République **A**lgérienne **D**émocratique et **P**opulaire

Ministère de l'**E**nseignement **S**upérieur et de la**R**echerche **S**cientifique

Université des **S**ciences et de la **T**echnologie d'**O**ran **M**ohamed **B**oudiaf

Faculté de Génie Electrique **D**épartement d'Electrotechnique

THÈSE

En vue de l'obtention du

Diplôme de Doctorat en Sciences

Présentée et Soutenue par :

CHAOUCH Djamel Eddine

Intitulé

Contrôle robuste des systèmes dynamiques non linéaires incertains par des approches de l'intelligence artificielle

---

Spécialité *: Electrotechnique*
Option *: Automatique*

Le jury est composé de :

Professeur	***Abdelhamid MIDOUN***	***Président***	***USTO-MB***
Professeur	***Zoubir AHMED FOITIH***	***Rapporteur***	***USTO-MB***
Professeur	***Mohmed Fyaçal KHELFI***	***Co-Rapporteur***	***Univ-Oran1***
Professeur	***Abdelhafid OMARI***	***Examinateur***	***USTO-MB***
Professeur	***Abdelhafid HAFFAF***	***Examinateur***	***Univ-Oran1***
Professeur	***Mokhtar ZERIKAT***	***Examinateur***	***ENP-Oran***

Année Universitaire 2015/ 2016

REMERCIEMENTS

Je tiens à exprimer d'abord ma reconnaissance au Professeur Zoubir AHMED-FOITIH, en tant que directeur de thèse pour son soutien scientifique et humain ainsi que la confiance qu'il m'a témoigné tout au longde ce travail de recherche.

Je remercie sincèrement Professeur Mohamed Fayçal KHELFI, pour avoir codirigé ce travail et pour leurs nombreux conseils ainsi que leur soutien tout au long de cette thèse.

Mes profonds remerciements à Monsieur A. OMARI, Professeur à l'Université d'USTO-MB Oran, à Monsieur A. HAFFAF, Professeur à l'Université d'Oran, et à Monsieur M. ZERIKAT Professeur à l'ENSET d'Oran, pour l'honneur qu'ils m'ont fait en acceptant de juger mon travail et d'être les rapporteurs de cette thèse.

Je tiens également à remercier Monsieur A. MIDOUN, Professeur à l'Université d'USTO-MB Oran, pour l'honneur qu'il m'a fait en acceptant de présider les jurés.

Je remercie très chaleureusement Monsieur Hichem MAAREF, Professeur à l'Université d'Evry Val d'Essone Paris, pour ses conseils et toute l'aide qu'il a pu m'apporter durant cette thèse.

Enfin, je ne saurais terminer ces remerciements sans oublier toute ma famille, sans exception, ainsi que tous mes amis et collègues.

A

La mémoire de mon père, il aurait été si fière
Ma mère, ma source de bénédiction
Ma femme, et mes trois fils
Mes frères et ma sœur
Mes amis et collègues
Mes professeurs

TABLE DE MATIERES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

La linéarisation entrée-sortie a été très utilisée en automatique non linéaire pour trouver une relation directe entre la sortie du système et son entrée afin de mettre en œuvre une loi de commande. Cependant, la complexité et la présence de fortes non linéarités, dans certains cas, ne permettent pas d'avoir une compensation exacte de ces non linéarités et ainsi obtenir les performances de poursuite désirées. De plus, la connaissance du modèle est indispensable; ce qui est généralement très difficile. Pour contourner ce problème, l'approximation du modèle ou de la loi de commande peut être une alternative.

Les réseaux de neurones et les systèmes flous constituent des approximateurs universels. Les points faibles des systèmes flous sont causés essentiellement par la difficulté à définir avec précision les fonctions d'appartenance et le manque de procédure systématique pour la transformation des connaissances d'un expert en une base de règles. Cependant, dans les réseaux neuronaux, l'inclusion, l'extraction et la représentation des connaissances sont difficiles. Ces différentes raisons ont conduit, à partir du début des années 90, à la création d'un domaine de recherche, communément désigné par "neuro-flou", cherchant à fusionner les deux techniques. Il s'agissait de les "combiner" soit de façon à conserver les avantages des deux, tout en évitant les inconvénients, soit en exploitant les analogies entre les deux. De nombreuses recherches ont été effectuées et des solutions très variées ont été proposées. Dans ce contexte, plusieurs commandes adaptatives pour des systèmes non linéaires affines dans la commande ont été présentées dans la littérature où l'approximation est assurée soit par un système flou, soit par un réseau de neurones ou un réseau neuroflou [Takagi, 1985], [Wang, 1994], [Narendra, 1997], [Noriega, 1998], [Delyon, 1995], [Cheng, 1996], [Guerra, 2001], [Labiod, 2005].

Comme dans la commande adaptative classique, on peut distinguer deux cas : direct et indirect. Dans le cas de la loi directe, la commande est approximée par un approximateur mis à jour selon une loi d'adaptation déduite de l'étude de la stabilité. Dans le cas indirect, on approxime d'abord la dynamique du système par deux approximateurs, puis on met en œuvre la loi de commande. Les lois d'adaptation sont également déduites de l'étude de la stabilité au sens de Lyapunov. Cependant, ce type de commandes ne permet pas de maintenir de bonnes performances de poursuite en présence de perturbations externes.

Connu par sa robustesse et sa simplicité de mise en œuvre, le mode glissant a été largement utilisé pour commander une large classe de systèmes non linéaires [Utkin, 1977], [Slotine, 1984]. Il s'agit de définir une surface dite de glissement en fonction des états du système de façon qu'elle soit attractive. La commande globale synthétisée se compose de

deux termes : le premier permet l'approche jusqu'à cette surface et le second le maintien et le glissement le long de celle-ci vers l'origine du plan de phase. La commande globale ainsi construite permet d'assurer en plus des bonnes performances de poursuite, une dynamique rapide et un temps de réponse court. Cependant, cette loi de commande représente quelques inconvénients qui peuvent être résumés en deux points. Le premier réside dans la nécessité d'avoir des informations précises sur l'évolution du système dans l'espace d'état et les bornes supérieures des incertitudes et des perturbations. Or, la nature incertaine des systèmes non linéaires rend difficile, si ce n'est impossible, de disposer d'une description analytique de la dynamique du système. Le second inconvénient réside dans l'utilisation de la fonction signe dans la loi de commande pour assurer le passage de la phase d'approche à celle du glissement. Ceci donne lieu au phénomène de broutement qui consiste en des variations brusques et rapides du signal de commande, ce qui peut exciter les hautes fréquences du processus et l'endommager. Pour remédier à ces problèmes, plusieurs approches ont été présentées dans la littérature [Slotine, 91], [Bartolini, 98], [Lin, 2002], [Barron, 1993], [Yoo, 1998]. En effet, pour le premier inconvénient plusieurs travaux ont été focalisés sur la combinaison des modes glissants avec la commande adaptative où la dynamique du système incertain est approximée à l'aide d'un réseau de neurones, d'un système flou ou une combinaison des deux (réseau neuro-flou) [Wang, 1997], [Yoo, 1998], [Hsu, 2004]. Cependant, quelques problèmes sur la convergence de l'algorithme adaptatif et les conditions de stabilités restent posés. Pour le second inconvénient, l'introduction d'une bande de transition autour de la surface de glissement pour transformer la fonction signe en saturation, peut être une solution [Slotine, 1991]. Néanmoins, une erreur statique subsiste, et un compromis entre la largeur de la bande et les variations de la commande s'impose. D'autres approches utilisant un système flou ont été également proposées dans la littérature [Lin, 2002], [Boukezzoula, 2000].

L'objectif de cette thèse est de développer des commandes combinant les approches de l'intelligence artificielle et la commande robuste pour le contrôle des systèmes dynamiques non-linéaires incertains. En effet, deux méthodes sont proposées, la première combinant le mode glissant avec les réseaux neuro-floues et l'autre est avec les réseaux de neurones.

La thèse est organisée en quatre chapitres :

Le **chapitre 1,** présente un aperçu sur les réseaux de neurones utilisés en automatique, quelques concepts de base ainsi que leurs structures générales. Ensuite, l'utilisation des réseaux de neurones pour l'identification et la commande des systèmes dynamiques non-linéaires.

Le **chapitre 2**, présente un aspect bibliographique allant de la définition des ensembles flous à la synthèse de la commande par l'approche de modélisation neuro-floue en passant par les principales caractéristiques des modèles flous et la structure des contrôleurs correspondants, et terminant par les différentes techniques de commande neuro-floue à savoir la méthode ANFIS, et l'approche STFIS. La commande par réseaux neuro-flou STFIS est adoptée afin de résoudre le problème de poursuite de trajectoire.

Le **chapitre 3,** traite la synthèse de contrôleur mode glissant à base de réseau neuro-flou STFIS en présence des incertitudes paramétriques et des perturbations externes. La dynamique non linéaire est supposée totalement inconnue et l'approximation par le modèle STFIS de type Takagi-Sugeno concerne cette partie inconnue pour générer la commande équivalente. Les performances de poursuite et de stabilité sont obtenues en se basant sur l'approche de Lyapunov.

Le **chapitre 4**, présente deux structures de commande des systèmes MIMO non linéaires, appliquées à la robotique :

- La première est la commande Slotine neuronale, où les estimés du modèle dynamique du robot manipulateur sont calculées par réseaux de neurones.
- La deuxième est la commande robuste 'STFSM', c'est une commande par mode glissant basée sur la technique neuro-floue STFIS utilisée pour chaque articulation du robot.

Enfin, dans une conclusion générale, nous repositionnons l'ensemble de nos développements en regard de l'objectif initial de l'étude : "développer des commandes combinant les approches de l'intelligence artificielle et la commande robuste pour le contrôle des systèmes dynamiques non linéaires incertains". Nous résumons à ce stade les principaux résultats de notre étude. Nous abordons enfin une discussion sur les perspectives de travail qui découlent de cette thèse.

CHAPITRE

I

LES RESEAUX DE NEURONES : IDENTIFICATION DES SYSTEMES NON LINEAIRES

I.1. INTRODUCTION

Pendant des décennies, les scientifiques ont été hantés du rêve de développer des machines intelligentes avec un grand nombre d'éléments simples en s'inspirant des cerveaux humains. Ce vieux rêve a vu le jour pendant les années 1940 quand les chercheurs ont développé des modèles simples des neurones biologiques. McCulloch et Pitts ont posé la première pierre en publiant leur étude systématique du réseau neurologique artificiel.

Et ce n'était que le début bien que les réseaux de neurones aient connu des hauts et des bas dans les années 50 et 60.Cependant, les chercheurs, tels que Minsky et Papert, ont fortement défié ces travaux.

Le paradigme de réseaux de neurones a connu son essor dans les années quatre-vingts avec les extensions de travaux d'éminents chercheurs tels que ceux de Rosenblatt, de Widrow et de Hoff qui traitent l'apprentissage dans un réseau complexe multicouche.

L'intérêt pour les réseaux neurologiques vient de la capacité des réseaux d'imiter le cerveau humain aussi bien que sa capacité d'apprendre et répondre. En conséquence, des réseaux neurologiques ont été employés dans un grand nombre d'applications, et se sont avérés efficaces en exécutant des fonctions complexes dans une variété de champs. Ceux-ci incluent l'identification de modèle, la classification, la commande des systèmes, et la prévision.

Ce chapitre est consacré à la présentation des réseaux de neurones, et à l'utilisation des réseaux de neurones pour l'identification des systèmes dynamiques non-linéaires. L'identification d'un processus consiste à représenter son comportement dynamique à l'aide d'un modèle mathématique paramétré, qui est très souvent une simplification des relations qui existent entre différentes variables du système, mais doit rester représentatif du fonctionnement du processus étudié. La plupart des résultats connus en identification linéaire conventionnelle s'étendent directement à l'identification par réseaux de neurones.

I.2. RESEAUX DE NEURONES FORMELS

I.2.1. Motivations biologiques

Le neurone biologique est une cellule vivante spécialisée dans le traitement des signaux électriques, les neurones sont reliés entre eux par des liaisons appelées axones. Les axones vont eux-mêmes jouer un rôle important dans le comportement logique de l'ensemble en conduisant les signaux électriques de la sortie d'un neurone vers l'entrée (synapse) d'un autre neurone.

