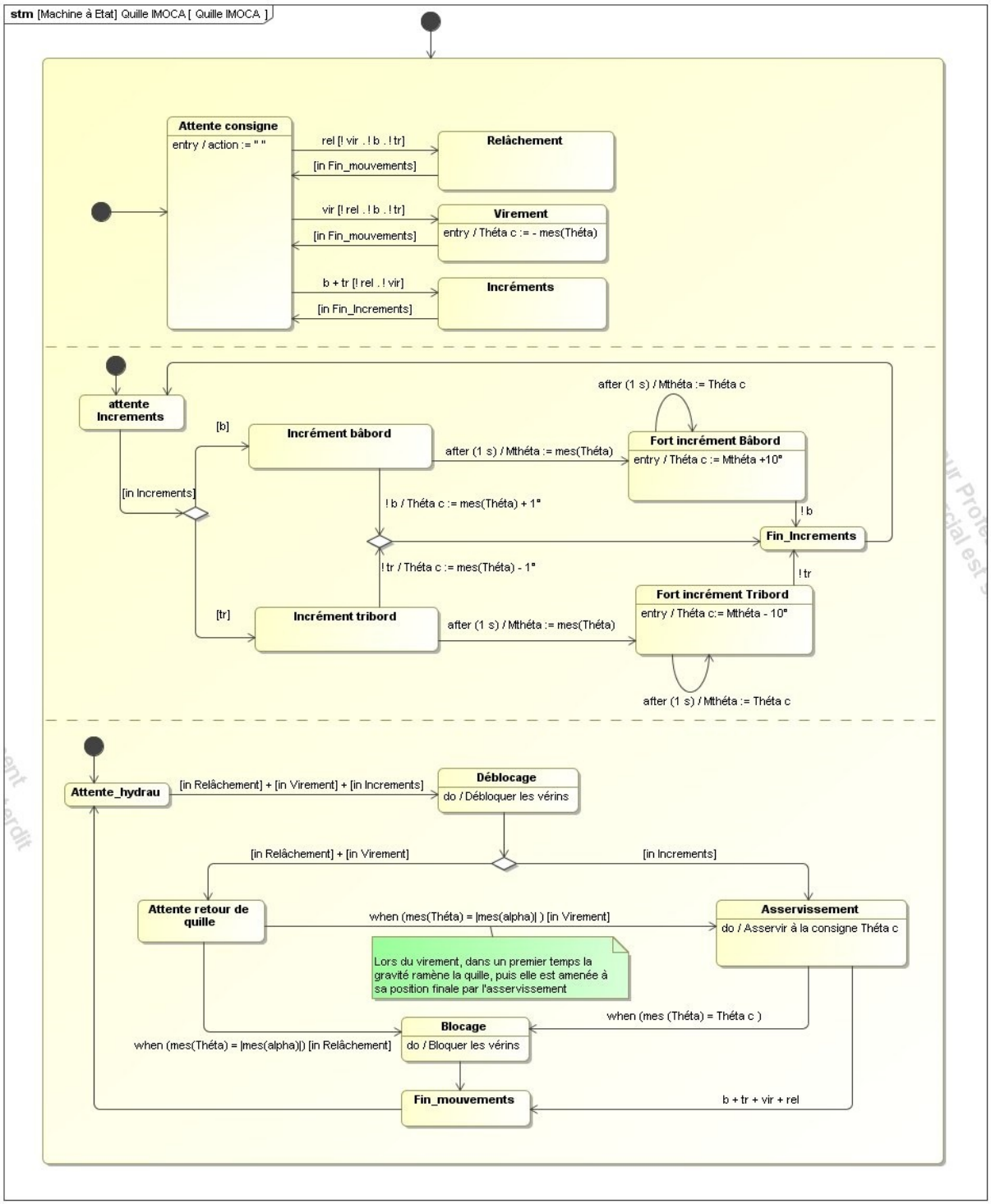


FIGURE 2 : Diagramme d'états décrivant le modèle de commande implanté dans l'automate

Q1. À l'instant t_0, la situation du graphe de commande est telle que les états Attente (Attente consigne, attente Incréments et Attente_hydrau) soient actifs et la quille est alors inclinée de $+20°$. Le navigateur donne une série d'impulsions sur b et tr conformément au chronogramme donné sur la figure 3. En analysant le modèle de commande, compléter le graphe des évolutions temporelles de la consigne angulaire θ_C donné sur le document réponse jusqu'à l'instant t_3 et donner la valeur obtenue pour θ_C, et ce, sans se préoccuper de la façon dont la partie opérative réagit à cette consigne.

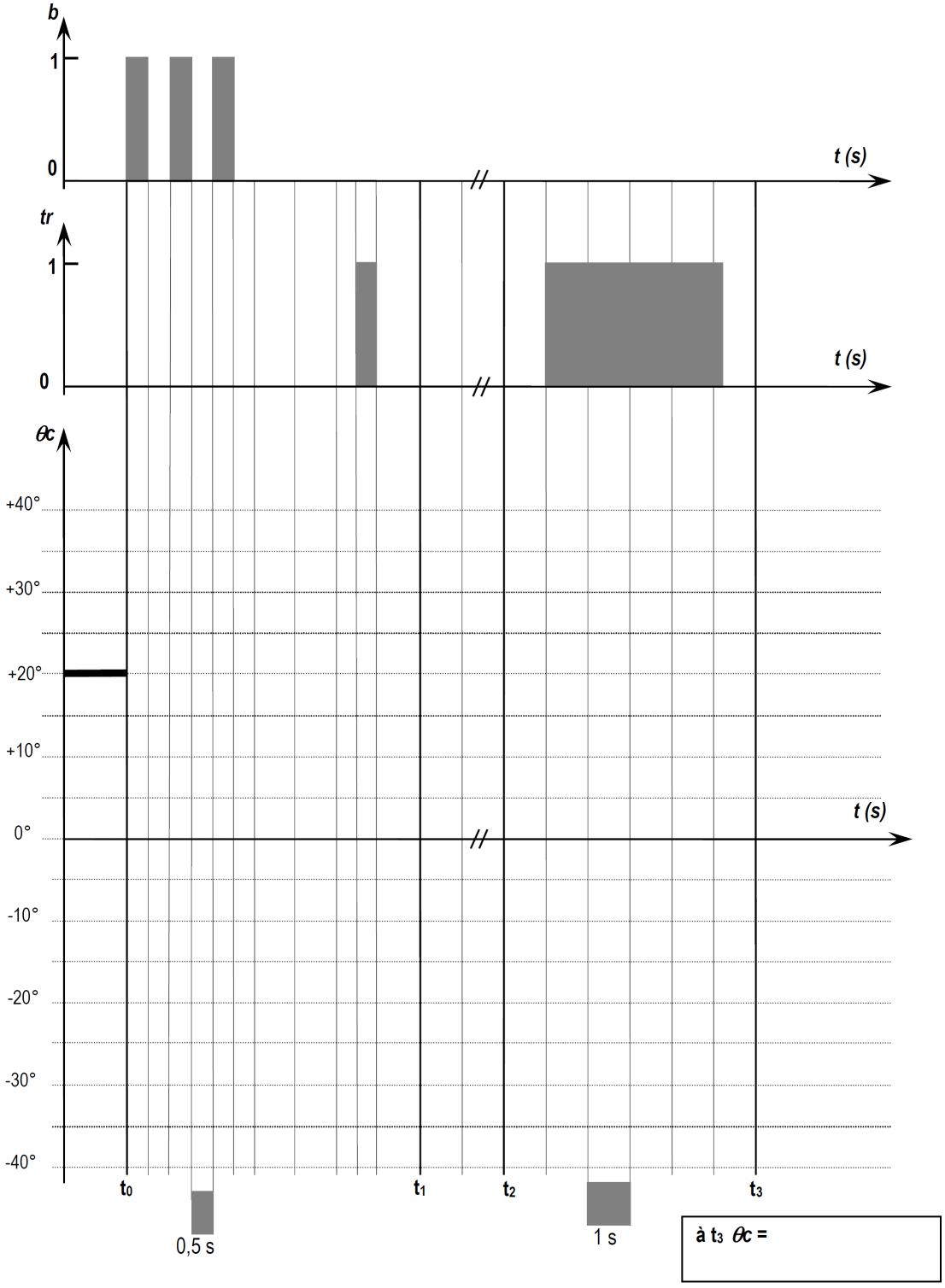
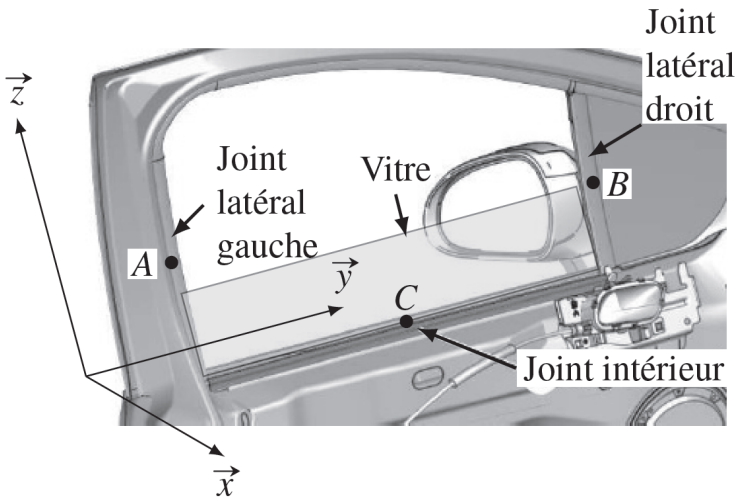


FIGURE 3 : Série d'impulsions sur b et tr imposée par le navigateur

Q2. La situation de départ est telle que les états Attente (Attente consigne, attente Incréments et Attente_hydrau) soient actifs et la quille est alors inclinée de +40°. Le navigateur donne la consigne de virement de bord, *vir*.

- Donner la liste des états successivement actifs présentées par le modèle de commande jusqu'au retour dans la situation identique à celle de départ. La situation initiale est notée (Attente consigne, attente Incréments, Attente_hydrau) ;
- En considérant que la chaîne de commande de la quille est précise, donner la valeur angulaire que représente $mes(\theta)$ en fin de ce cycle.

TD 4 – Commande générique d'ouvrants pilotés d'automobiles [1]



Les constructeurs automobiles sont sans cesse dans l'obligation d'innover pour rester attractifs vis-à-vis du client. Les ouvrants pilotés automobiles font partie des atouts différenciateurs. Le terme ouvrant désigne à la fois les lève-vitres électriques, les toits ouvrants, les toits escamotables, les coffres motorisés et les portes latérales coulissantes. Tous ces ouvrants sont une source d'attrait pour le client, de par leur praticité ou encore par leurs facteurs de différenciation importants.

FIGURE 1 : Différents types d'ouvrants du groupe PSA

Il existe deux types de pilotage des ouvrants :

— le premier est un système classique et/ou d'assistance. L'utilisateur gère complètement le déplacement de l'ouvrant. Dès qu'il arrête son action sur la commande, l'ouvrant s'immobilise, c'est le cas par exemple du lève-vitre électrique non séquentiel. Ainsi, avec un système classique et/ou d'assistance, le déplacement de l'ouvrant est entièrement imputable aux actions de l'utilisateur ;

— le second type est le pilotage automatisé des ouvrants. Ici, l'utilisateur demande simplement à ce que l'ouvrant se déplace jusqu'à une position prédéfinie. Une brève action de sa part entraîne le déplacement complet de l'ouvrant. Pour le lève-vitre électrique séquentiel, l'utilisateur demande à ce que la vitre remonte complètement, par une courte

1. Extrait CCINP - PSI - 2017

action sur l'interrupteur. Dès lors, le système de contrôle/commande gère le déplacement de l'ouvrant dans le cas normal, mais aussi en cas de dysfonctionnement (perte de fonctionnalité ou présence d'un obstacle sur le trajet de la vitre). Il faut donc assurer un fonctionnement sûr et robuste du système d'ouvrant piloté automatisé pour éviter que le système blesse un occupant.