Chaque neurone fait une sommation des signaux reçus en entrée, et en fonction du résultat obtenu va fournir un courant en sortie (voir Figure I.1). [Eliasmith, 2002].

- La structure d'un neurone se compose de trois parties:

* La somma, ou cellule d'activité nerveuse, au centre du neurone.

* L'axone attaché au somma qui est électriquement actif; ce dernier conduit l'impulsion conduite par le neurone.

* Dendrites, électriquement passives, reçoivent les impulsions d'autres neurones.

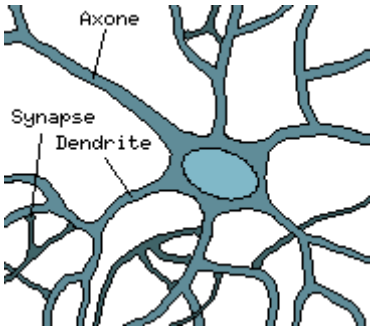
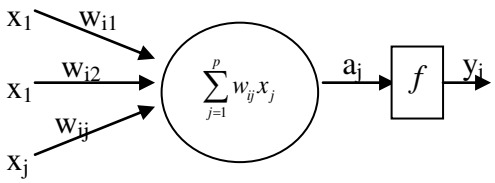


Figure I.1. Neurone biologique

I.2.2. Définitions

Un neurone formel est une fonction algébrique non linéaire et bornée, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelés ''entrées '' du neurone, et la valeur de la fonction est appelée sa sortie [Dreyfus, 2002].

Un neurone est avant tout un opérateur mathématique numérique qui réalise deux opérations. La première est la somme pondérée du neurone par les poids synaptiques ; cette somme est appelée potentiel neuronal. La seconde génère la sortie du neurone, image par une fonction *f* appelée généralement fonction d'activation ou d'évaluation (voir Figure I.2).



$$a_i = \sum_{j \in P_i} w_{ij} x_i \,, \;\; y_i = f(a_i)$$

x_i : paramètres d'entrées.
w_{ij} : poids synaptiques.
a_i : potentiel neuronal.
y_i : la sortie.
P_i : l'ensemble des indices des entrées du neurone *i*.
f : la fonction d'activation.

Figure I.2. Neurone formel

La fonction d'activation peut être:

-La fonction sigmoïde

$$f(a) = \frac{1}{1+\mathrm{e}^{-Ta}} \tag{I.1}$$

-Ou la tangente hyperbolique:

$$f_T(a) = th(a/T) \tag{I.2}$$

Le paramètre T, strictement positif, permet d'ajuster la pente de l'activation. Nous l'appelons la température.

Un réseau de neurones formels est un système d'opérateurs non linéaires interconnectés, recevant des signaux de l'extérieur par ses entrées et délivrant des signaux de sortie, qui sont les activités de certains neurones. Ces signaux d'entrées et de sortie sont constitués de suites numériques. Un réseau de neurone est donc considéré comme un filtre non linéaire à temps discret.

I.2.3.Structures de connexions des neurones

L'intérêt des neurones formels réside essentiellement dans les propriétés qui résultent de leur association en réseaux, c'est-à-dire de la composition des fonctions non linéaires réalisées par chacun des neurones. Le réseau de neurones formels est constitué de deux types de neurones: les entrées du réseau et les neurones eux-mêmes. Chaque neurone (déterministe) est un processeur non linéaire (généralement simulé sur ordinateur, parfois réalisé sous forme de circuit électronique). Un réseau de neurones est conçu pour remplir une tâche que le concepteur définit par un ensemble de valeurs d'entrée, et par un ensemble de valeurs désirées correspondantes pour les activités de certains neurones du réseau appelés neurones de sortie (les éléments de ces ensembles sont appelés "exemples d'apprentissage"). Les neurones qui ne sont pas des neurones de sortie sont dits cachés. Il existe deux types de réseaux de neurones: les réseaux non bouclés et les réseaux bouclés [Dreyfus, 2002].

I.2.3.1. Les réseaux de neurones non bouclés

Un réseau de neurones non bouclé est donc représenté graphiquement par un ensemble de neurones connectés entre eux, l'information circulant des entrées vers les sorties sans retour en

arrière. Le réseau est représenté par un graphe acyclique dont les nœuds sont les neurones et les arêtes les connexions entre ceux-ci, si on se déplace dans le réseau, à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ [Dreyfus, 2002].

I.2.3.1.a. Perceptrons multicouches

Le perceptron multicouche (PMC) est un réseau de simples neurones appelés perceptrons. Le principe de base du perceptron a été introduit par Rosenblatt en 1958. La sortie du perceptron est une fonction non-linéaire d'une combinaison linéaire de ses entrées réelles et les poids. Ceci peut être écrit [Hagan, 1996]:

$$y = f(\sum_{i=1}^{n} w_i x_i + b) = f(w^T x + b) \tag{I.3}$$

Où $\boldsymbol{w}$ représente le vecteur des poids, $\boldsymbol{x}$ est le vecteur des entrées, $\boldsymbol{b}$ est le biais et $\boldsymbol{f}$ la fonction d'activation. Une représentation du perceptron est donnée en Figure I.3.

Le perceptron original de Rosenblatt's utilise une fonction de Heaviside comme fonction d'activation. Actuellement, et spécialement dans les perceptrons multicouches, la fonction d'activation est souvent choisie parmi la fonction sigmoïde ou la fonction tangente hyperbolique. Ces fonctions sont préférées pour leurs comportements mathématiques; leurs dérivées sont faciles à obtenir et ressemblent aux fonctions linéaires près de l'origine, et se saturent rapidement en s'en éloignant.

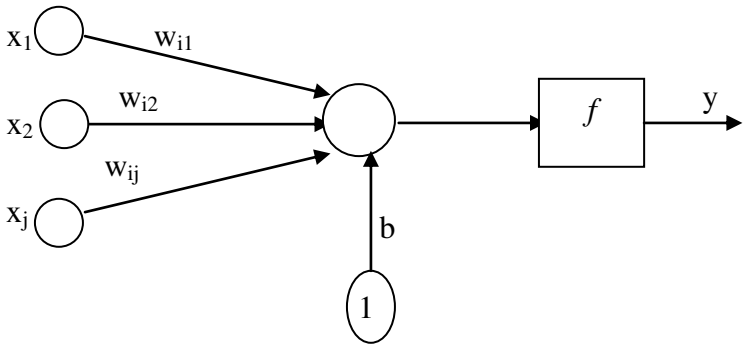


Figure I.3. Le perceptron

Un simple perceptron ne peut discriminer que les ensembles linéairement séparables, quel que soit la fonction d'activation. Cette limite est remédiée par la combinaison de perceptrons pour construire un réseau.

Dans cette architecture, les neurones sont organisés en couches, comme il est montré dans la Figure I.4. Le réseau contient une couche d'entrées notée *r*, une ou plusieurs couches intermédiaires h_1, h_2,…,h_l entre les entrées et les sorties, appelées couches cachées et un neurone (ou une couche de neurones) de sortie. Les connexions se font d'une couche à la suivante sans qu'il y ait de connexion entre couches non adjacentes. Chaque connexion reliant une cellule *i* à une cellule *j* est notée w_{ij}^h si elle appartient à la couche k du réseau; la sortie de la cellule *i* de la couche *k* est notée $s_i^h = f_i(a_i^h)$, et quand *k*=0 (le neurone est une entrée) nous avons $s_i^0 = x_i$, et quand il s'agit de la couche de sortie, nous avons $s_i^k = a_i$.

Nous appellerons activation du neurone *i* de la couche *k* la quantité:

$$\forall h, 1 \le k \le l+1,\ a_i^k = \sum_{j=1}^{N_h} w_{ij}^{k-1} s_j^{k-1} + b_i \qquad \text{(I.4)}$$

Où N_h est le nombre du neurone de la couche k-1.

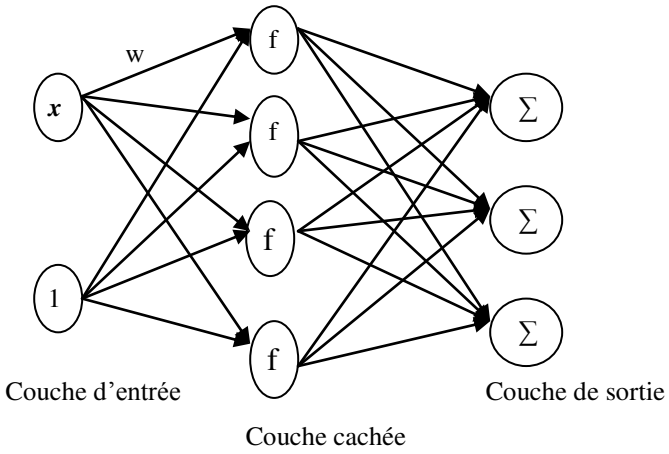


Figure I.4. Le perceptron multicouche

I.2.3.1.b. Réseau à fonction radiale

Les réseaux à fonction radiale (RBF) qui possèdent deux couches forment une classe particulière de réseaux multicouches. Chaque cellule de la couche cachée utilise une fonction noyau (kernel function) telle que la gaussienne en tant que fonction d'activation. Cette fonction est centrée au point spécifié par le vecteur de poids associé à la cellule. La position et la 'largeur' de ces courbes sont apprises. Il y a, en général, beaucoup moins de fonctions noyaux dans un réseau RBF que de patrons d'entrée. Chaque cellule de sortie implémente une combinaison

linéaire de ces fonctions; l'idée étant d'approximer une fonction par un ensemble de fonctions. De ce point de vue, les cellules cachées fournissent un ensemble de fonctions qui forment une base représentant les patrons d'entrées dans l'espace "couvert" par les cellules cachées.

I.2.3.2 Les réseaux de neurones bouclés

L'architecture la plus générale pour un réseau de neurones, «réseaux bouclés», a un graphe de connexions cyclique: lorsque nous nous déplaçons dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ (un tel chemin est désigné sous le terme de «cycle» (Voir Figure I.5). La sortie d'une cellule du réseau peut donc être fonction d'elle-même; cela n'est évidemment concevable que si la notion de temps est explicitement prise en considération [Medsker, 2001]**.**

Ainsi, à chaque connexion d'un réseau de neurones bouclé, outre un poids comme pour les réseaux non bouclés, est attaché un retard, multiple entier (éventuellement nul) de l'unité de temps choisie. Une grandeur, à un instant donné, ne pouvant pas être fonction de sa propre valeur au même instant, tout cycle du graphe du réseau doit avoir un retard non nul. [Aussem, 1995]. Dans cette catégorie en cite par exemple : Cartes auto organisatrices; Réseau de Hopfield….etc.

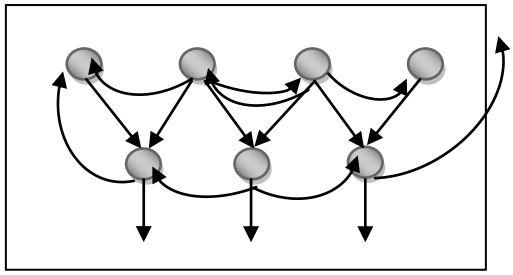


Figure I.5. Réseau à connexions récurrentes

I.3. PROPRIETE DES RESEAUX DE NEURONES

Les réseaux de neurones à couches, présentés au paragraphe précédent, ont la propriété générale d'être des approximateurs universels parcimonieux. Il s'agit en fait de deux propriétés distinctes détaillées ci-dessous [Hagan, 1995].

I.3.1 Propriétés d'approximation des réseaux de neurones

Les travaux de Cybenko [Cybenko, 1989] et Funahachi [Funahashi, 1989] ont prouvé la possibilité d'approcher des fonctions continues. Ces résultats affirment donc, pour toute fonction déterministe usuelle, l'existence d'une approximation par un réseau de neurones [Dreyfus, 2002].

I.3.2. La propriété de parcimonie

Lorsque nous cherchons à modéliser un processus à partir des données, nous nous efforçons toujours d'obtenir les résultats les plus satisfaisants possibles avec un nombre minimum de paramètres ajustables.

Ceci est obtenu en utilisant des réseaux de neurones à fonction d'activation sigmoïdale puisque la sortie de ces neurones n'est pas linéaire par rapport aux poids synaptiques. Cette propriété montre l'intérêt des réseaux de neurones par rapport à d'autres approximateurs comme les polynômes dont la sortie est une fonction linéaire des paramètres ajustables: pour un même nombre d'entrées, le nombre de paramètres ajustables à déterminer est plus faible pour un réseau de neurones que pour un polynôme. Cette propriété devient d'autant plus intéressante dans le cas du filtrage de textes car le nombre d'entrées est typiquement de l'ordre de plusieurs dizaines.

I.4. APPRENTISSAGE DES RESEAUX DE NEURONES

I.4.1. Définition

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux neuronaux. L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. L'apprentissage neuronal fait appel à des exemples de comportement.

Durant cette phase de fonctionnement, le réseau adapte sa structure (le plus souvent, les poids des connexions) afin de fournir sur ses neurones de sortie les valeurs désirées. Cet apprentissage nécessite des exemples désignés aussi sous l'appellation d'échantillon d'apprentissage ainsi qu'un algorithme d'apprentissage. Après initialisation des poids du réseau (en général des valeurs aléatoires), il y a présentation des exemples au réseau et calcul des sorties correspondantes. Une valeur d'erreur ou de correction est calculée, et une correction des poids est appliquée.

Au niveau des algorithmes d'apprentissage, il a été défini deux grandes classes selon que l'apprentissage est dit supervisé ou non supervisé. Cette distinction repose sur la forme des exemples d'apprentissage. Dans le cas de l'apprentissage supervisé, les exemples sont des couples (Entrée, Sortie associée) alors que nous ne disposons que des valeurs (Entrée) pour l'apprentissage non supervisé.

I.4.2. Règles d'apprentissage

Les règles d'apprentissage peuvent être classées en trois catégories:

- La première regroupe les règles d'apprentissage supervisé. Elles s'appliquent dans le cas où les valeurs désirées en sortie du réseau sont quantifiables. Les poids synaptiques sont ceux qui optimisent un critère de performance défini par une fonction coût (voir Figure I.6).

- La seconde catégorie réunit les règles d'apprentissage par renforcement: l'ajustement des poids est guidé par un critique qui estime la qualité du réseau à réaliser l'objectif fixé. L'apprentissage n'est pas guidé par la connaissance des grandeurs désirées, son efficacité dépend fortement de la pertinence du critique. L'algorithme d'apprentissage par pénalité et récompense, est de type stochastique et privilégie les variations des poids qui conduisent au comportement souhaité du réseau. Une application originale de cette technique a été proposée pour faire apprendre à un robot hexapode la marche et l'évitement d'obstacles.

- Une troisième catégorie concerne les règles d'apprentissage non supervisé qui ne font appel à aucun professeur. Elles sont préconisées lorsqu'il est difficile de quantifier les valeurs que doivent prendre les sorties du réseau. Nous pouvons citer comme exemples les réseaux auto-organisateurs de Kohonen. Les propriétés du réseau sont construites au fur et à mesure de la présentation des données sans qu'un objectif ne soit défini à priori.

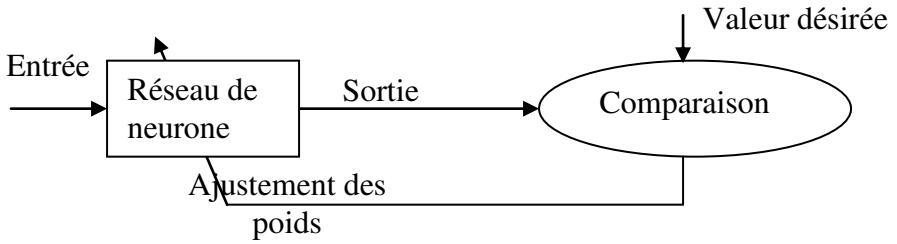


Figure I.6. Apprentissage supervisé

I.4.3. Choix de la fonction coût

Le choix de la fonction coût est conditionné par l'objectif à atteindre. Pour les problèmes de régression, l'ensemble d'apprentissage est constitué d'exemples pour lesquels la sortie désirée y_{di}^p est une variable continue. La fonction coût la plus utilisée est l'erreur quadratique sur la base d'apprentissage: elle consiste à minimiser la somme des carrés des erreurs entre la sortie du réseau et la valeur réelle de la sortie.

En notant y_i^p la valeur de la sortie *i* calculée par le réseau pour l'exemple *p*, la fonction coût devient

$$E(w) = \sum_p E_p = \sum_p \sum_{i=1}^{N_{l+1}} \frac{1}{2} (y_{di}^p - y_i^p)^T (y_{di}^p - y_i^p) \tag{I.5}$$

Cette fonction coût est issue du principe de maximum de vraisemblance avec une hypothèse gaussienne sur la distribution des sorties.

I.4.4. Algorithme d'apprentissage

I.4.4.1. Principe des algorithmes

Soit E*(w)* la fonction coût (I.5). Les algorithmes utilisés nécessitent que E*(w)* soit dérivable par rapport aux poids. Le principe de ces méthodes est de se placer en un point initial, de trouver une direction de descente du coût dans l'espace des paramètres *w*, puis de se déplacer d'un pas dans cette direction. Nous atteignons un nouveau point, et nous itérons la procédure jusqu'à satisfaction d'un critère d'arrêt. Ainsi, à l'itération *k*, Nous calculons:

$$w_{k+1} = w_k + \Delta w_k = w_k + \eta_k p_k \tag{I.6}$$

p_k est le pas de la descente et η est la direction de descente: les différents algorithmes se distinguent par le choix de ces deux quantités.

I.4.4.2. Descente du gradient

L'objectif de l'algorithme est de minimiser la fonction coût (I.5), en d'autres termes la fonction E(*w*) doit décroître à chaque itération:

$$E(w_{k+1}) < E(w_k) \tag{I.7}$$

Considérant le développement de série de Taylor du premier ordre de E(w) au voisinage de w_k:

$$\begin{aligned} E(w_{k+1}) &= E(w_k + \Delta w_k) = E(w_k) + \nabla_k \Delta w_k \\ E(w_{k+1}) - E(w_k) &= \nabla_k \Delta w_k \end{aligned} \tag{I.8}$$

Où ∇_k est le gradient évalué pour w_k. Pour s'assurer que E(w_{k+1}) soit inférieur à E(w_k), le second terme de la partie droite de (I.8) doit être négatif:

$$\nabla_k \Delta w_k = \eta_k \nabla_k p_k < 0 \tag{I.9}$$

Et puisque le pas d'apprentissage est petit mais supérieur à zéro, alors:

$$\nabla_k p_k < 0 \tag{I.10}$$

N'importe vecteur p_k qui satisfait cette condition est appelé direction de descente. La fonction va décroître si nous prenons un pas assez petit dans cette direction. Ce qui nous ramène à une autre question: dans quelle direction la descente est plus rapide? Ceci intervient lorsque $\nabla_k p_k$ est le plus négatif, *i.e.* lorsque la direction est à l'opposé du gradient et de même valeur:

$$p_k = -\nabla_k \tag{I.11}$$

Remplacé dans (I.6) nous aurons:

$$\begin{aligned} w_{k+1} &= w_k - \eta_k \nabla_k \\ \Delta w_k &= -\eta_k \frac{\partial E(w)}{\partial w_k} \end{aligned} \tag{I.12}$$

L'algorithme consiste donc à choisir comme direction de descente l'opposé du gradient de la fonction coût. Cette méthode est efficace loin du minimum et permet uniquement de s'en approcher. Pour cette raison, la détermination du pas n'est pas cruciale: loin du minimum, il faut seulement vérifier que le pas n'est ni trop petit ni trop grand.

I.4.4.3. Algorithme de rétro-propagation

Un algorithme d'apprentissage supervisé particulièrement connu est l'algorithme de rétro-propagation. Il permet de calculer la dérivée d'une fonction coût par rapport à un poids synaptique quelconque. Chacun des poids est alors réajusté selon une règle de descente de gradient. Son application suppose cependant la dérivabilité de la fonction coût. L'algorithme peut être considéré comme un algorithme de gradient appliqué à un problème d'optimisation non linéaire. Les apports décisifs de Rumelhart et McCulland concernent l'adaptation de cet algorithme pour réajuster les poids synaptiques des neurones des couches cachées dans un réseau multicouche. Le terme rétro-propagation du gradient provient du fait que l'erreur calculée en sortie est transmise en sens inverse vers l'entrée.

Le principe de l'algorithme est de minimiser une fonction d'erreur. Il s'agit ensuite de calculer la contribution à cette erreur de chacun des poids synaptiques des couches précédentes, étape qui est difficile. En effet, chacun des poids influe sur le neurone correspondant, mais, la modification pour ce neurone va influer sur tous les neurones des couches suivantes.

La modification des poids w_{ij} pour l'itération k:

$$\Delta w_{ij} = -\eta \Lambda_k \quad \text{où} \quad \Delta_k = \frac{\partial E_k}{\partial w_{ij}} \tag{I.13}$$

Il nous suffit donc de calculer $\partial E/\partial w_i$, pour cela, nous distinguons deux cas: le cas où le neurone i est un neurone de sortie et le cas où c'est un neurone caché.

Le neurone i est un neurone de sortie, nous avons:

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}}$$
$$\frac{\partial E_k}{\partial s_i} = -(y_{di}^k - y_i^k) \tag{I.14}$$

Et:

$$\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial s_i}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}} = f_i'(a_i)s_j^l \tag{I.15}$$

A partir de (I.14), (I.15), nous aurons:

$$\frac{\partial E_k}{\partial w_{ij}} = -(y_{di}^k - y_i^k).f_i^{'}(a_i).s_j^l \tag{I.16}$$

Cas où la cellule est un neurone caché: Il s'agit de calculer la modification des poids des w_{ij} appartenant à la couche cachée *c*. Nous allons donc exprimer E_p en fonction de w_{ij}^c, en utilisant la composition des dérivations. Nous évaluons la fonction pour ***c***=1 avant de généraliser avec toutes couches ***c***. Il est noté que w_{ij}^l est utilisé pour calculer a_i^l qui permet de calculer s_i^l .

$$\frac{\partial E_k}{\partial w_{ij}^l} = \sum_{m=1}^{N_{l+1}} \frac{\partial E_k}{\partial s_m^{l+1}} \cdot \frac{\partial s_m^{l+1}}{\partial a_m^{l+1}} \cdot \frac{\partial a_m^{l+1}}{\partial s_m^l} \cdot \frac{\partial s_m^l}{\partial a_i^l} \cdot \frac{\partial a_i^l}{\partial w_{ij}^l} \tag{I.17}$$

En calculant chaque terme de l'équation (I.17), nous obtenons:

$$\frac{\partial E_k}{\partial w_{ij}^l} = -(y_k^p - a_k).f_k^{'}(a_k^{l+1}).w_{k,i}^l.f_i^{'}(a_i^l).s_j^{l-1} \tag{I.18}$$

Cet algorithme, qui présente l'avantage d'exister, reste discutable dans la mesure où sa convergence n'est pas prouvée. Son utilisation peut conduire à des blocages dans un minimum local de la surface d'erreur. Son efficacité dépend, en effet, d'un grand nombre de paramètres que doit fixer l'utilisateur: le pas du gradient, les paramètres des fonctions sigmoïdes, l'architecture du réseau, nombre de couches, nombre de neurones par couche.

La valeur du taux d'apprentissage μ a un effet significatif sur les performances du réseau; si ce taux est petit l'algorithme converge lentement, par contre s'il est grand l'algorithme risque de générer des oscillations. Généralement, μ doit être compris entre 0 et 1 pour assurer la convergence de l'algorithme vers une solution optimale. Il n'existe pas de règles permettant de déterminer le nombre de couches cachées dans un réseau donné ni le nombre de neurones dans chacune d'elles.

Théoriquement, l'algorithme doit s'arrêter dès que le minimum de l'erreur commise par le réseau sera atteint, correspondant à un gradient nul, ce qui n'est jamais rencontré en pratique. C'est pourquoi un seuil est fixé à priori afin d'arrêter l'apprentissage.

I.4.5.L'apprentissage

Nous considérons le système non-linéaire discret décrit par modèle général suivant:

$$\mathrm{y}(k) = g_{nn}(\varphi(k), \theta) + e(k) \tag{I.19}$$

Où, pour chaque observation k, y(k) est la sortie du système, e(k) est un bruit aditif de moyenne nulle, g est une fonction non-linéaire arbitraire, θ est le vecteur des paramètres du modèle, et $\varphi(k)$ le vecteur d'information défini tel que:

$$\varphi(k) = [x_1(k)...x_i(k)...x_{n_0}(k)]^T \tag{I.20}$$

Où $x_i(k)$, i=1…n_0 sont les n_0 entrées du système pour la $k^{\text{ième}}$ observation.