Le diagramme des exigences de la figure 3 liste quelques performances attendues pour le lève-vitre électrique.

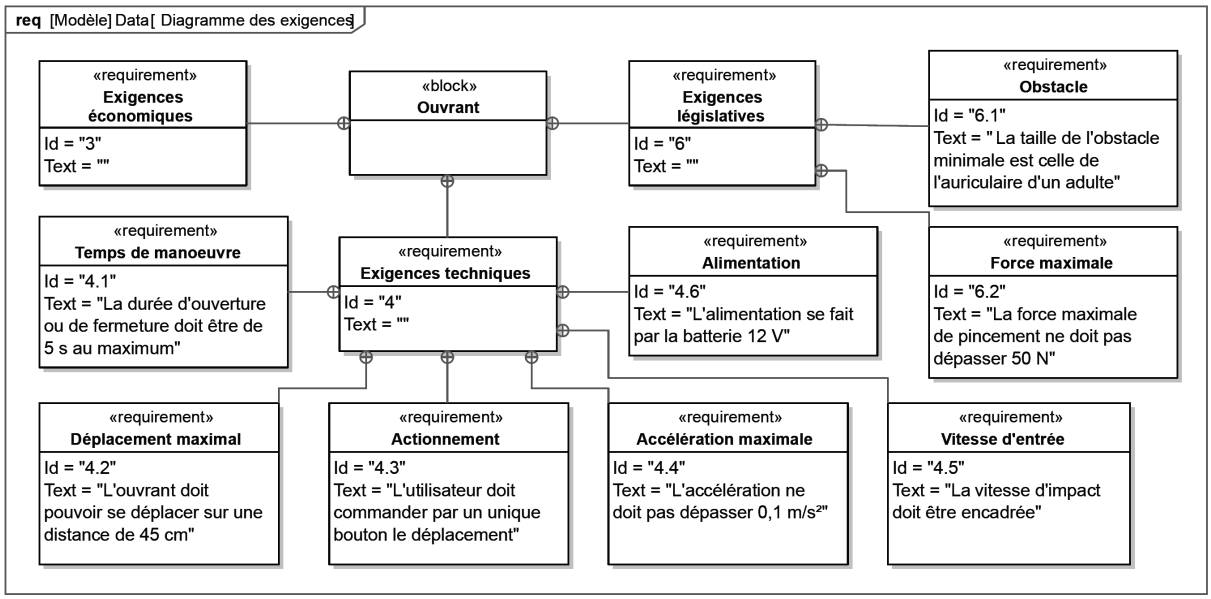


FIGURE 2 : Diagramme des exigences

Modélisation multiphysique du système

Un modèle multiphysique doit être mis en place pour pouvoir prendre en compte tous les phénomènes qui apparaissent lors du fonctionnement de la vitre sans et avec obstacle.

Le schéma-blocs est donné sur la figure 3. Les différents éléments intervenant dans ce modèle doivent être caractérisés séparément pour obtenir une représentation la plus fidèle possible de la réalité.

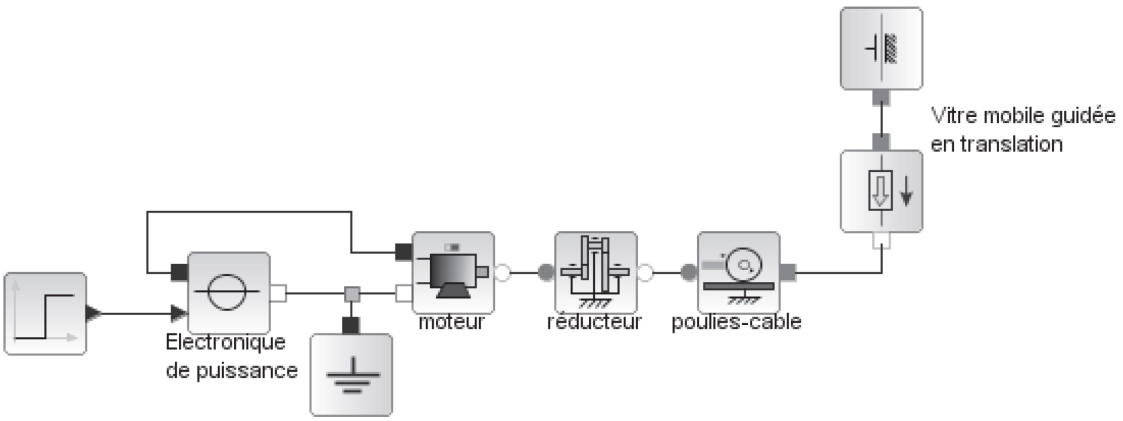


FIGURE 3 : Schéma-blocs du lève-vitre sans obstacle

Modélisation des efforts de frottement

D'un point de vue des actions mécaniques, les joints jouent fortement sur le comportement de la motorisation au cours du temps. C'est pourquoi il est important d'évaluer l'impact des frottements entre les joints et la vitre sur le comportement du système. Les joints appliquent une action **de part et d'autre** de la vitre.
Le paramétrage est donné sur la figure 4 où seules les actions normales sont représentées. Le contact entre le joint inférieur et la vitre est permanent et se fait approximativement sur un segment de longueur $L = 776$ mm. Le contact entre les joints latéraux (gauche et droit) se fait progressivement au cours du déplacement de la vitre.
La hauteur des deux joints, supposés identiques, est $H = 450$ mm.

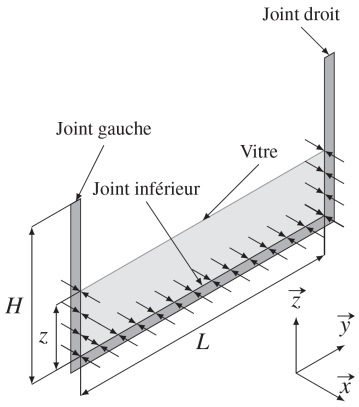


FIGURE 4 : Paramétrage et représentation des efforts normaux uniquement

Le coefficient de frottement entre un joint et la vitre est pris égal à $f = 0{,}5$. Les zones de contact sont supposées être linéiques et la densité linéique d'effort au contact entre un joint et la vitre est supposée constante et égale à $p = 25$ N·m^{-1}.

Q1. Déterminer l'expression littérale de la résultante selon $\vec{z}$ de l'action mécanique du joint inférieur sur la vitre au cours du déplacement de celle-ci.

On suppose que la vitesse de déplacement de la vitre est constante et que le temps du déplacement complet est de 4 s.

Q2. Représenter l'évolution au cours du temps de la résultante des efforts résistants selon $\vec{z}$ de l'ensemble des joints sur la vitre (2 joints verticaux de hauteur H et un joint horizontal de longueur L). Donner les valeurs numériques minimale et maximale de cet effort.

Sur le schéma-blocs de la figure 5, apparaissent les actions de frottements des joints qui sont exercées sur la vitre.

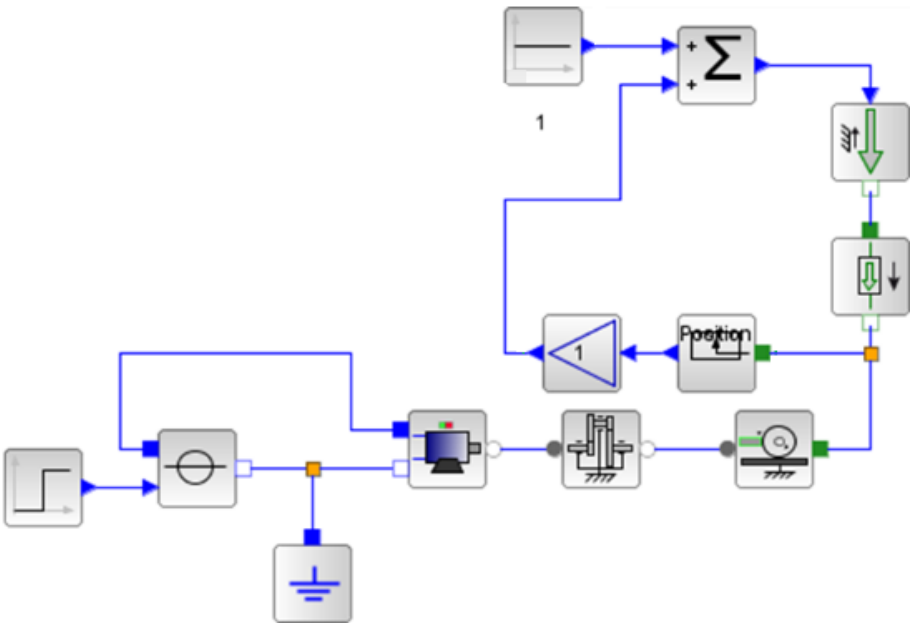


FIGURE 5 : Schéma-blocs à compléter

Q3. Indiquer, en entourant sur la figure 5, l'action du joint horizontal inférieur (en rouge) et l'action des joints verticaux latéraux (en vert).

Modélisation du contact avec un obstacle

Dans le cas d'un ouvrant piloté, l'obstacle est souvent une main. Des études montrent que les phalanges sont très résistantes et peuvent supporter des efforts allant de 250 à 1 150 N en fonction des différentes phalanges.

On modélise donc l'obstacle entre le châssis et la vitre par une raideur k (cette raideur peut varier de 10 à 50 N/mm).

Q4. Compléter, en réalisant un tracé de couleur bleue, le schéma-blocs multiphysique de la figure 6 pour prendre en compte cet obstacle. Une palette composée de constituants standards est donnée sur la figure 7.