Le but de la modélisation par réseau de neurones est de trouver une structure paramétrée notée NN, qui approxime la fonction non-linéaire g à partir d'une base de données d'apprentissage:

$$Z^N \left\{ \left[y(k), \varphi(k) \right] | \, k = 1,..., N \right\} \tag{I.21}$$

Le vecteur d'information $\varphi(k)$ étant présenté aux entrées $\left\{ x_1^0,.., x_{n_0}^0 \right\}$ de la première couche, le prédicteur neuronal peut être écrit comme (voir Figure I.7):

$$\tilde{\mathrm{y}}(k, \theta) = g_{nn}(\varphi(k), \theta) \tag{I.22}$$

Où θ comprend tous les poids et biais inconnus du réseau de neurones.

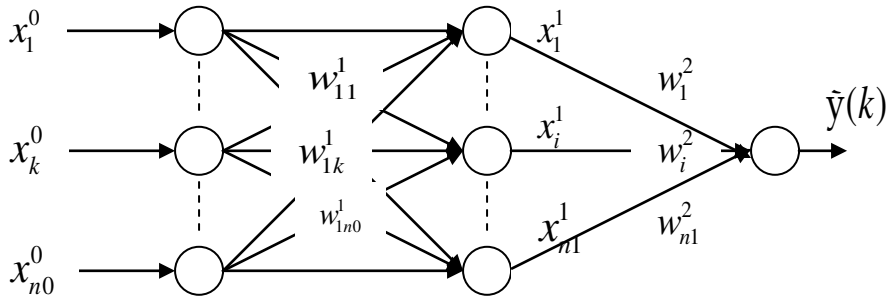


Figure I.7. Prédicteur neuronal

L'erreur de prédiction peut alors être introduite:

$$\varepsilon(k, \theta) = y(k) - \tilde{\mathrm{y}}(k, \theta) \tag{I.23}$$

Et le critère suivant peut être écrit:

$$V_N(\theta, Z^N) = \frac{1}{N}\sum_{k=1}^{N} L(\varepsilon(k,\theta)) \tag{I.24}$$

Où L est une fonction scalaire également appelée fonction coût par observation. Les paramètres sont obtenus par minimisation du critère (I.24).

$$\hat{\theta} = \arg\min_{\theta} V_N(\theta, Z^N) \tag{I.25}$$

L'estimation $\hat{\theta}^{(i)}$ de θ est définie par la minimisation du critère $V_N(\theta, Z^N)$ à l'itération *i*.

I.4.5.1 La méthode de Newton

La méthode de Newton est basée sur les séries de Taylor de deuxième ordre [Kelley, 1999]. La modification des poids en utilisant la méthode de Newton est donnée par:

$$\theta^{(i+1)} = \theta^{(i)} - H_i^{-1}\nabla_i^T(\theta^{(i)}) \tag{I.26}$$

Cette méthode converge en une seule itération pour une fonction quadratique. Elle est inefficace loin du minimum de la fonction et très efficace près du minimum. Si la fonction coût n'est pas quadratique, la méthode ne converge pas généralement en une seule itération.

I.4.5.2 La méthode de quasi-Newton

Les méthodes de quasi-Newton sont une généralisation de la méthode de Newton; elles consistent à approcher l'inverse du Hessien plutôt que de calculer sa valeur exacte. Nous allons présenter deux de ces méthodes: l'algorithme BFGS et celui de Levenberg-Marquardt [Kelley, 1999].

I.4.5.2.a L'algorithme BFGS

L'algorithme de BFGS appartient aux méthodes d'optimisation de quasi–Newton. L'algorithme ajuste les paramètres par la formule suivante :

$$\theta^{(i+1)} = \theta^{(i)} - \eta_i M_i \nabla_i V_N(\theta^{(i)}, Z^N) \tag{I.27}$$

Où M_i est une approximation calculée itérativement pour l'inverse du Hessien.

I.4.5.2.b Levenberg–Marquardt algorithm (LM).

Comme les méthodes quasi-Newton, l'algorithme de Levenberg–Marquardt est basé sur l'approximation du Hessien plutôt que calculer sa valeur. Quand la fonction coût est quadratique (ce qui est le cas pour les réseaux feed-forward), l'Hessien est approché par [Marquardt, 1963 ; Kelley, 1999]

$$H(\theta^{(i)}) \approx J^T(\theta^{(i)})J(\theta^{(i)}) \tag{I.28}$$

Et le gradient est donné par:

$$\nabla_k^T(\theta^k) = J^T(\theta^k)E(\theta^k) \tag{I.29}$$

Où J est la matrice Jacobienne contenant les dérivées premières de la fonction coût par rapport à ses poids et biais. Le Jacobien peut être calculé plus facilement que le Hessien.

L'algorithme repose sur la formule de modification suivante**:**

$$\theta^{(i+1)} = \theta^{(i)} - \left[H(\theta^{(i)}) + \eta_i I \right]^{-1} \nabla_i V_N(\theta^{(i)}, Z^N) \tag{I.30}$$

$$\theta^{(i+1)} = \theta^{(i)} - \left[J^T(\theta^{(i)})J(\theta^{(i)}) + \eta_i I \right]^{-1} J^T(\theta^{(i)})V_N(\theta^{(i)}, Z^N) \tag{I.31}$$

où $H(\theta^{(i)})$ est le Hessien de la fonction coût et η_k est le pas d'apprentissage.

Pour des pas d'apprentissage trop petits, La méthode de Levenberg –Marquardt approche celle de Newton, et pour des valeurs plus grandes elle est équivalente à l'application du simple algorithme de descente de gradient avec un pas d'apprentissage de $1/\eta_k$.

I.4.5.3. Concept de généralisation

Une fois l'apprentissage des paramètres terminé, la sortie du modèle ne reproduira toujours pas à l'identique la sortie du système et il existe toujours une erreur de modélisation. Deux raisons sont à l'origine de cette erreur [Norgaard, 1996]. Premièrement, la structure du modèle ne correspond généralement pas à la structure exacte du système. Cette erreur est appelée erreur de biais. Nous savons notamment qu'un réseau de neurones est capable d'approcher toute fonction non linéaire avec une précision arbitraire. Cette précision dépend du nombre de paramètres constituant le modèle. Plus nombreux sont les paramètres introduits dans le modèle, plus l'erreur de biais sera faible, il est cependant évident que cette erreur est inévitable. La seconde origine de

l'erreur de modélisation est due au fait que le jeu de donnée d'apprentissage est bruité et de taille limitée. Ceci implique que les paramètres obtenus lors de l'apprentissage seront différents des paramètres optimaux, cette erreur est appelée erreur de variance.

Si l'erreur de biais diminue avec l'accroissement de la taille du réseau, il n'en est pas le cas pour l'erreur de variance qui, elle, a plutôt tendance à s'accroître. Il est nécessaire donc, de trouver un compromis entre ces deux sources d'erreur lors du choix de la structure du modèle.

I.4.5.4 Les méthodes de régularisation

Les méthodes de régularisation ne cherchent pas à limiter la complexité du réseau, mais elles contrôlent la valeur des poids pendant l'apprentissage. Il devient possible d'utiliser des modèles avec un nombre élevé de poids et donc un modèle complexe, même si le nombre d'exemples d'apprentissage est faible. [Bartlett, 1994] a montré que la valeur des poids était plus importante que leur nombre afin d'obtenir des modèles qui ne sont pas sur-ajustés. Il montre que, si un grand réseau est utilisé et que l'algorithme d'apprentissage trouve une "erreur quadratique moyenne" faible avec des poids de valeurs absolues faibles, alors les performances en généralisation dépendent de la taille des poids plutôt que de leur nombre.

Plusieurs méthodes de régularisation existent dans la littérature, comme *l'arrêt prématuré* (*early stopping*) qui consiste à arrêter l'apprentissage avant la convergence ou les méthodes de pénalisation qui ajoutent un terme supplémentaire à la fonction coût usuelle afin de favoriser les fonctions régulières:

$$V_N^1(\theta^{(i)}, Z^N) = V_N(\theta^{(i)}, Z^N) + \alpha\Omega \tag{I.32}$$

$V_N(\theta^{(i)}, Z^N)$ est une fonction coût, et Ω est une fonction qui favorise les modèles réguliers.

L'apprentissage est réalisé en minimisant la nouvelle fonction $V_N^1(\theta^{(i)}, Z^N)$. Un modèle qui a bien appris la base d'apprentissage correspond à une valeur faible de $V_N(\theta^{(i)}, Z^N)$, alors qu'une fonction régulière correspond à une fonction faible: l'apprentissage doit trouver une solution qui satisfasse ces deux exigences. Parmi les différentes formes possibles pour la fonction Ω, la méthode du '*weight-decay*' et souvent utilisée, car elle est simple à mettre en œuvre, et plusieurs études ont montré qu'elle conduisait à de bons résultats.

I.4.5.4.a Arrêt prématuré

Comme nous l'avons vu précédemment, l'apprentissage consiste à minimiser, grâce à un algorithme itératif, une fonction coût calculée sur la base d'apprentissage. La méthode de l'arrêt prématuré (*early stopping*) consiste à arrêter les itérations avant la convergence de l'algorithme. Si la convergence n'est pas menée à son terme, le modèle ne s'ajuste pas trop finement aux données d'apprentissage: le sur-ajustement est limité. Pour mettre en œuvre cette méthode, il faut déterminer le nombre d'itérations à utiliser pendant l'apprentissage. La méthode la plus classique consiste à suivre l'évolution de la fonction coût sur une base de validation, et à arrêter les itérations lorsque le coût calculé sur cette base commence à croître. Cependant, cette méthode peut être inapplicable, car il est difficile de déterminer avec précision le moment exact où il faut arrêter l'apprentissage puisque les performances sur la base de validation ne se dégradent pas nettement.

I.4.5.4.b Régularisation par modération des poids (Weight-Decay)

Lorsque les poids du réseau sont grands en valeur absolue, les sigmoïdes des neurones cachés sont saturés, si bien que les fonctions modélisées peuvent avoir des variations brusques. Pour obtenir des fonctions régulières, il faut travailler avec la partie linéaire des sigmoïdes, ce qui implique d'avoir des poids dont la valeur absolue est faible. La méthode de régularisation du weight-decay limite la valeur absolue des poids en utilisant:

$$\Omega = \sum_{i=1}^{p} \theta_i^2 \tag{I.33}$$

L'apprentissage s'effectue en minimisant:

$$V_N^1(\theta^{(i)}, Z^N) = V_N(\theta^{(i)}, Z^N) + \frac{\alpha}{2} \sum_{i=1}^{p} \theta_i^2 \tag{I.34}$$

Où p est le nombre de poids que comporte le réseau.

Cette méthode est appelée 'ridge regression' dans le cas des modèles linéaires par rapport aux paramètres [Saporta, 1990].

α est un hyper-paramètre qui détermine l'importance relative des deux termes dans la nouvelle fonction coût. Si α est trop grand, les poids tendent rapidement vers zéro, le modèle ne tient plus compte des données. Si α est trop petit, le terme de régularisation perd de son

importance et le réseau de neurones peut être sur-ajusté. Dans le cas intermédiaire, les poids après l'apprentissage ont des valeurs modérées.

Cette méthode présente l'avantage d'être très simple à mettre en œuvre, puisque le gradient de $V_N(\theta^{(i)}, Z^N)$ se calcule très simplement à partir du gradient de $V_N(\theta^{(i)}, Z^N)$ et du vecteur des poids du réseau θ :

$$\nabla V_N^1(\theta^{(i)}, Z^N) = \nabla V_N(\theta^{(i)}, Z^N) + \alpha\theta \tag{I.35}$$

Il suffit d'ajouter la quantité $\alpha\theta$ au vecteur $\nabla V_N(\theta^{(i)}, Z^N)$ calculé par l'algorithme de rétro-propagation.

En pratique, pour tenir compte du caractère différent des poids en fonction des couches, il faut considérer plusieurs hyper-paramètres [MacKay, 1992]:

$$V_N^1(\theta^{(i)}, Z^N) = V_N(\theta^{(i)}, Z^N) + \frac{\alpha_1}{2}\sum_{i\in\theta_0}\theta_i^2 + \frac{\alpha_2}{2}\sum_{i\in\theta_1}\theta_i^2 + \frac{\alpha_3}{2}\sum_{i\in\theta_2}\theta_i^2 \tag{I.36}$$

θ_0 représente l'ensemble des poids reliant les biais aux neurones cachés, θ_1 représente l'ensemble des poids reliant les entrées aux neurones cachés, et θ_2 représente l'ensemble des poids reliés aux neurones de sortie (y compris le biais du neurone de sortie).

Il convient donc de déterminer les valeurs des trois hyper-paramètres α_1 , α_2 , α_3. Dans ce but, MacKay [McKay 1992] propose une démarche fondée statistiquement d'une manière solide, mais qui repose sur de nombreuses hypothèses et conduit à des calculs lourds. En pratique, il apparaît que les valeurs de ces hyper-paramètres ne sont pas critiques: une démarche heuristique, qui consiste à effectuer plusieurs apprentissages avec des valeurs différentes des paramètres, à tester les modèles obtenus sur un ensemble des données de validation, et à choisir le meilleur, est généralement suffisante.

I.4.5.6 Critères d'arrêt d'apprentissage

Pour arrêter l'apprentissage automatiquement, il est préférable de posséder des critères d'arrêt. Un critère d'arrêt est typiquement une variable scalaire qui indique que rien n'est significativement gagné en continuant l'apprentissage. Une des difficultés dans la sélection d'un critère d'arrêt significatif est que la nature du problème de minimisation nous laisse quelque fois

penser que le réseau est bien appris, alors qu'il ne l'est pas en réalité. Pour cela, il est préférable d'appliquer plusieurs critères en même temps. Les critères les plus utilisés sont:

Nombre maximum d'itérations: l'apprentissage s'arrête quand ce nombre est atteint. Mais il n'est pas raisonnable d'utiliser ce critère quand les poids ont déjà convergé vers la solution désirée.

Seuil du gradient: Au point minimum $\theta=\theta'$ le gradient dans les critères de minimisation devrait s'annuler $\nabla(\theta')=0$. Pour une méthode itérative de minimisation, cela est bien sûr impossible d'être atteint exactement en un nombre fini d'itérations. Nous pouvons choisir que la norme du gradient (par exemple la norme euclidienne) soit inférieure à un certain seuil.

$$\left\|\nabla(\theta)\right\| \le \varepsilon \tag{I.37}$$

Seuil de variation des valeurs des poids: Ce critère consiste à mesurer des poids entre deux itérations. Si la variation maximale des poids, $\max_k \left\{\theta_k^{i+1}-\theta_k^i\right\}$, k=1,..., p, où p est le nombre de paramètres, est inférieure à une certaine valeur, nous pouvons arrêter l'apprentissage.

I.4.5.7 Validation

Le modèle obtenu à partir de l'estimation de ces paramètres n'est valide en toute rigueur, que pour l'expérience utilisée. Il faut donc vérifier qu'il est compatible avec d'autres formes d'entrées, afin de représenter correctement le fonctionnement du processus à identifier. Nous présentons dans ce paragraphe des tests statistiques de validation d'un modèle de prédiction basée sur la fonction d'auto-corrélation des résidus, sur la fonction d'inter-corrélation entre les résidus et les autres entrées du processus.

La plupart de ces tests requièrent un jeu de données qui n'était pas utilisé en apprentissage. Un tel jeu de test ou validation doit si possible couvrir la même gamme que le jeu utilisé pour l'apprentissage.

La plupart des indicateurs de qualité du modèle ont été mis au point pour la validation du modèle et en particulier pour la vérification des hypothèses posées pour le construire. Cependant, leur utilisation pour des modèles neuronaux ne pose aucun problème particulier.

I.5 RESEAUX DE NEURONES POUR LA COMMANDE DE PROCESSUS.

L'automaticien dispose de méthodes et d'outils très puissants pour l'analyse, la modélisation et la commande des systèmes linéaires. De nombreux résultats théoriques ont été établis dans ce domaine, dès lors qu'une description générale des systèmes dynamiques par représentation d'état ait été adoptée et que des outils mathématiques particulièrement efficaces, basés sur l'algèbre linéaire, aient été développés.

Cependant, il n'existe pas, jusqu'à présent, d'outils mathématiques très généraux pour la modélisation ou la commande des systèmes non linéaires. Les propriétés des approximateurs universels et d'apprentissage encouragent l'étude des capacités des réseaux de neurones à reproduire le comportement dynamique des systèmes non linéaires.

Les techniques conventionnelles de modélisation et de commande requièrent une élaboration d'un modèle mathématique du système, ce qui constitue un handicap pour ces techniques pour prouver leur efficacité dans les systèmes dynamiques non-linéaires, surtout quand ceux-ci présentent des complexités élevées et de fortes non linéarités. En effet, l'indisponibilité de connaissances suffisantes nécessaires à la formulation d'un modèle mathématique précis du système est une des causes de cet échec.

Ce manque a dirigé les regards vers les techniques de synthèse par apprentissage, et parmi elles les modèles connexionnistes présentant certaines propriétés intéressantes, faisant d'elles un bon candidat citons [Renders, 1995]:

- Ils ont la capacité attrayante de pouvoir construire des modèles mathématiques à partir de données empiriques historiques.
- Ils ont la qualité théorique d'être de bons approximateurs universels de fonctions continues. Ainsi, ils peuvent modéliser des phénomènes de complexité élevée, présentant de fortes non linéarités.
- Ils sont capables de s'adapter à une dynamique évoluant au cours du temps.
- Ils sont caractérisés par une certaine résistance aux bruits.
- En tant que systèmes multi entrées/multi sorties, ils peuvent être utilisés dans le cadre du contrôle des systèmes multi variables.
- De par leur rapidité de calcul en phase d'exploitation, ils sont adaptés aux applications temps réel.

- De par leur nature distribuée, ils sont facilement mis en place dans des architectures hardware.

I.6. IDENTIFICATION PAR LES RESEAUX DE NEURONES

L'approche itérative adoptée dans ce chapitre est en concordance avec les approches de l'identification des systèmes linéaires que nous trouvons dans [Ljung, 1987]. Elle passe par quatre étapes:

- Choix de la structure du modèle (réseau de neurones).
- Détermination de l'architecture du réseau.
- Estimation des paramètres du réseau (détermination des poids).
- Validation.

Plusieurs modèles ont été présentés dans [Chen, 1997], en s'inspirant des modèles linéaires: les modèles NNARX, NNOE et NNARMAX,

I.7. LA COMMANDE NEURONALE

L'utilisation des réseaux de neurones en commande de processus ne se justifie que dans le cas où il est difficile, voire impossible, de concevoir un système de commande classique. Ces difficultés de conception découlent généralement de la complexité du système dynamique dont il faut assurer la commande: non linéarité, dimensionnalité élevée, objectifs et contraintes de commande multiples, erreurs de modélisation, bruits de mesure, perturbations, pannes [Ronco, 1997].

La classification systématique des différents schémas de commande neuronale proposés dans la littérature, se divise en deux classes: La première utilise les réseaux de neurones comme outil d'aide à l'identification, à la supervision et à l'implémentation des lois de commande**.** La seconde utilise les réseaux de neurones comme contrôleurs (voir figure I.8)

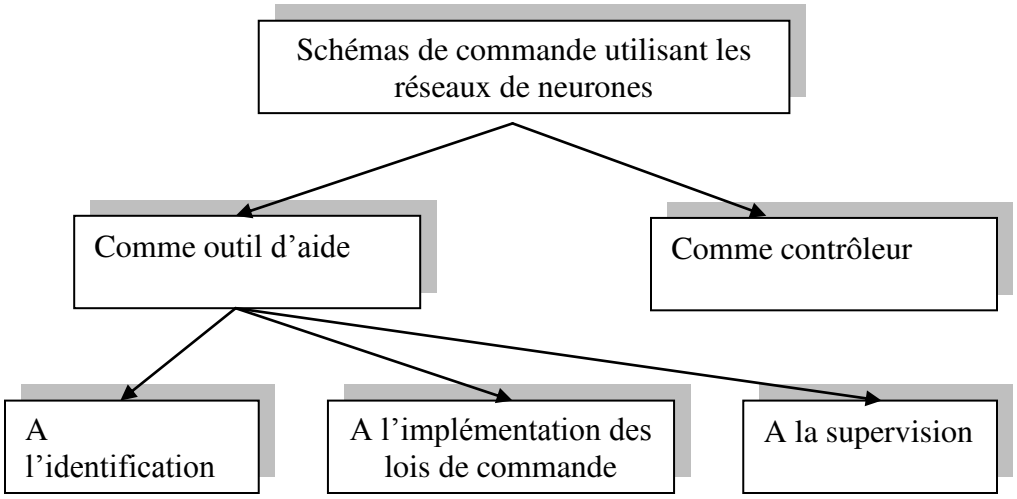


Figure I.8. Classification des différents schémas de commande neuronale

L'utilisation des réseaux de neurones dans les schémas de commande a été traitée par plusieurs auteurs, par exemple [Antsaklis, 1990], [Levin, 1993], [Hunt, 1991] et [Psaltis, 1987], etc. Les principales méthodes de commande par réseau de neurones sont résumées dans [Harris, 1994] et [Rivals, 1995]. Plusieurs classifications peuvent être établies.

Un système de commande est direct, s'il n'utilise pas de modèle du processus à commander pour estimer les paramètres du régulateur. Il est indirect s'il requiert l'utilisation d'un modèle du processus.

Un système de commande peut être non adaptatif: les paramètres du correcteur sont fixés lors d'une phase de synthèse préalable à son utilisation. Il peut être adaptatif: l'apprentissage des paramètres du correcteur est réalisé en ligne; ces paramètres sont ajustés en permanence pendant la commande du système (voir Figure I.9).

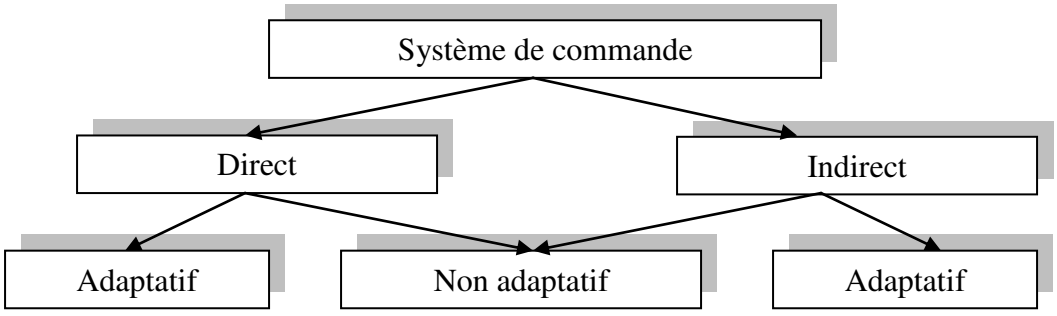


Figure I.9. Classification des systèmes de commande

a) Méthodes de commande de type direct:

_ Commande directe avec modèle inverse [Miller, 1990].

_ Commande adaptative directe [Slotine, 1991].

_ Commande par anticipation [Harris, 1994].

b) Méthodes de commande de type indirect:

_ Commande optimale [White, 1992].

_ Commande adaptative indirecte [Narendra, 1990].

_ Commande avec modèle interne [Sbarbaro, 1993).

_ Commande adaptative avec modèle de référence [Nerrand,1993).

_ Commande prédictive [Grondin, 1994].

_ Commande par linéarisation [Sauner, 1992].

Dans ce chapitre, une synthèse des différentes stratégies de commande neuronale sera développée. L'utilisation d'un même exemple de simulation nous aidera à analyser les caractéristiques et les performances ainsi que les limitations de chaque stratégie.

L'utilisation du modèle inverse d'un processus comme contrôleur est la première architecture qui vient à l'esprit. Pour commencer, il est essentiel de rappeler les différentes méthodes d'apprentissage du modèle inverse d'un processus.