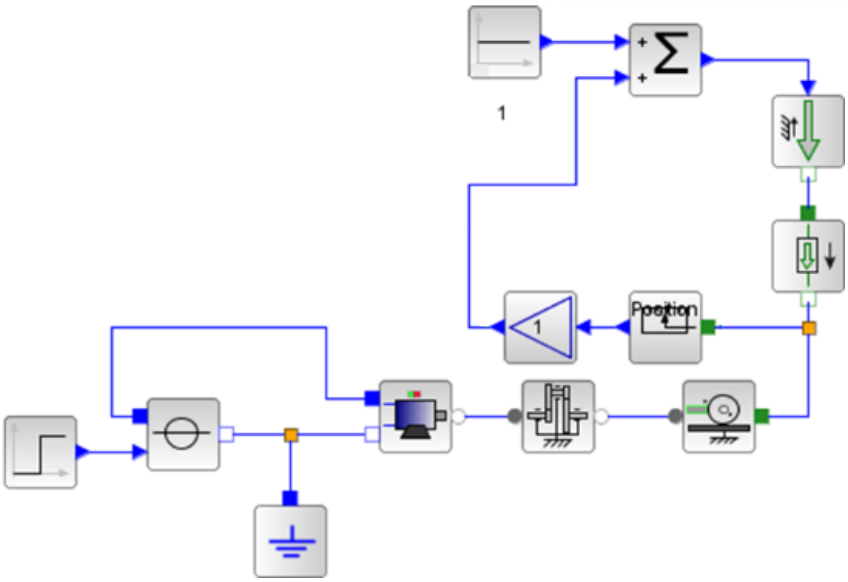


FIGURE 6 : Schéma-blocs à compléter

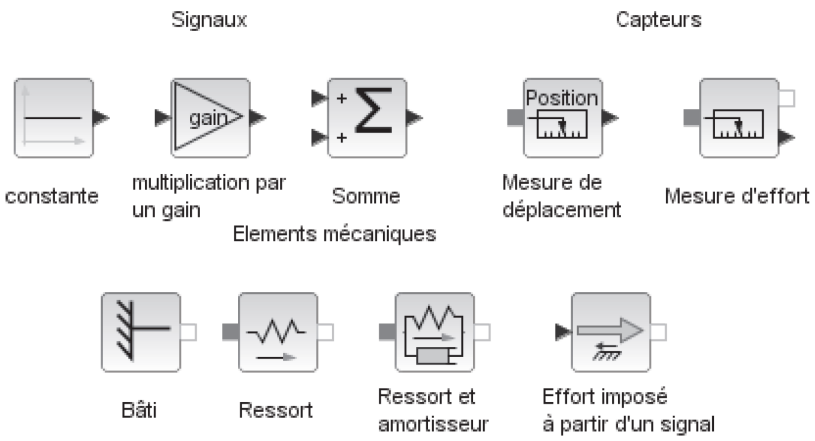


FIGURE 7 : Palette de constituants ou fonctions standards

Validation du modèle simplifié

Les caractéristiques et équations classiques du moteur à courant continu permettent de compléter le modèle multiphysique du lève-vitre.

La simulation de la figure 8 compare l'évolution de la position de la vitre et de la vitesse du moteur selon trois cas sans obstacle : sans effort résistant, pour un effort résistant constant moyen et pour un effort résistant variable en fonction de la position de la vitre.

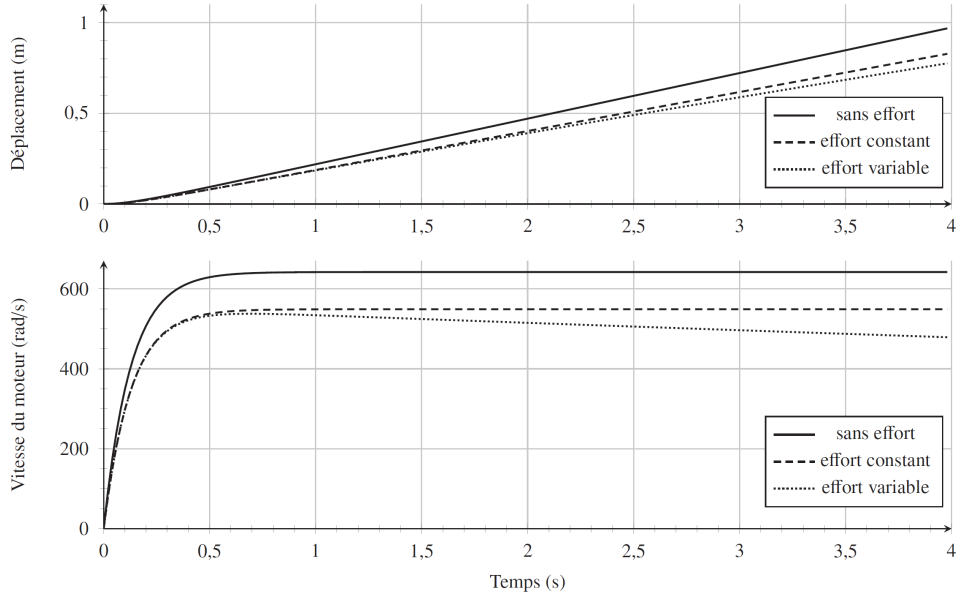


FIGURE 8 : Courbes de la position de la vitre et de la vitesse de rotation du moteur

Q5. Dans chacune des situations, relever le temps au bout duquel la vitre atteint la position maximale définie dans le diagramme des exigences. Commenter l'influence de l'effort résistant sur la vitesse en régime permanent et en régime transitoire. Justifier votre choix entre un modèle sans effort résistant et un modèle avec prise en compte de l'effort résistant.

Pour détecter un pincement, une solution envisagée est d'utiliser le courant dans le moteur et de repérer une variation de ce courant. La simulation multiphysique permet de calculer le courant dans le moteur dans le cas de la présence d'un obstacle ou non.

Les courbes de la figure 9, ont été obtenues en prenant une raideur d'obstacle de 20 N/mm et correspondent à la position de la vitre en m, à l'intensité du moteur en A et à l'effort de pincement en N.

Q6. Déterminer l'intervalle de temps où l'effort est inférieur à la force maximale admissible donnée par la législation (diagramme des exigences de la figure 2). En déduire la variation de courant sur cet intervalle et la comparer à celle obtenue au démarrage. Conclure sur la fiabilité de la mesure de courant pour repérer précisément un obstacle.

Commande tout ou rien

Mesure de la position de la vitre

La position de la vitre est détectée à l'aide de capteurs à effet Hall situés près du moteur (figure 10). Une roue magnétique possédant 2 paires de pôles Nord/Sud est solidaire de l'axe du rotor du moteur. Deux capteurs à effet Hall sont placés en quadrature et repèrent les

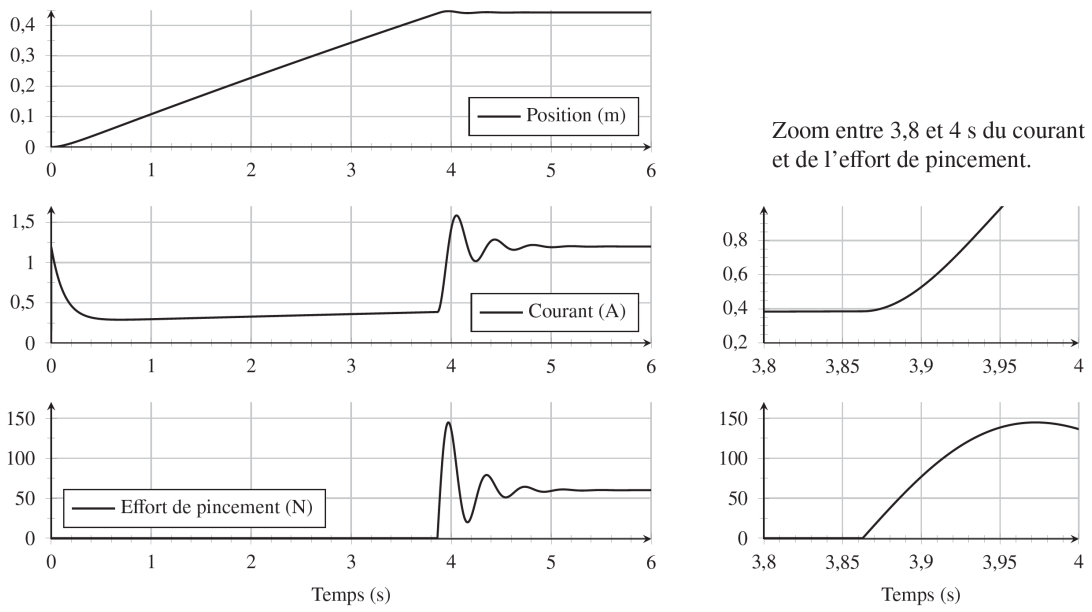


FIGURE 9 : Courbes de position de la vitre, d'intensité moteur et d'effort de pincement

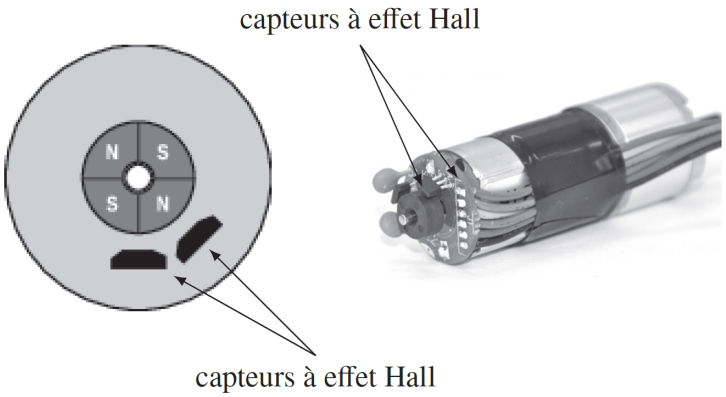


FIGURE 10 : Principe du capteur à effet Hall

changements de champ magnétique (fronts montants et descendants) de la roue en fonction de la rotation du moteur.

Q7. Quels sont les intérêts d'utiliser deux capteurs à effet Hall placés en quadrature ?

Compte-tenu des capteurs utilisés, il est possible d'obtenir une précision de $1/8^e$ de tour du moteur.

Q8. Déterminer le plus petit déplacement de la vitre en mm qu'il est possible de mesurer avec ce capteur.

Q9. En prenant une raideur d'obstacle $k = 20$ N/mm correspondant à la dernière phalange de l'auriculaire, combien d'impulsions auront été comptées à partir du moment où la phalange commence à être écrasée jusqu'à ce que l'effort dans la phalange soit de 50 N (diagramme des exigences, figure 2) ? Commenter ce résultat.

Analyse de la qualité de la mesure de la vitesse

Pour détecter un obstacle, une solution envisagée est d'utiliser la mesure de la vitesse dont les variations sont plus grandes que celles du courant. La vitesse de rotation du moteur est tout

d'abord calculée. Lorsque la variation de cette vitesse est supérieure à une valeur donnée, on indique qu'un obstacle est rencontré et le moteur est stoppé.

Les simulations de la partie précédente montrent qu'il faut détecter rapidement la variation de vitesse, ce qui impose de prendre une période d'échantillonnage de 10 ms au maximum. Ainsi, toutes les 10 ms, le programme va calculer la vitesse en prenant le nombre d'impulsions comptées depuis le dernier calcul et en le divisant par la période d'échantillonnage.

Q10. En supposant que le moteur tourne parfaitement à la vitesse nominale de 300 rad/s, déterminer le nombre d'impulsions moyen N_{moy} mesuré à chaque période d'échantillonnage.

Le nombre N réellement utilisé par le programme est un entier égal, soit à l'entier immédiatement inférieur à N_{moy}, soit à l'entier immédiatement supérieur. Par conséquent, il y a deux valeurs possibles pour la vitesse de rotation du moteur.

Q11. Déterminer les deux valeurs extrêmes de rotation du moteur en tours/min.

Q12. Conclure quant à la pertinence de l'utilisation de la variation de la vitesse pour obtenir un résultat fiable pour la détection. Au vu de la simulation de la figure 8, commenter également l'hypothèse de vitesse constante avant détection d'obstacle.

Mise en place de l'algorithme de commande

L'algorithme finalement mis en place se base sur la variation des temps mesurés entre deux impulsions successives. Après la détection d'une impulsion, un prédicteur temporel permet de déterminer le temps auquel la prochaine impulsion est attendue. Si la nouvelle impulsion intervient avant le temps prédit, alors il n'y a pas de blocage, sinon un blocage est détecté et une alarme est déclenchée.

En réalité, cette technique conduit à de fausses détections et une modification permettant d'améliorer la robustesse est de ne déclencher l'alarme qu'au bout de 3 dépassements du temps prédit.

Cet algorithme est résumé sur la figure 11 pour lequel :

- **appui bouton haut** est un évènement qui survient quand le bouton « monter la vitre » est actionné ;
- **M+** est la variable permettant de faire tourner le moteur dans le sens de la montée de la vitre, **M0** permet d'arrêter le moteur ;
- **impulsion** est un évènement qui survient à chaque nouvelle impulsion envoyée par les capteurs ;
- **fin course haut** est un événement permettant de détecter l'arrivée en position haute de la vitre ;
- **prediction()** est une fonction qui renvoie le temps auquel la prochaine impulsion est attendue ;
- **alarme** permet d'activer l'alarme.

Q13. Donner l'expression des deux conditions notées « transition 1 » et « transition 2 » permettant de passer de l'état montée à l'état arrêt directement.

Q14. Compléter le chronogramme de la figure 12 en indiquant par des créneaux les durées pendant lesquelles un état est activé et l'évolution du contenu de la variable N. La durée de l'alarme et de l'arrêt est supposée très faible et sera représentée par un Dirac (une impulsion).

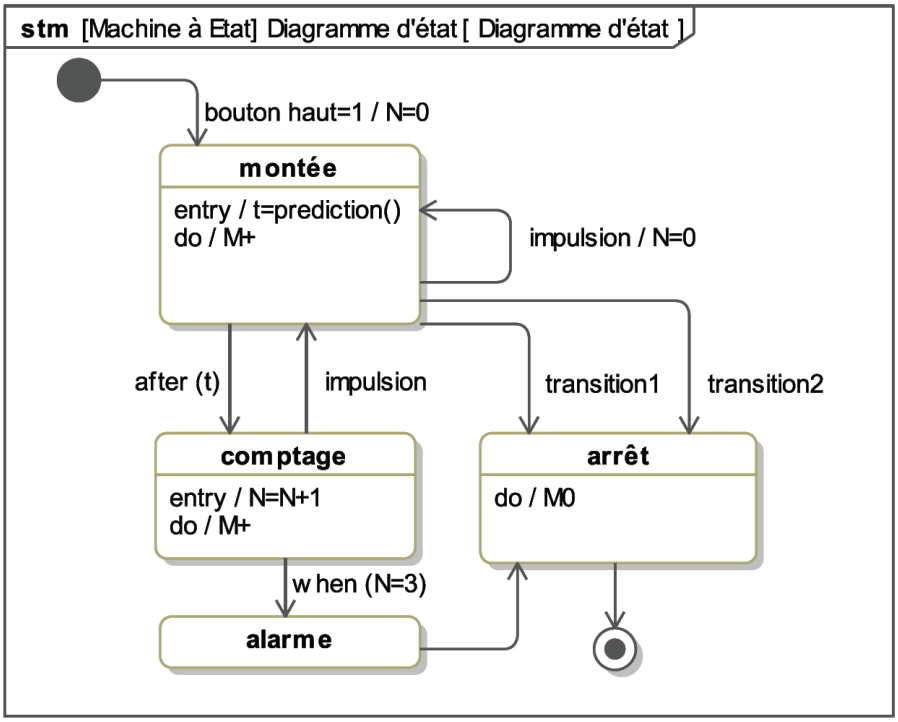


FIGURE 11 : Diagramme d'états de l'algorithme en version simplifiée

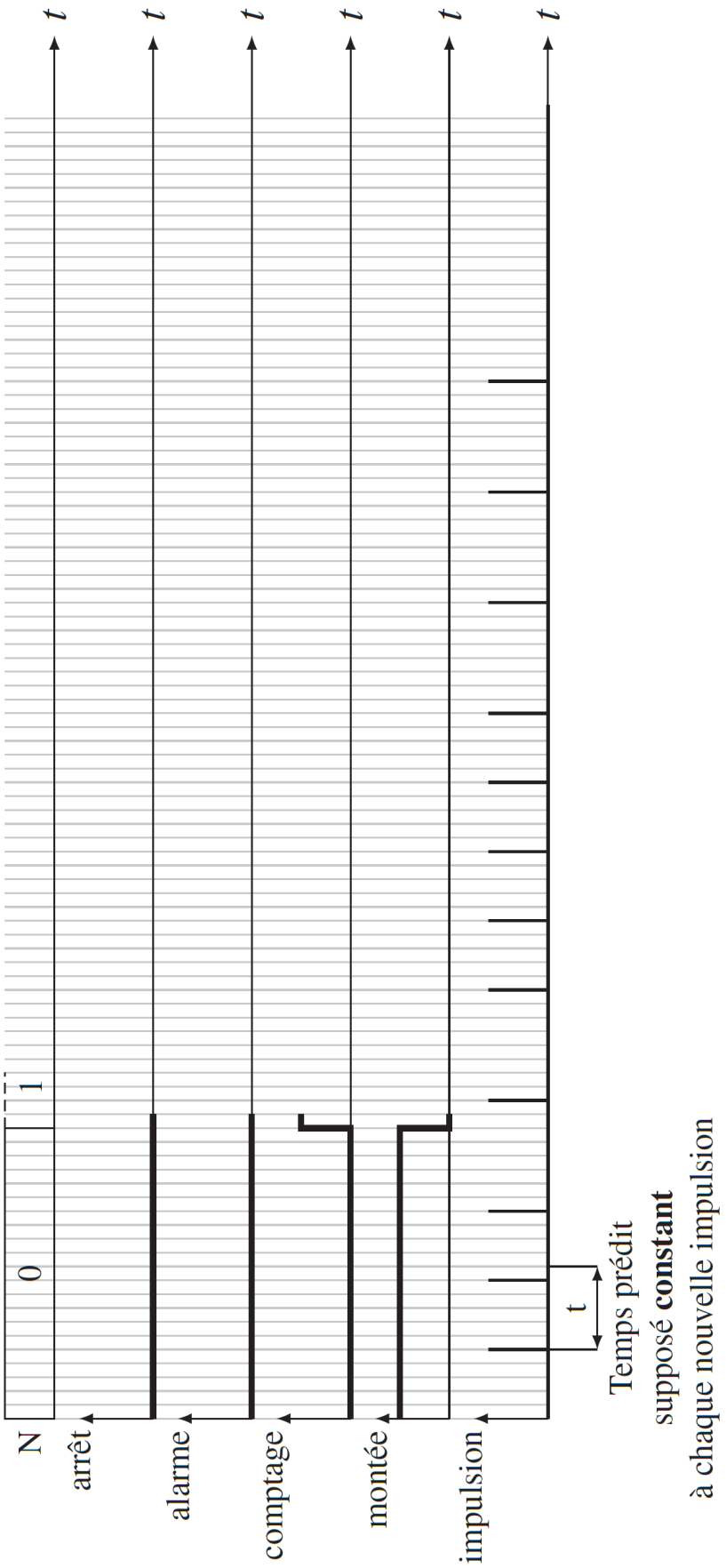


FIGURE 12 : Chronogramme à compléter

TD 5 – Gestion d'un codeur incrémental[1]

Les usagers de deux-roues sont soumis à un risque accru d'accidents en comparaison aux autres catégories d'usagers. Dans le but de réduire ce risque, la simulation de conduite offre une nouvelle opportunité pour appréhender le comportement des conducteurs dans un cadre sécuritaire et constitue un outil alternatif pour la formation à la conduite.
L'objectif de la simulation de conduite est de stimuler le conducteur afin de donner l'illusion d'une conduite sur un véhicule réel. Cette illusion est un phénomène complexe qui met en jeu les capteurs proprioceptifs de l'être humain, notamment ceux des systèmes visuels, somesthésiques et vestibulaires.

FIGURE 1 : Chute de moto

Concevoir un simulateur nécessite de faire des compromis entre la fidélité de la reproduction perceptive et le coût global de l'architecture proposée. Jouer sur la stimulation conjointe des différents systèmes de perception permet d'augmenter l'illusion de mouvement.

Préambule – Présentation du simulateur

Le simulateur Ifsttar (Institut français des sciences et technologies des transports, de l'aménagement et des réseaux) étudié est un simulateur à plate-forme mobile. Afin de restituer les sensations de mouvement aux utilisateurs du simulateur, trois degrés de liberté ont été privilégiés :

- le roulis, c'est le mouvement le plus important dans la dynamique de la moto. Ce degré de liberté est essentiel à la stabilisation et au guidage du véhicule. Il intervient surtout dans la simulation de manœuvres de prise de virages, de slalom et de changement de voie ;
- le tangage, ce mouvement est utilisé pour restituer une partie de l'accélération longitudinale ressentie lors des phases d'accélération et de freinage, celui-ci étant accompagné d'un mouvement de plongée de la fourche ;
- le lacet, ce mouvement a été sélectionné spécifiquement pour reproduire le dérapage de la roue arrière de la moto comme dans le cas de situations classiques de danger.

La plateforme du simulateur (figures 2 et 3) se compose essentiellement d'un bâti métallique fixe et d'un châssis de moto mobile.

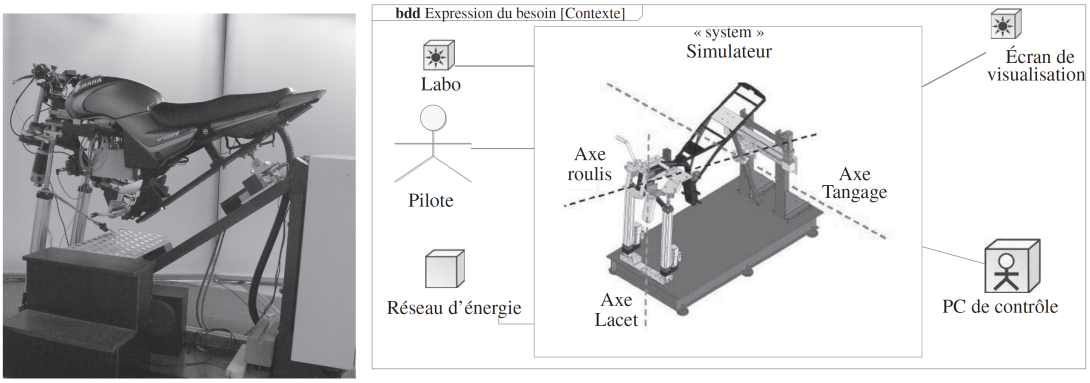
Les deux sous-systèmes « vérins avants » et « glissière arrière » permettent de générer les mouvements de roulis, tangage et lacet du châssis de la moto par rapport à la structure fixe.