I.7.1. Apprentissage du modèle inverse

Une des premières stratégies de commande neuronale proposée consiste à entraîner un réseau de neurones, à se comporter comme l'inverse du processus et l'utiliser ensuite comme contrôleur. Si le processus à commander peut être exprimé par la relation [Salem, 2007].

$$y(k+1) = f(y(k),..., y(k-n+1), u(k),..., u(k-m+1)) \tag{I.38}$$

Le modèle inverse du système peut être construit en utilisant un réseau de neurones

$$u(k) = f^{-1}(y(k+1), y(k)..., y(k-n+1), u(k-1),..., u(k-m)) \tag{I.39}$$

Ce modèle inverse peut être utilisé pour la commande en remplaçant la sortie réelle du processus à l'instant k+1 par la consigne $y_d(k+1)$. Si le réseau représente l'inverse exact du processus, la commande produite conduira la sortie du processus $y(k+1)$ à suivre exactement la

sortie désirée $y_d(k+1)$. Nous retrouvons cette idée dans les travaux de [Psaltis, 1987] et [Hunt, 1991].

I.7.2. Commande directe par modèle inverse

Comme son nom l'indique, le modèle neuronal inverse placé en amant du système, est utilisé comme contrôleur pour commander le système en boucle ouverte (voir Figure I.10)**.**

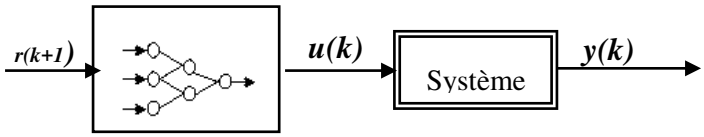


Figure I.10. Commande directe par modèle inverse

La valeur de $y(k+1)$ dans (I.39) est substituée par la sortie désirée $r(k+1)$, et nous alimentons le réseau par les valeurs retardées de $u(k)$ et $y(k)$. Si le modèle neuronal est exactement l'inverse du système, alors il conduit la sortie à suivre la consigne [Ichikawa, 1992].

I.7.3. Commande neuronale hybride

La deuxième approche de commande est celle qui fait intervenir le réseau de neurones au sein d'une structure de commande à laquelle participe aussi un contrôleur classique existant. Elle est présentée dans [Gomi,1993] dans le cadre de la manipulation d'un bras de robot. Elles consistent globalement à faire fonctionner simultanément un contrôleur feedback conventionnel et un modèle connexionniste. Cette approche ne vise plus à apprendre la dynamique inverse du système, mais à effectuer une régulation consistante de celui-ci. Nous fournissons au régulateur feedback classique ainsi qu'au réseau de neurone l'erreur e_s commise en sortie du système. Le réseau dispose de plus de la consigne $r(k)$. Le signal de commande total est constitué de la somme des sorties des deux dispositifs. L'apprentissage des poids synaptiques est, quant à lui, réalisé par rétro-propagation de la sortie du régulateur classique. Cette structure est représentée sur la Figure I.11, [Salem, 2007].

L'approche est justifiée par le fait que lorsque la régulation opérée est consistante, la sortie du régulateur classique est nulle, confiant ainsi la tache de régulation au réseau de neurones. Il s'agit d'une optimisation du fonctionnement du modèle neuronal par d'autres techniques. Citons à ce titre, [Jorgensen, 1990] utilisant cette méthode pour réaliser la commande d'un modèle d'avion en phase d'atterrissage. L'apprentissage de la dynamique d'un bras articulé par une

méthode de ce type est évoqué dans [Guez, 1990]. Pour la commande d'une colonne à distiller, [Steck, 1991] proposent de copier un régulateur existant.

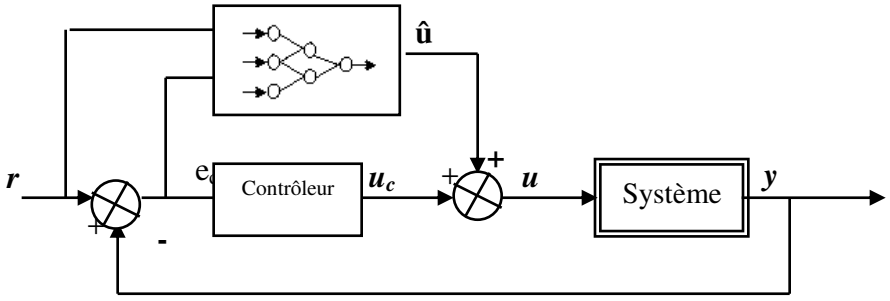


Figure I.11. Structure de commande neuronale hybride

En ce qui concerne la coopération des modèles feedback classiques avec des modèles neuronaux, [Gomi, 1993] procèdent à des tests sur le problème classique du pendule inversé. Enfin, [Psaltis,1987] présentent un autre exemple de coopération, dans lequel le modèle neuronal utilise des informations issues d'un PID lors de son initialisation.

I.7.4. Copie d'un contrôleur existant

Si nous disposons d'un contrôleur capable de commander le processus, l'information pour entraîner le réseau de neurones peut être issue de ce contrôleur, comme l'indique la Figure I.12. Pour une entrée donnée, la sortie désirée pour le réseau de neurones est la sortie du contrôleur classique. Le réseau de neurones ainsi entraîné apprend le contrôleur existant.

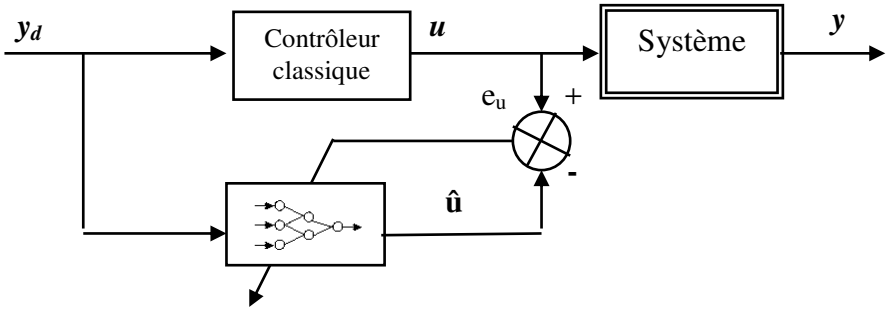


Figure I.12. Copie d'un contrôleur classique existant

Nous pouvons s'interroger sur l'intérêt d'une telle méthode où nous disposons déjà d'un contrôleur. Plusieurs arguments peuvent répondre à cette objection. Le contrôleur classique qui sert à entraîner le réseau de neurones peut consister en un appareillage difficile à mettre en œuvre

en dehors des expériences servant à l'apprentissage. Il peut s'agir d'un opérateur humain qu'on ne saurait affecter en permanence à la tâche de régulation considérée.

I.8. APPLICATION

I.8.1. Exemple d'application

Pour montrer les caractéristiques et les performances des différentes stratégies de commande présentées dans ce chapitre, le système didactique masse ressort amortisseur sera utilisé. Le système composé d'une masse, d'un amortisseur et d'un ressort rigide et possédant une non linéarité (voir Figure I.13), a été étudié par [Norgaard, 1996].

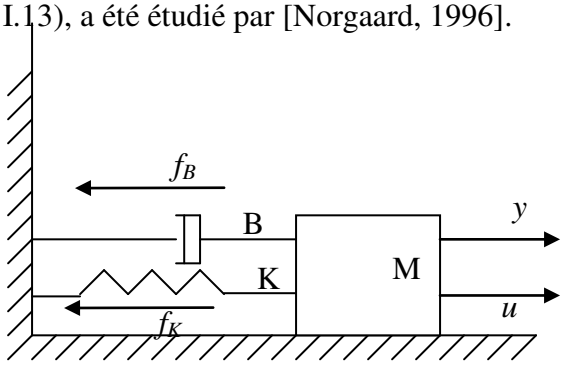


Figure I.13. Système masse- ressort-amortisseur

L'équation de mouvement est donnée par:

$$M\,\ddot{y}(t) = u - f_K(t) - f_B(t) - f_C(t) + d \tag{I.40}$$

Où y désigne la position, $\boldsymbol{x} = [y, \dot{y}]^T$, $f_K(x)$la force du ressort due à K, $f_B(x)$ la force du frottement due à B, $f_C(x)$ la force des frottements de Coulomb, et d la perturbation externe. On suppose que la trajectoire désirée est donnée par $y_d(t) = 0.5\sin(t)$. La table (I.1) donne les valeurs et les définitions des paramètres nominaux, les forces du ressort et des frottements de Coulomb.

Paramètres nominaux	M= 1, K=2, B=2
Les forces du ressort et des frottements de Coulomb	$f_C(x)= 0.01\ \text{sign}(\dot{y})$, $f_B(x) = 2\,\dot{y}$, $f_K(x)= 2\,y$

Table I.1: Paramètres de simulation

I.8.2. Commande par modèle inverse

Pour l'application du modèle inverse pour la commande directe, nous avons fait en premier lieu l'apprentissage du modèle inverse, dont le signal d'entrée $u(k)$ est un signal suffisamment riche en information et persistant avec amplitude dans l'intervalle [-5 5].

Après plusieurs essais, un réseau de neurones d'une seule couche cachée avec 7 neurones a été retenu avec des tangentes hyperbolique (*tansig*) comme fonction d'activation et une sortie linéaire, {$y(k+1)$, $y(k)$,$y(k-1)$, $y(k-2)$, $u(k-1)$, $u(k-2)$} sont ses 4 entrées. L'apprentissage a été effectué par l'algorithme de Levenberg-Marquardt (voir Figure I.14).

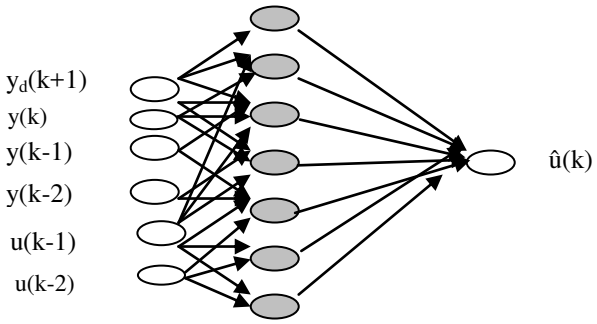


Figure I.14. Architecture du réseau inverse

L'erreur de prédiction ($\hat{u}$-u) est présentée dans Figure I.15.

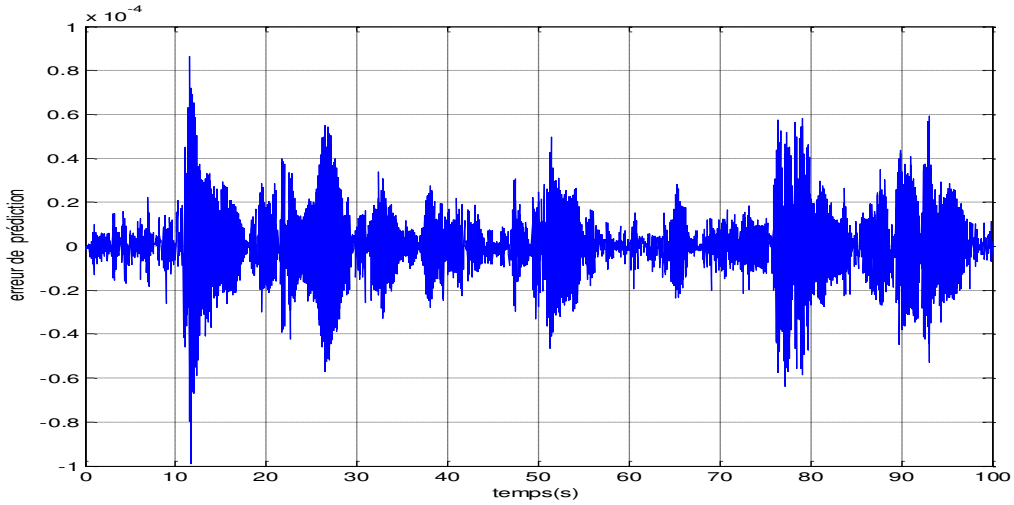


Figure I.15. Erreur de prédiction du modèle inverse

Après, nous avons testé le modèle pour deux consignes, l'une est un signal aléatoire *yd* (voir Figure I.16).