Pilotage des vérins de roulis

Objectif

Vérifier le comportement dynamique des vérins

1. Extrait CCINP - PSI - 2019



(a) Simulateur Ifsttar (b) Diagramme de contexte

FIGURE 2 : Simulateur de moto

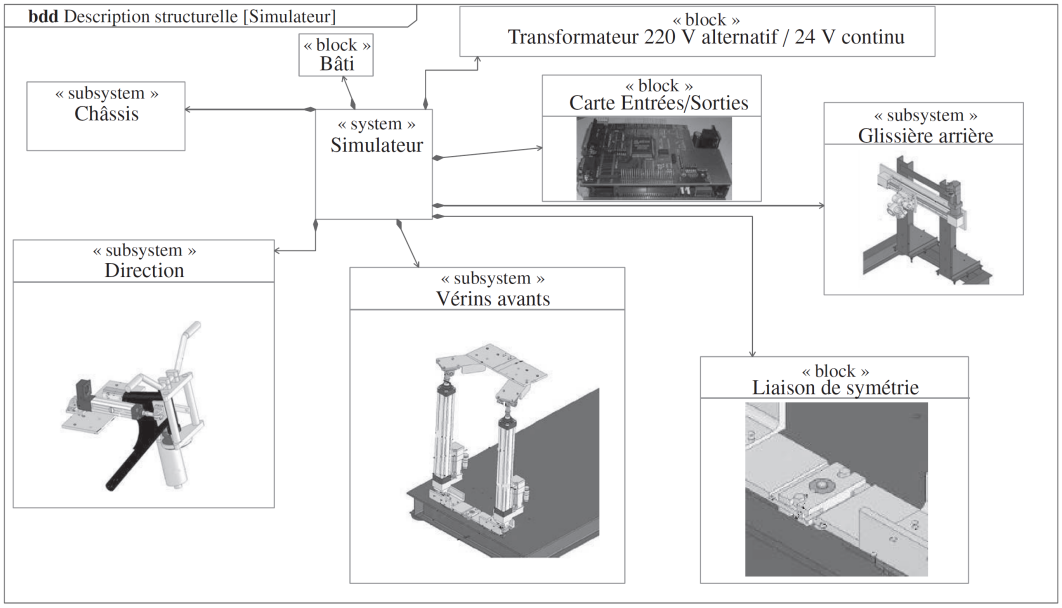


FIGURE 3 : Diagramme de définition de blocs

Les vérins utilisés sont du type vérin électrique. Ils sont constitués d'un actionneur électrique du type machine à courant continu, d'un réducteur de vitesse, et d'un système de transformation de mouvement du type rotation vers translation (système vis-écrou).

Afin de contrôler la position linéaire des vérins avants, des capteurs de position angulaire sont utilisés. La technologie utilisée pour ces capteurs de position angulaire est du type codeur optique composé de deux voies (Voies A et B) qui permettent de détecter le sens de rotation, dont le principe de fonctionnement est décrit sur la figure 4.

La gestion de la position angulaire (comptage des impulsions N_{mes}) et du sens de rotation est décrite par le diagramme d'états fourni sur la figure 5.

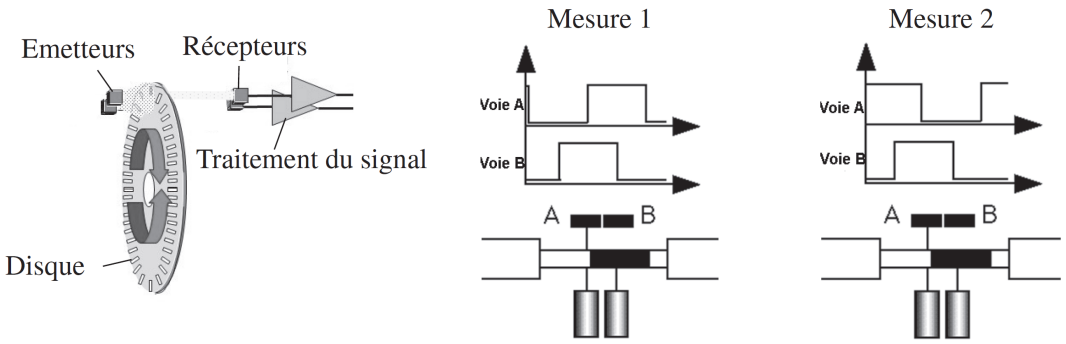


FIGURE 4 : Fonctionnement du codeur incrémental

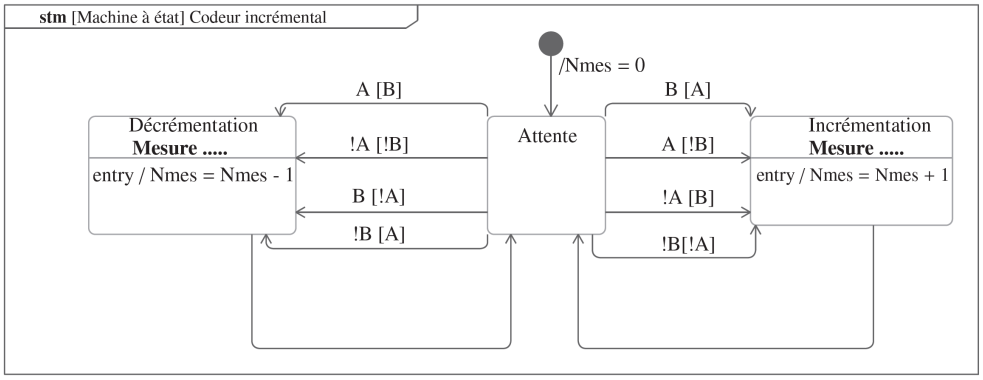


FIGURE 5 : Diagramme d'états de gestion de la position $N_{mes}(t)$

Le complément de la variable logique A (respectivement B) est noté !A (respectivement !B) sur le diagramme. Selon le contexte, la notation A pourra se référer à un évènement ou à une condition de garde.

L'allure des signaux reçus (après traitement électronique) est donnée sur la figure 6.

Q1. Compléter sur la figure 6, le chronogramme donnant l'évolution de la valeur N_{mes} renvoyée par le compteur. Indiquer sur le diagramme d'états (figure 5) à quel numéro de mesure (mesures numérotées sur la figure 4) correspond chacun des états.

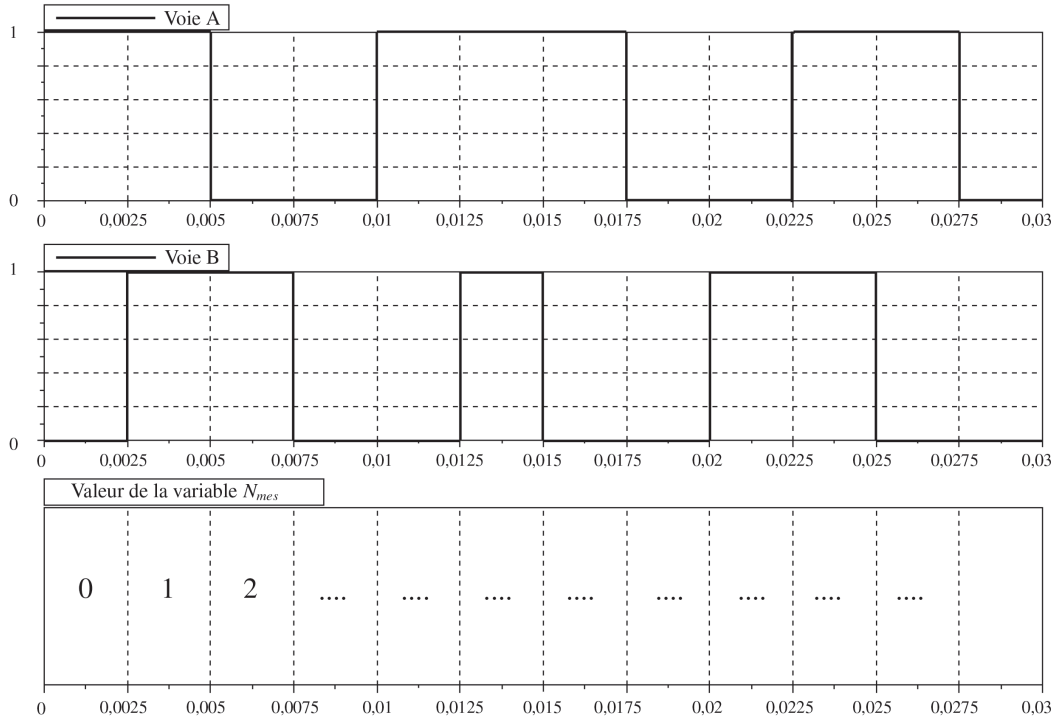
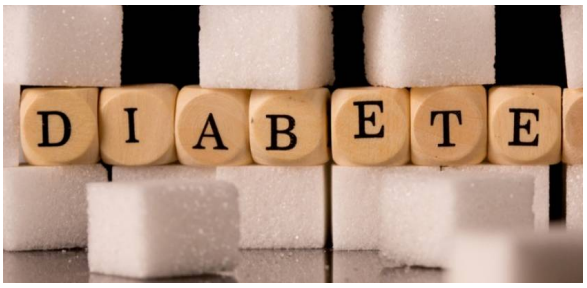


FIGURE 6 : Allure des signaux délivrés par un codeur incrémental

TD 6 – Régressions linéaires

Ce TD est composé de 2 exercices similaires. L'un sur la modélisation de l'évolution de la maladie du diabète chez des patients diabétiques, et l'autre sur le coût de la vie.

Suivi de la maladie du diabète



Présentation

Le suivi des patients diabétiques est un des enjeux majeur pour les médecins, et de nombreuses études consistent à évaluer l'évolution du diabète au cours du temps, notamment après la prise d'un traitement spécifique.

Dix variables de base, l'âge, le sexe, l'indice de masse corporelle, la pression artérielle moyenne et six mesures de sérum sanguin ont été obtenues pour chacun des n=442 patients diabétiques, ainsi que la réponse d'intérêt (mesure quantitative de la progression de la maladie) un an après la prise du traitement.

Ces données sont stockées dans le fichier `diabetesOK.csv`, qui vous a été envoyé par email. Celui-ci contient 10 caractéristiques de chacun des n=442 patients diabétiques, qui sont :

- `age`, *age in years* – age du patient en année ;
- `sex`, le sexe du patient ;
- `bmi`, *body mass index* – indice de masse corporelle ;
- `bp`, *average blood pressure* – pression sanguine moyenne ;
- `tc`, *total serum cholesterol* – cholestérol sérique total ;
- `ldl`, *low-density lipoproteins* – lipoprotéines de basse densité ;
- `hdl`, *high-density lipoproteins* – lipoprotéines de haute densité ;
- `tch`, *total cholesterol/hdl* – rapport cholestérol total sur hdl ;
- `ltg`, *possibly log of serum triglycerides level* – logarithme du taux de triglycérides sériques ;
- `glu`, *blood sugar level* – taux de sucre dans le sang.

et la mesure quantitative de la progression de la maladie.

On note `X` le tableau contenant les caractéristiques de chaque patient (en colonne) et n=442 lignes correspondant à l'ensemble des patients, et `Y` le vecteur colonne contenant la mesure quantitative de la progression de la maladie (de n=442 lignes).

Le cahier des charges relatifs à l'exigence « Prévoir la mesure quantitative de la progression de la maladie du diabète à partir de certaines caractéristiques des patients diabétiques » est fourni ci-dessous.

Exigence	Critère	Niveau
Prévoir la mesure quantitative de la progression de la maladie	RMSE	<8 (unité de Y)
	R^2	>0.8

Mise en place d'une régression linéaire multivariable sur les 10 caractéristiques

Objectif

Déterminer si une loi linéaire ($Y = f(X)$) vis-à-vis de chaque caractéristique permet de modéliser la mesure quantitative de la progression de la maladie.

Les questions suivantes doivent être traitées à partir du fichier `TD6_Reg_Lin.py` envoyé par email, et la base de données `diabeteOK.csv`.

Q1. Télécharger les fichiers `TD6_Reg_Lin.py` et `diabeteOK.csv` et compléter, dans la Zone 1 du fichier Python, le chemin absolu où se trouve le fichier de la base de données.

Les données des caractéristiques des patients ont préalablement été normalisées afin d'éviter tout biais.

Q2. Analyser le code Python, dans la Zone 2, relatif à la définition de la fonction `lecture_data`. Expliquer le rôle de cette fonction, et ce qu'elle renvoie.

Q3. En complétant la Zone 2, procéder à l'appel de la fonction `lecture_data` afin de stocker, dans une variable `X` **toutes** les caractéristiques des patients, et dans une variable `Y` les mesures quantitatives de la progression de la maladie.