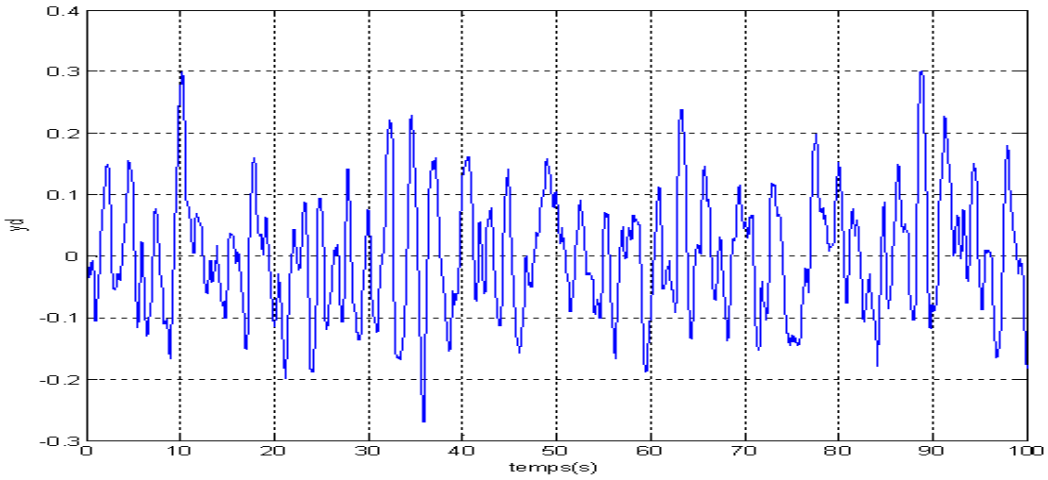


Figure I.16. Consigne y_d

Nous présentons l'erreur entre la consigne y_d et la sortie du système commandé par modèle inverse dans (voir Figure I.17).

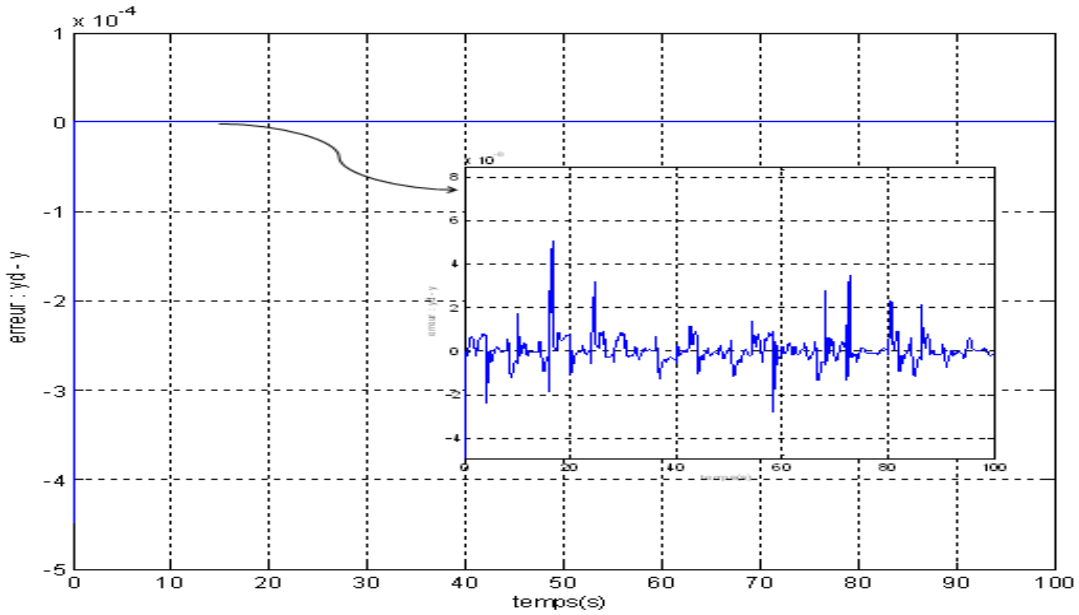


Figure I.17. Erreur commise lors de la commande directe (Consigne y_d)

I.8.3. Commande neuronale hybride

Quant à la commande hybride directe, nous avons obtenu des résultats satisfaisants dont l'erreur entre la consigne y_d et la sortie y est présentée en Figure I.18 :

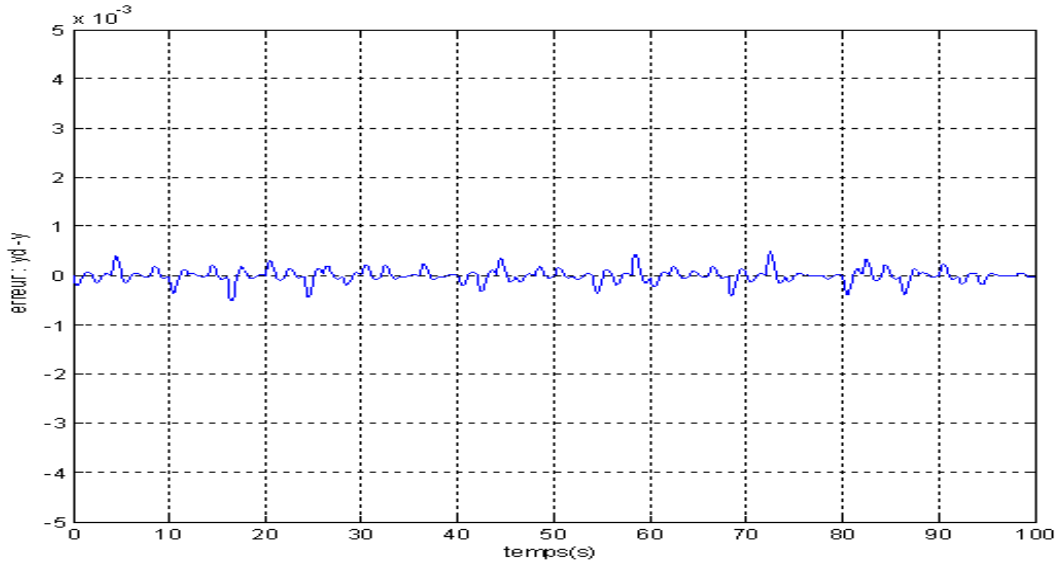


Figure I.18. Erreur commise lors de la commande hybride (Consigne y_d)

Nous remarquons des pics au début de la séquence, ceci est dû au fait que le réseau au début n'a pas assez d'informations passées pour suivre la consigne. Lorsque le temps passe et le réseau acquérait les informations nécessaires, l'erreur de poursuite est très réduite.

En dernier, nous avons introduit un bruit blanc (voir Figure I.19) dans le système en appliquant la commande hybride utilisée dans la simulation précédente. Les erreurs pour les consignes y_d et y_{d1} sont montrées en Figure I.20.

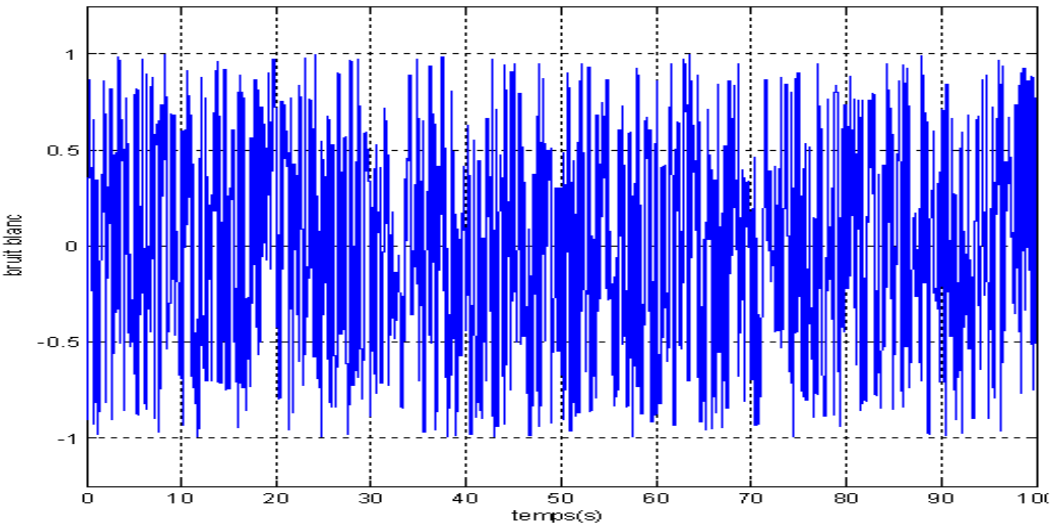


Figure I.19. Bruit blanc introduit dans le système

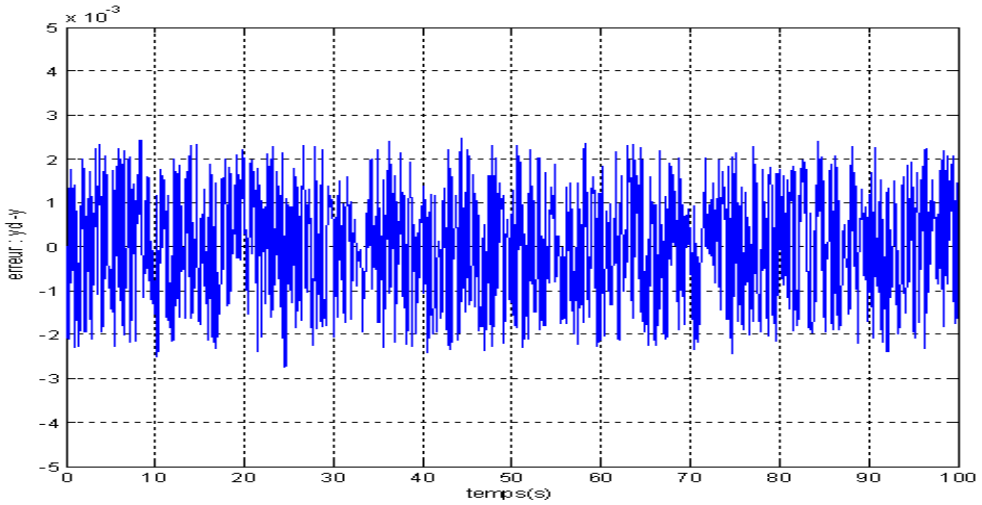


Figure I.20. Erreur commise lors de la commande hybride avec bruit (Consigne y_d)

I.9. CONCLUSION

Au cours de ce chapitre, nous nous sommes intéressés au problème de l'identification par réseaux de neurones. Pour cela, nous avons commencé par rappeler le principe des réseaux de neurones et la procédure générale à suivre pour identifier un système. Les différentes étapes sont similaires à celles utilisées en linéaire. Nous avons présenté les divers algorithmes d'apprentissage. L'intérêt de la détermination de l'architecture optimale a été mis en évidence.

Ensuite, nous avons présenté des structures de commande des systèmes non linéaires par réseaux de neurones basées sur le modèle neuronal inverse, l'apprentissage de ce modèle a été fait après plusieurs tentatives pour arriver à une architecture optimale, et minimiser le nombre de paramètres du modèle. Le modèle retenu a été utilisé ensuite dans une structure de commande directe en boucle ouverte mais ceci suppose que le modèle inverse est presque parfait et le système est non bruité, ce qui est loin de la réalité. Pour essayer de remédier à ce problème, nous avons utilisé le modèle inverse dans une structure hybride avec un PID classique, et nous somme arrivé à des résultats très encourageants. Et enfin, et pour valider la structure hybride nous avons injecté un bruit mais ceci n'a pas trop affecté le système commandé, ce qui traduit une relative robustesse de la méthode.

CHAPITRE

II

LES SYSTEMES NEURO-FLOUS

II.1. INTRODUCTION

Plusieurs techniques dites "modernes" ou "non-conventionnelles" émergeant de l'intelligence artificielle ont été proposées et récemment regroupées par L.A. ZADEH [ZAD, 96] au sein du concept unique de *soft computing*. Ce concept décompose d'une manière didactique ces techniques en deux classes : approches *cognitive* et *non cognitive* qui ne sont pas des techniques concurrentes mais plutôt synergiques, et possèdent de nombreuses ressemblances. Une approche non cognitive est une approche utilisée lorsque le système est trop complexe (le cas d'un espace multidimensionnel) de sorte que l'être humain ne peut ni l'analyser ni expliciter les connaissances s'y rapportant.

Dans cette catégorie, on classe toutes les méthodes d'apprentissage supervisé et non supervisé, statistiques et neuronales. Elles nécessitent toutes des données suffisamment nombreuses pour fournir des échantillons d'apprentissage représentatifs. Contrairement à cette dernière, l'approche cognitive exploite les connaissances disponibles sur le système pour les décrire sous forme experte sans avoir besoin de modèles mathématiques. Dans cette classe, on peut inclure par exemple les systèmes experts classiques ou flous, les contrôleurs flous et les techniques basées sur la théorie de l'incertain.