Une partie des n=442 données sera utilisée pour l'apprentissage et le complément pour le test, et donc la validation du modèle.

Q4. Analyser le code fourni dans la Zone 3, et le compléter afin de :
- disposer dans la variable `X_train` des caractéristiques qui seront utilisées pour l'apprentissage ;
- disposer dans la variable `X_test` des caractéristiques qui seront utilisées pour le test ;
- disposer dans la variable `Y_train` des mesures quantitatives de la progression de la maladie qui seront utilisées pour l'apprentissage ;
- disposer dans la variable `Y_test` des mesures quantitatives de la progression de la maladie qui seront utilisées pour le test.

Les données utiles à l'apprentissage (et au test) sont désormais chargées, il est possible de définir le type de modèle à utiliser et sa détermination.

La modélisation retenue dans un premier temps est une modélisation linéaire multivariable sur l'ensemble des 10 caractéristiques telle que :

$$Y = a_0 + a_1 \cdot age + a_2 \cdot sex + a_3 \cdot bmi + a_4 \cdot bp + a_5 \cdot tc + a_6 \cdot lpl + a_7 \cdot hdl + a_8 \cdot tch + a_9 \cdot ltg + a_{10} \cdot glu$$

Dans le cadre d'une régression linéaire (ici, multivariable), il est possible d'utiliser le module `sklearn`, et ses fonctions associées, afin de déterminer notamment les coefficients a_i optimaux (minimisant en moyenne le carré de l'écart entre les données d'apprentissage et les données prédites).

On donne ci-dessous les fonctions utiles pour ce TD.

```
# Importation du module dédié à la régression linéaire
from sklearn import linear_model

# Création du modèle d'une régression linéaire
modele=linear_model.LinearRegression()
# Lancement de la phase d'apprentissage sur les données d'
   apprentissage data_train et Y_train
modele.fit(data_train,Y_train)
# Affichage des poids associés à chaque caractéristique (ai)
print(modele.coef_) # Liste des poids ai avec i!=0
print(modele.intercept_) # Ordonnée à l'origine a0
# Affichage de l'ordonnée à l'origine
# Prévision à partir du modèle sur la donnée new_data
modele.predict(new_data)
```

Q5. Dans la Zone 4, compléter le code permettant de déclarer le modèle sous le nom `regr_diabete`, d'exécuter la phase d'apprentissage sur les données d'apprentissage (`X_train` et `Y_train`) et d'afficher les poids relatifs aux différentes caractéristiques et l'ordonnée à l'origine.

Nous disposons maintenant d'un modèle suite à la phase d'apprentissage. Il est possible de déterminer les mesures quantitatives de la progression de la maladie prédites sur les données de test.

Q6. Compléter, dans la Zone 5, le code permettant de prévoir les mesures quantitatives de la progression de la maladie sur les données de test (`X_test`), notées `Y_pred`. Les afficher et les comparer aux valeurs des données de test `Y_test`.

Détermination des métriques et validation du cahier des charges

Nous disposons désormais du modèle de régression linéaire multivariable et il est nécessaire de valider ce modèle en comparant ses performances à celles attendues dans le cahier des charges.

La détermination des métriques permettant de caractériser les performances de la modélisation linéaire est possible en utilisant directement les fonctions associées au module `sklearn`.

```
# Importation des fonctions liées à l'évaluation des métriques
# Pour la détermination de la MSE
from sklearn.metrics import mean_squared_error
# Pour la détermination du coefficient de détermination R2
from sklearn.metrics import r2_score
```

Les 2 fonctions permettant l'évaluation des métriques MSE (*mean squared error* et coefficient de détermination R^2) nécessitent 2 arguments, le premier est le tableau des valeurs test (`Y_test`) et le second est le tableau des valeurs prévues par le modèle (`Y_pred`).

Q7. Compléter, dans la Zone 6, le code permettant de déterminer et d'afficher les métriques RMSE (racine carrée de MSE) et coefficient de détermination, à partir de notre modèle de régression linéaire multivarié sur les 10 caractéristiques des patients.

Q8. Conclure quant à la satisfaction ou non du cahier des charges.

Et le coût de la vie...

Présentation

On s'intéresse dans ce travail à la modélisation du pouvoir d'achat d'un ménage en fonction de son lieu d'habitation dans le monde. À cet effet, on dispose d'une base de données de 563 enregistrements, fournie dans le fichier `cost-of-living-2016.txt`.
Les caractéristiques (*features*) retenues sont :

- `PrixLitreLait`, correspondant au prix d'un litre de lait ;
- `PrixAboTransport`, correspondant au prix mensuel de l'abonnement de transport en commun ;
- `PrixHotel`, correspondant au prix moyen d'une chambre d'hôtel standard ;
- `PrixAboInternet`, correspondant au tarif mensuel moyen de l'abonnement Internet ;
- `PrixCafe`, correspondant au prix d'un café dans un bar ;
- `PrixBtlleEau`, correspondant au prix de la bouteille d'eau de 33 cl en supermarché ;
- `PrixDouzEufs`, correspondant au prix d'une douzaine d'oeufs ;
- `Prix1L5Eau`, correspondant au prix d'une bouteille d'un litre et demi d'eau ;
- `PrixPinteBiere`, correspondant au prix d'une pinte (demi-litre) de bière dans un bar ;
- `PrixTicketBus`, correspondant au prix d'un ticket de bus aller-retour ;
- `Coutbasique`, correspondant aux dépenses d'électricité, en eau, en chauffage et ordures ménagères pour un logement de 85 m^2 ;
- `PrixPlaceCinema`, correspondant au prix d'une place de cinéma ;
- `PrixKgPommes`, correspondant au prix d'un sachet de 1 kg de pommes.

Les grandeurs de sortie sont :

- `IndexCdlV`, correspondant à l'indice du coût de la vie ;
- `IndexLocation`, correspondant à l'indice représentant les coûts de location ;
- `IndexCdlVPlusLocation`, correspondant à l'indice représentant le coût de la vie en y associant les dépenses liées à la location ;
- `IndexCourses`, correspondant aux dépenses liées aux courses ;
- `IndexPrixRestaurant`, correspondant aux dépenses liées à la restauration dans un restaurant ;
- `IndexPouvAchatLocal`, correspondant au pouvoir d'achat dans la ville concernée.

Chaque grandeur de sortie dépend de plusieurs caractéristiques, par conséquent, une régression linéaire multivariée doit être mise en place. L'hypothèse de modèles linéaires (par opposition à des modèles multivariés polynomiaux) est faite, afin de décrire l'évolution des grandeurs de sorties en fonction des caractéristiques.

On estime que la modélisation est pertinente si le coefficient de détermination R^2 est supérieur à 0.85 pour chaque prédicteur.

Par ailleurs, dans une première approche, on considère que les caractéristiques influentes pour les grandeurs de sorties sont :

Grandeurs de sortie	Index dans bdd	Caractéristiques pertinentes	Index dans bdd
IndexCourses	16	PrixLitreLait	0
		PrixDouzEufs	6
		Prix1L5Eau	7
		PrixKgPommes	12
IndexLocation	14	PrixAboTransport	1
		PrixHotel	2
		PrixAboInternet	3
		PrixTicketBus	9
		PrixPlaceCinema	11
		Coutbasique	10
IndexPrixRestaurant	17	PrixLitreLait	0
		PrixBtlleEau	5
		PrixDouzEufs	6
		PrixPinteBiere	8
		PrixTicketBus	9
IndexPouvAchatLocal	18	PrixAboTransport	1
		PrixHotel	2
		PrixAboInternet	3
		PrixTicketBus	9
		PrixPlaceCinema	11

Extraction des données

Q1. Télécharger le fichier `cost-of-living-2016.txt` et compléter, dans la Zone 1 du fichier `TD6_Reg_Lin.py` relative au cout de la vie, le chemin absolu où se trouve le fichier de la base de données.

Q2. Dans la Zone 2, et en vous inspirant de la définition de la fonction `lecture_data`, définir une fonction `lecture_data2` prenant en arguments :

- la variable `nom`, du type chaîne de caractères, correspondant au nom de la base de données ;
- la variable `liste_colonne_car`, correspondant à la liste des indices des caractéristiques utiles de la base de données pour prévoir la grandeur de sortie ;
- la variable `liste_colonne_Y`, correspondant à la liste des indices des grandeurs de sortie de la base de données.

Elle renvoie 2 tableaux, `X` et `Y`, du type tableau, contenant respectivement les caractéristiques des données et les grandeurs de sortie.

Q3. En complétant la Zone 2, procéder à l'appel de la fonction `lecture_data2` afin de stocker, dans une variable `X` les caractéristiques pertinentes, et dans une variable `Y` les valeurs de la grandeur `IndexCourses`.

Mise en place des modèles et leurs validations

Q4. En vous inspirant des questions 4, 5, 6 et 7 de l'exercice précédent sur la maladie du diabète, compléter, dans la Zone 3, les codes permettant de créer les modèles de prévision des grandeurs de sortie `IndexCourses`, `IndexLocation`, `IndexPrixRestaurant` et `IndexPouvAchatLocal` et d'évaluer leurs coefficients de détermination respectifs. On utilisera 80% des données pour l'apprentissage, et donc 20% pour le test.

Q5. Conclure quant à la satisfaction du cahier des charges imposant un coefficient de détermination supérieur à 0.85.

Q6. Pour les éventuels modèles ne respectant pas le cahier des charges, proposer (sans les mettre en œuvre) des solutions qui permettraient d'améliorer la modélisation de ces grandeurs.

TD 7 – Partitionnement de vins par algorithme des k-means

Présentation

Pour mettre en œuvre l'algorithme des k-moyennes, nous allons utiliser une base de données sur les vins italiens de la région du Piémont. Dans cette région, on cultive essentiellement 3 cépages, Barbera, Nebbiolo et Grignolino.

FIGURE 1 : Cépages principaux dans la région du Piémont en Italie

Nous disposons d'une base de données de 178 bouteilles qui ont été analysées chimiquement selon 27 critères. Pour le partitionnement, nous n'avons retenu que les 8 caractéristiques les plus intéressantes.

Un extrait de cette base de données est fourni ci-dessous.

Alcohol (% vol)	Malic acid (g/L)	Total phénols (g/L)	Flavanoids (dg/L)	Non-flavanoids (dg/L)	Color Intensity*	Color Hue*	Proline (mg/L)
14.23	1.71	2.8	3.06	0.28	5.64	1.04	1065
13.16	2.36	2.8	3.24	0.3	5.68	1.03	1185
13.2	1.78	2.65	2.76	0.26	4.38	1.05	1050
14.37	1.95	3.85	3.49	0.24	7.8	0.86	1480
13.24	2.59	2.8	2.69	0.39	4.32	1.04	735

TABLEAU 1: Extrait de la base de données des vins du Piémont

Remarque : Les caractéristiques associées à la couleur du vin sont calculées à partir des absorbances à 420 nm et à 520 nm :

$$\text{Color Intensity} = 23 \cdot \log(A_{420} + A_{520}) \text{ et Color Hue} = A_{420}/A_{520} \text{ (teinte)}$$

Objectif

Analyser les données et déterminer si ces vins ont des points communs et donc, s'il est possible de les regrouper (au sens partitionner).

Pour atteindre l'objectif, l'utilisation du module `sklearn` de `Python` sera faite. On donne ci-dessous les éléments principaux pour l'utilisation du module concerné.

```
from sklearn.cluster import KMeans
model=KMeans(n_clusters=3) # On définit le modèle de
    classification et l'hyperparamètre (ici 3)
model.fit(X) # On entraine le modèle sur des données d'
    apprentissage X
model.predict(Xnew) # Utilisation du modèle pour faire la pré
    vision de Xnew
model.cluster_centers_ # Permet de renvoyer les coordonnées des k
    centroides
model.inertia_ # On peut évaluer l'efficacité du modèle
```

Q1. Quel type d'apprentissage doit être mis en œuvre ? Justifier votre réponse.

Les questions suivantes doivent être traitées à partir du fichier `TD7_kMeans_Wine.py` envoyé par email, et la base de données `wine.txt`.

Par ailleurs, le chargement de la base de données peut être réalisé par l'instruction

```
np.loadtxt(nom,delimiter,skiprows)
```

où :

- `nom` est le nom du fichier à charger (avec son extension), du type chaîne de caractères ;
- `delimiter` est le type de délimiteur utilisé pour séparer les colonnes, du type chaîne de caractères ;
- `skiprows`, du type `int`, correspond au nombre de ligne(s) en entête du fichier contenant du texte (ou tout élément autre que les données numériques à charger).

Q2. Ouvrir, avec le bloc-Notes (ou équivalent sous Mac), le fichier de données `wine.txt` et déterminer le séparateur (délimiteur) de colonne utilisé, et si la première ligne est celle d'une donnée ou celle des caractéristiques des bouteilles de vin. Dans la zone prévue à cet effet (Zone 1), compléter le code Python permettant de charger le fichier de données `wine.txt` dans la variable `X`.

Q3. Dans la zone 2, compléter le code permettant d'extraire les données relatives aux colonnes `Nonflavanoid_Phenols` et `Proline`, respectivement dans les variables `X1` et `X2`. Exécuter la Zone 2, et visualiser la figure 1. Est-il si évident que 3 cépages sont présents à partir de ces 2 critères ?

Une analyse complémentaire du Tableau 1 permet de conclure quant à un éventuel « biais » dans les critères. En effet, les intervalles de variation des différents critères sont très différents. À titre d'exemples, le critère `Proline` varie entre 278 et 1680, alors que le critère `Nonflavanoid_Phenols` varie entre 0.13 et 0.66. Ainsi, sans normalisation des données, une petite variation du critère `Nonflavanoid_Phenols` a une importance absolue beaucoup plus

grande qu'une petite variation du critère `proline`. Il est donc nécessaire de pré-traiter les données d'apprentissage en réalisant une normalisation. Cette normalisation consiste, ici, à rendre les critères dans l'intervalle [0,1].

Q4. Soit une grandeur x dont l'intervalle de variation est $[min, max]$. On souhaite opérer une même transformation affine à tous les éléments x d'une même colonne afin de ramener leurs valeurs de l'intervalle $[min, max]$ dans l'intervalle [0,1]. Cette nouvelle valeur est appelée x_{norm}. Déterminer l'équation de x_{norm}, en fonction de x, min et max afin que l'intervalle de variation de x_{norm} soit l'intervalle $[0, 1]$. Dans la Zone 3, compléter le code de la fonction `normalise(X)` qui retourne le tableau `X` dont les valeurs de chaque colonne sont comprises entre 0 et 1. Elle devra aussi renvoyer les listes `Mini` et `Maxi` contenant respectivement les minimas et maximas de chaque critère. Procéder à l'appel de cette fonction sur les données d'apprentissage, et vérifier le bon fonctionnement de la normalisation (par exemple, en traçant sur une figure 2, l'équivalent de la figure 1).

Il est désormais possible de créer le modèle de classification par l'algorithme des k-moyennes.

Q5. Compléter, dans la Zone 4, le code permettant de définir le modèle de partitionnement avec k=3 (sous le nom `classifieur_wine`), d'exécuter l'apprentissage et d'afficher les coordonnées des k centroïdes.

Q6. Procéder à la prévision de la classe d'appartenance de la nouvelle donnée `New_Data=np.array([[13.4],[1.8],[3.5],[2.5],[0.15],[5],[1.01],[890]])`, et afficher la classe d'appartenance.

On cherche à vérifier si le choix de 3 groupes (correspondant aux 3 cépages) est globalement pertinent. Afin de valider ce choix, il est usuel d'utiliser le critère de l'inertie, définie par :

$$\text{Inertie} = \sum_{j=0}^{k} \left(\sum_{i=0}^{n} d^2(x_i, c_j) \right)$$

où :

— k le nombre de classes (ici de cépages) ;

— n le nombre de données dans la base d'apprentissage (ici, 178) ;

— $d^2(x_i, c_j)$ est la distance (euclidienne) entre la donnée x_i et le centroïde c_j.

Cette grandeur représente la somme des carrés des distances entre chaque point et le centroïde qui lui est associé.

Si on prend k=1, les données ne sont pas classées. Si l'on prend k=n, l'inertie est nulle puisqu'il y a autant de centroïdes que de points, et donc les distances relatives sont toutes nulles.

Une règle empirique (qui s'appelle heuristique... et donc qui ne se démontre pas théoriquement, mais expérimentalement), appelée *elbow method* (méthode du coude), précise qu'un bon choix de l'hyperparamètre k est lorsque la pente de la courbe Inertie=f(k) tend à s'horizontaliser.

Ainsi, pour valider le choix de k=3, nous souhaitons tracer l'évolution de l'inertie en fonction de k pour $k \in [1, 10]$.

Q7. Compléter la Zone 5 afin de générer la liste des inerties pour $k \in [1, 10]$ et de tracer l'évolution de l'inertie en fonction de k sur la figure 3.

Q8. Identifier le coude sur cette figure, et valider le choix de k=3.

Remarque : L'année prochaine, en classe de seconde année, vous reprendrez cet exercice en réécrivant l'ensemble des fonctions, sans passer par la bibliothèque `sklearn`.

TD 8 – Algorithme k-NN et réseaux de neurones

Objectif

Évaluer les performances d'une classification par algorithme k-NN et par réseau de neurones et faire un choix argumenté.

Présentation

L'étude porte sur une base de données qui répertorie les classes d'appartenance des tarifs du prix des biens moyens par district en Californie aux États-Unis. Cette base de données contient 20 640 données. Chaque enregistrement est caractérisé par 8 caractéristiques et une classe d'appartenance.

Les classes d'appartenance sont au nombre de 4 (de la classe 0 à 3) et correspondent à 4 gammes de prix moyens des biens immobiliers par district dans l'état de la Californie (classe 0, prix inférieur à 100 000 $, classe 1, prix compris entre 100 000 $ et 200 000 $, classe 2, prix compris entre 200 000 $ et 300 000 $ et classe 3, prix supérieur à 300 000 $). Ces prix sont des moyennes dans le district concerné.

Les caractéristiques associées à chaque district sont :

- `MI`, le revenu moyen dans le district exprimé en centaine de milliers de dollar ;
- `HA`, l'age moyen des habitants du district ;
- `TR`, le nombre moyen de pièces d'habitation du district ;
- `TBR`, le nombre moyen de chambres à coucher du district ;
- `Pop`, le nombre d'habitants dans le district ;
- `HH`, le nombre moyen de personnes par famille vivant dans le district ;
- `lat`, latitude du district ;
- `long`, longitude du district.

L'ensemble des questions nécessitant la complétion d'un code informatique, en langage Python, sera traité dans le fichier `TD8_kNN_RN.py` qui vous a été envoyé par email.

La base de données nécessaire à ce travail est déjà pré-implémentée dans l'environnement Python, dans le module `sklearn.datasets` sous le nom `sklearn.datasets.fetch_california_housing()`. Le chargement de cette base de données (dans la variable de nom `bdd`) est réalisé par l'instruction `bdd=sklearn.datasets.fetch_california_housing()`. L'accès aux caractéristiques est disponible par la commande `X=bdd.data` alors que la classe d'appartenance de chaque enregistrement est disponible par la commande `Y=bdd.target`.

MI	HA	TR	TBR	Pop	HH	lat	long	Classe
8.352	41	6.984	1.023	322	2.5555	37.88	-122.23	3
8.301	21	6.238	0.971	2401	2.109	37.86	-122.22	3
3.203	52	5.477	1.079	910	2.263	37.85	-122.26	2
2.125	50	4.242	1.071	697	2.640	37.85	-122.26	1
2.180	52	5.193	1.036	853	2.624	37.84	-122.27	0
...	...	...	...	...	...	...	...	...

TABLEAU 1: Extrait de la base de données

À titre d'information, on donne ci-dessous un extrait de quelques enregistrements de cette base de données.

On considère que la classification est pertinente si la sensibilité (appelée aussi rappel – *recall* en anglais) moyenne est supérieure à 0.55 et la spécificité (appelée aussi précision) moyenne est supérieure à 0.6. On appelle métrique macro-moyenne la moyenne non pondérée des métriques des différentes classes.

Analyse de la base de données

Q1. Est-ce un problème de régression ou de classification qu'il est nécessaire de résoudre ? Est-ce un apprentissage supervisé ou non-supervisé ? Justifier vos réponses.

Q2. Dans la Zone 1 du fichier Python, compléter le code permettant déterminer et d'afficher le nombre d'enregistrements de chaque classe.

On considère qu'il y a un déséquilibre dans la base de données si le cardinal de la classe majoritaire est 2 fois supérieur à celui de la classe minoritaire.

Q3. Peut-on considérer qu'il y a un déséquilibre dans la base de données vis-à-vis du critère retenu ci-dessus ?

Classification par algorithme k-NN

On envisage dans cette partie de mettre en œuvre une classification par l'algorithme des k plus proches voisins, et d'évaluer les performances de classification en caractérisant la sensibilité et la spécificité de la fonction de prédiction pour chaque classe.

On donne ci-dessous le code Python du module `sklearn` permettant la mise en place d'un classifieur du type k plus proches voisins et l'évaluation de ces performances.

```
# Fonction pour la déclaration de l'objet classifieur par kNN
from sklearn.neighbors import KNeighborsClassifier
# Fonction pour séparation données d'apprentissage et de test
from sklearn.model_selection import train_test_split
# Fonction pour le tracé de la matrice de confusion
from sklearn.metrics import confusion_matrix
# Fonction pour l'évaluation des métriques (sensibilité et spé
    cificité)
from sklearn.metrics import classification_report
```

Q4. Analyser le code Python et indiquer le rôle des lignes de code Python de la Zone 2.

Q5. Exécuter le code de la Zone 2 et de la Zone 3. Que représente le contenu de la variable `Y_kNN_predit` ?

Q6. Exécuter le code de la Zone 4, et visualiser la figure 1 relative à la matrice de confusion pour la classification par les k plus proches voisins (avec k=3). La classification des données de tests est-elle parfaite ? Justifier votre réponse.

Q7. Compléter le tableau ci-dessous en indiquant, pour chaque classe, le nombre d'enregistrement considéré comme Vrai Positif (VP), Faux Positif (FP), Vrai Négatif (VN) et Faux Négatif (FN).

	Classe 0	Classe 1	Classe 2	Classe 3
VP				
FP				
VN				
FN				

Q8. Déterminer pour chaque classe, la sensibilité et la sensibilité, en complétant le tableau ci-dessous.

	Classe 0	Classe 1	Classe 2	Classe 3
Sensibilité				
Spécificité				

Q9. Exécuter le Zone 5 du code Python, et visualiser les performances de la classification fournies par Python, à l'aide de l'instruction `classification_report`. Comparer et valider les valeurs calculées à la question 8.

Q10. Conclure sur les performances du classifieur par l'approche des k plus proches voisins vis-à-vis du cahier des charges.

Classification par réseau de neurones

On envisage dans cette partie de mettre en œuvre une classification par un réseau de neurones, et d'évaluer les performances de classification en caractérisant la sensibilité et la spécificité de chaque classe. Ces 2 métriques seront évaluées dans différentes configurations du réseau de neurones en faisant varier le nombre de couches cachées et de neurones dans chacune d'elles et le type de fonction d'activation.

Un premier réseau de neurones

Dans un premier temps, on construit un réseau de neurones, tel que décrit sur la Figure 11.

Q11. Identifier, sur la Figure 11, la couche d'entrée, la couche de sortie, et les éventuelles couches cachées. Dans le cas de couches cachées, déterminer le nombre de neurones par couches cachées.

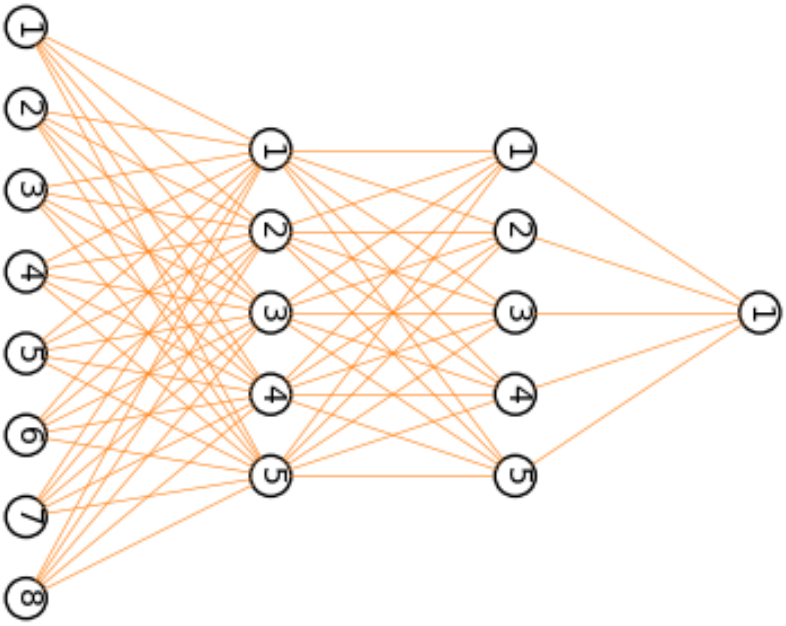


FIGURE 1 : Exemple de réseau de neurones

Q12. Après analyse de la FIGURE **??**, justifier la présence de 8 neurones sur la partie gauche. De même, justifier la présence d'un neurone sur la partie droite. Que représente les 8 entrées des neurones de gauche ? Que représente la sortie du neurone de droite ?

On donne dans la Zone 6 du fichier `TD8_kNN_RN.py` le code Python permettant de déclarer un classifieur par réseau de neurones.

Q13. Analyser les arguments de la fonction permettant la déclaration d'un réseau de neurones (fonction `MLPClassifier`), et les comparer à l'analyse de la question 11. Quelle fonction d'activation est utilisée dans ce classifieur ?

Q14. Exécuter le code de la Zone 7 permettant de calculer la prévision sur les données de test et de tracer la matrice de confusion. Comparer les performances de ce classifieur à celles attendues dans le cahier des charges et à celles du classifieur par l'algorithme du k-NN.

D'autres réseaux de neurones

Afin de mettre en évidence l'influence des paramètres d'un réseau de neurones sur les performances de classification, nous nous proposons de faire évoluer le nombre de couches cachées et au nombre de neurones associés, et à la fonction d'activation.

Remarque : Dans le cas d'un nombre de couches cachées « important » et d'un nombre de neurones « important » dans celles-ci, la durée d'apprentissage peut être un peu longue (quelques secondes, donc patience...).

Q15. Modifier les paramètres du classifieur par réseau de neurones tels que décrit dans le tableau ci-dessous, et évaluer les performances en termes de classification (sensibilité moyenne – *recall avg* et spécificité moyenne – *precision avg*).

Q16. Conclure quant au(x) paramètre(s) du classifieur à réseau de neurones à adopter afin de satisfaire le cahier des charges. Les critères retenus seront bien évidemment les performances de classification et l'espace mémoire de stockage des différents poids et biais associés à chaque neurone.

	(5,5)	(10,10)	(10,10,10)	(100,100)
`relu`	recall avg : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :
`tanh`	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :
`logistic`	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :	*recall avg* : *precision avg* :

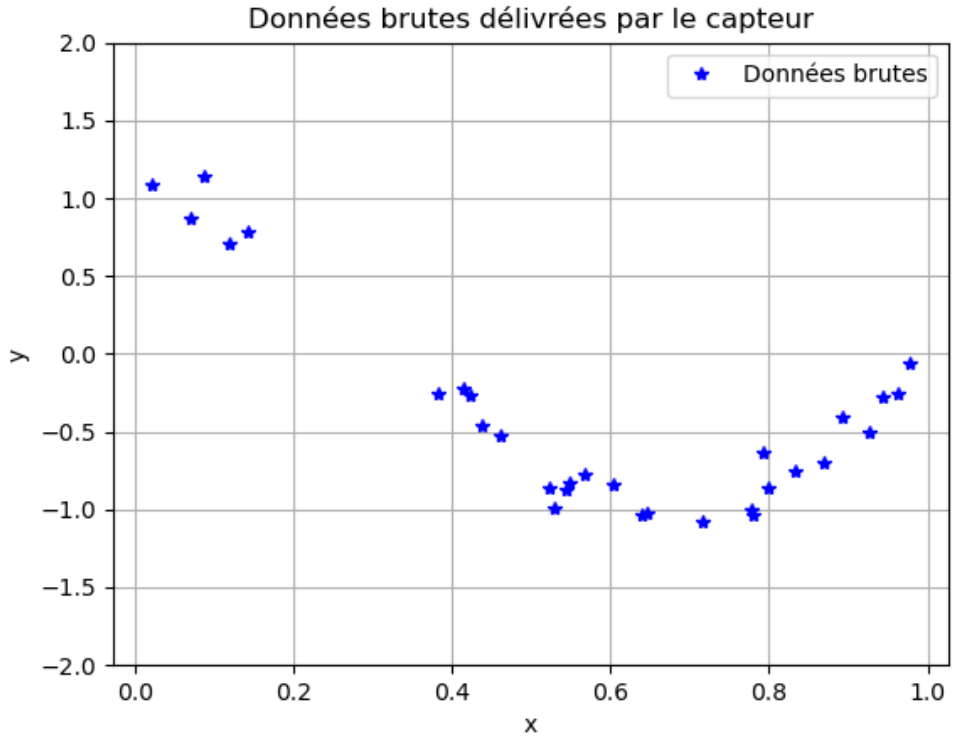
Synthèse sur les classifieurs mis en œuvre

Q17. Dresser une synthèse sur le classifieur à adopter afin de respecter le cahier des charges.

TD 9 – Mise en évidence du phénomène de sur-apprentissage

Présentation du problème

Un capteur délivre un signal y pour une entrée x dans l'intervalle [0,1]. Ce signal est entaché de bruit, qui ne permet pas de disposer d'une information fiable. Par ailleurs, il est difficile d'obtenir le signal de sortie pour une entrée x dans l'intervalle [0.2,0.4]. Il est représenté sur la Figure 11.



Les réponses aux questions doivent être rédigé `TD9_SurApprentissage.py`. La Zone 1 du fichier Python `TD9_SurApprentissage.py` contient les données issues de l'expérience sur le capteur.

Première modélisation - régression linéaire monovariable

On souhaite dans un premier temps proposer un modèle linéaire de la loi entrée-sortie du capteur, par régression linéaire monovariable, et évaluer les performances de celle-ci.

Le cahier des charges relatif à la modélisation est fourni ci-dessous.

Exigence	Critère	Niveau
Valider le modèle du capteur	RMSE	$\leq$0.02 (unité de y)
	R^2	$\geq$0.97

On donne ci-dessous les fonctions utiles pour cette partie.

```
# Importation du module dédié à la régression linéaire
from sklearn import linear_model
# Création du modèle d'une régression linéaire
modele=linear_model.LinearRegression()
# Lancement de la phase d'apprentissage sur les données d'
    apprentissage X et Y
modele.fit(X,Y)
# Prévision à partir du modèle sur la donnée new_data
modele.predict(new_data)

# Importation des fonctions liées à l'évaluation des métriques
# Pour la détermination de la MSE et du coefficient de dé
    termination R2
from sklearn.metrics import mean_squared_error,r2_score
# Affichage des métriques
print(mean_squared_error(Yreel,Ypredit))
print(r2_score(Yreel,Ypredit))
```

Q1. En vous inspirant des questions réalisées dans le TD6, construire, dans la Zone 2, le modèle de la régression linéaire monovariable, et afficher la MSE et le coefficient de détermination de cette modélisation. Conclure quant à la satisfaction du cahier des charges.

Q2. Afin de visualiser la courbe représentative du modèle linéaire, compléter la Zone 3 pour procéder au tracé.

Seconde modélisation - régression polynomiale de degré 4

Dans cette partie, on souhaite déterminer un modèle par régression polynomiale de degré 4. Autrement dit, on souhaite déterminer les coefficients a_i pour $i \in [0, 4]$ tels que :

$$y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + a_4 \cdot x^4$$

En fait, la détermination de ces paramètres optimaux (au sens qui minimisent la somme des carrés des écarts entre les données expérimentales et le modèle recherché) est basée sur les mêmes outils que la régression linéaire multivariable. En effet, en considérant que les entrées (caractéristiques) de la régression linéaire sont respectivement x, x^2, x^3 et x^4, alors, le travail est similaire à celui d'une régression linéaire multivariable.
Par conséquent, 2 approches sont possibles :

- **Approche 1 :** Créer artificiellement une base de données dont les colonnes sont x, x^2, x^3 et x^4, et alors, le travail est identique à l'étude précédente avec le module `LinearRegression` ;
- **Approche 2 :** Demander à Python de faire le travail à notre place par l'utilisation du module `PolynomialFeatures` de `sklearn.processing` (qui calcule les puissances de la grandeur d'entrée), **associé** au module `LinearRegression`.

C'est l'approche 2 que nous allons utiliser, mais vous pourriez vous amuser à tester avec l'approche 1, et trouver les mêmes résultats.

Le code réalisant ce travail est fourni dans la Zone 4.

Q3. Exécuter le code de la Zone 4, et observer le modèle obtenu par régression polynomiale de degré 4. Compléter cette Zone 4 afin de calculer et afficher la MSE et le coefficient de détermination R^2. Comparer ces valeurs à celles obtenues à la Q1, et au cahier des charges.

Troisième modélisation - régression polynomiale de degré 15

Nous avons observé que la régression polynomiale monovariable de degré 4 est bien plus performante que la régression linéaire monovariable, mais cela était une évidence...

Ah bon, une telle évidence ? ? ? Essayons avec une régression linéaire de degré 15 ! ! !

Q4. En vous inspirant fortement du code de la Zone 4, saisir dans la Zone 5 le code permettant de déterminer, de tracer la régression polynomiale monovariable de degré 15 et les 2 métriques (MSE et R^2). Comparer ces valeurs.

Q5. Qu'observez-vous sur la figure du tracé du polynôme de degré 15 ? Cela vous paraît-il intéressant ? Quel nom donne-t-on à ce phénomène ? Conclure quant au modèle à adopter pour modéliser au mieux le comportement du capteur.