En fait, la description d'un système est fonction de sa complexité et des connaissances dont on dispose, qui sont généralement imparfaites. Ces imperfections ont été traitées dans leur globalité jusqu'alors par la théorie des probabilités qui est initialement conçue pour la modélisation des jeux de hasard et les observations de fréquence dans les phénomènes physiques. Mais, celle-ci ne permet pas de traiter des opinions subjectives (croyance dans l'occurrence d'un événement) ni de résoudre le problème posé par les connaissances imprécises ou vagues. En effet, étudier ou gérer des systèmes complexes nécessite la prise en compte de certains types d'incertitudes liées à la difficulté des observations, aux imprécisions linguistiques, à la fiabilité tant des observateurs humains que des capteurs et instruments de mesure, à l'utilisation de connaissances empiriques et à l'imprécision du raisonnement humain. Par ailleurs, l'imperfection de l'information est rarement d'un type unique, et dépend fortement du contexte réel. Les notions d'imprécis et d'incertain sont fortement liées, et il est souvent difficile de les identifier clairement et séparément car, dans un raisonnement quelconque, l'imprécision sur une information induit de l'incertitude sur le résultat.

En 1965, L.A. ZADEH [ZAD 65] connu internationalement pour ses travaux sur la théorie des systèmes, a introduit la notion de sous-ensembles flous (*fuzzy sets*). Ceci permet d'une part de représenter et de manipuler des connaissances imprécisées et/ou vaguement

décrites, et d'autre part, d'établir une interface entre des données décrites symboliquement (avec des mots) et numériquement. Contrairement à sa dénomination, cette théorie est très précise et s'appuie sur des fondements mathématiques solides.

En 1972 [ZAD, 72], Il introduit la théorie des possibilités qui constitue un cadre permettant de traiter dans un même formalisme les notions d'incertitude et d'imprécision précédemment définies.

Ce chapitre est consacré à la description des éléments de base de la théorie des systèmes flous et des systèmes neuro-flous. Ces deux concepts constituent la plateforme pour les différents travaux exposés dans cette thèse.

Dans la suite du chapitre seront détaillés deux types spéciaux de systèmes neuro-flous. Il s'agit du réseau neuro-flou (ANFIS : *Adaptive Fuzzy Inference System*) proposé par Jang [Jang, 1993], et le réseau neuro-flou (STFIS : *Self Turing Fuzzy Inference System*) qui présente une grande souplesse d'utilisation et des performances intéressantes en identification [Maaref, 2001], [Maaref, 2000].

II.2. LA LOGIQUE FLOUE

La logique floue (*fuzzy logic*, en anglais) est une extension de la logique classique aux raisonnements approchés. Elle s'appuie sur la théorie mathématique des ensembles flous.

La théorie des ensembles flous est une théorie mathématique du domaine de l'algèbre abstraite. Elle a été développée par Lotfi Zadeh, [Zadeh, 1972] afin de représenter mathématiquement l'imprécision relative à certaines classes d'objets. Les ensembles flous (ou parties floues) ont été introduits afin de modéliser la représentation humaine des connaissances, et ainsi améliorer les performances des systèmes de décision qui utilisent cette modélisation. Les ensembles flous sont utilisés soit pour modéliser l'incertitude et l'imprécision, soit pour représenter des informations précises sous forme lexicale assimilable par un système expert, [Dubois, 1980], [Ross, 2009].

Les parties floues (ou ensembles flous) sont définies comme des ensembles pouvant contenir des éléments de façon partielle. Une partie floue A de B est caractérisée par une application de B dans [0,1]. Cette application, appelée **fonction d'appartenance** et notée μ_A, représente le degré de validité de la proposition "x appartient à A" pour chacun des éléments x de B. Si $\mu_A(x) = 1$, l'objet x appartient totalement à A, et si $\mu_A(x) = 0$, il ne lui appartient pas du tout.

Pour un élément x donné, la valeur de la fonction d'appartenance $\mu_A(x)$ est appelée **degré d'appartenance** de l'élément x au sous-ensemble A.

II.2.1. Propriétés

- Le **noyau** d'une partie floue A est l'ensemble des éléments qui appartiennent totalement à A, c'est-à-dire dont le degré d'appartenance à A vaut 1.

$$n(A) = \{x \in B / \mu_A(x) = 1\} \quad \text{(II.1)}$$

- Le support d'une partie floue A est l'ensemble des éléments appartenant, même très peu, à A c'est-à-dire dont le degré d'appartenance à A est différent de 0.

$$\text{supp}(A) = \{x \in B \setminus \mu_A(x) > 1\} \quad \text{(II.2)}$$

- La hauteur d'un sous-ensemble flou A de E est définie par :

$$h(A) = \sup\{ \mu_A(x) \setminus x \in B \} \quad \text{(II.3)}$$

- Une sous-partie floue A de B peut aussi être caractérisée par l'ensemble de ses α-coupes. Une α-coupe d'un ensemble flou A est le sous-ensemble net (classique) des éléments ayant un degré d'appartenance supérieur ou égal à α.

$$\alpha\text{-coupe}(A) = \{x \in B \setminus \mu_A(x) \geq \alpha\} \quad \text{(II.4)}$$

II.2.2. Opérateurs et normes

Comme dans la théorie des ensembles classiques, on définit l'intersection, l'union des ensembles flous ainsi que le complémentaire d'un ensemble flou. Ces relations sont traduites par les opérateurs 'et', 'ou', et 'non'. De nouvelles fonctions d'appartenances liées à ces opérateurs sont établies :

x appartient à A et $B \Leftrightarrow x \in A \cap B \Leftrightarrow \mu_{A \cap B}(x)$

x appartient à A ou $B \Leftrightarrow x \in A \cup B \Leftrightarrow \mu_{A \cup B}(x)$

x appartient au complément de $A \Leftrightarrow x \in \bar{A} \Leftrightarrow \mu_{\bar{A}}(x)$

L'opérateur 'et' se définit par une norme triangulaire (t-norme) :

$T : [0,1]\text{x}[0,1] \rightarrow [0,1]$

$(x, y) \rightarrow z = xTy$

T possède les propriétés suivantes :

- commutativité : $xTy = yTx$
- associativité : $xT(yTz) = (xTy)Tz$
- monotonie : $xTz \leq yTz$ si $x \leq y$
- admet 0 comme élément absorbant et 1 comme élément neutre : $0Tx = 0$, $1Tx = 1$.

De même, l'opérateur 'ou' se définit par une co-norme triangulaire (T^*), qu'on appel aussi s-norme (S) :

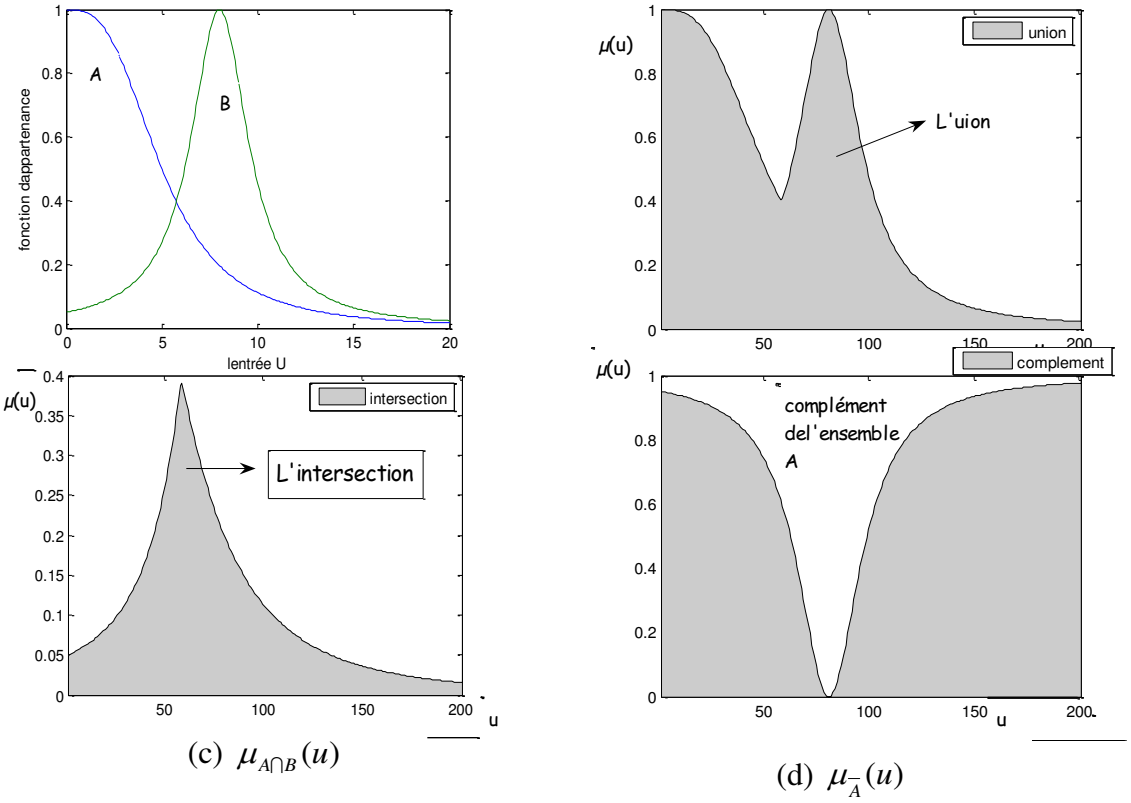
$S : [0,1]x[0,1] \rightarrow [0,1]$

$(x, y) \rightarrow z = xSy$

S possède également les propriétés de commutativité et de monotonie. Elle admet 1 comme élément absorbant et 0 comme élément neutre.

Les opérateurs les plus utilisés en logique floue sont :

- L'opérateur ''et '' pour la t-norme, qui correspond à l'intersection de deux ensembles *A* et *B*. Il peut être réalisé par :
  - La fonction ''Min'' : $\mu_{A\cap B}(x) = min\ (\mu_A(x), \mu_B(x))$
  - La fonction arithmétique ''Produit'' : $\mu_{A\cap B}(x) = \mu_A(x).\ \mu_B(x)$
- L'opérateur ''ou'' pour la s-norme, qui correspond à l'union de deux ensembles *A* et *B*. Il peut être réalisé par :
  - La fonction ''Max'' : $\mu_{A\cap B}(x) = max\ (\mu_A(x), \mu_B(x))$
  - La fonction arithmétique ''somme'' : $\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x)$
  - L'opérateur ''non'' est réalisé par : $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$



(c) $\mu_{A\cap B}(u)$ (d) $\mu_{\bar{A}}(u)$

Figure II.1. Les différents opérateurs de la logique floue

II.2.3. Inférence

En logique floue, la règle s'appelle '*modus ponens généralisé*'. L'inférence est l'opération d'agrégation des règles. L'ensemble B' est définit par :

$$\forall y \in B, \quad \mu_{B'}(y) = \sup_{x \in A} \mu_{A' \times B' \cap \Gamma}(x, y),$$

Avec Γ est le graphe définissant la relation (R) de A vers B.

C'est-à-dire que le degré d'appartenance de chaque élément y de B à l'ensemble flou B' est égal au plus grand degré d'appartenance des couples (x, y) à l'intersection de l'ensemble A' avec le graphe Γ de la relation R. Ce dernier est calculé en utilisant la fonction 'Min' pour l'opérateur 'et' de l'intersection : $\mu_{A' \times B' \cap \Gamma}(x, y) = \min(\mu_{A'}(x), \mu_R(x, y))$.

En ce qui nous concerne, on s'intéresse aux inférences avec plusieurs règles. En effet, dans le cas de la commande et de régulation, les variables floues ont plusieurs fonctions d'appartenance. Ainsi plusieurs règles peuvent être activées en même temps.

Les règles d'inférence peuvent être décrites de plusieurs façons :

a) Linguistiquement : on écrit les règles de façon explicite,

b) Symboliquement : Il s'agit en fait d'une description linguistique, où l'on remplace la désignation des ensembles flous par des abréviations.

c) Par matrice d'inférence : elle rassemble toutes les règles d'inférence sous forme d'un tableau à deux dimensions.

La commande floue est une application de la logique floue au contrôle des systèmes dynamiques pour lesquels on ne possède pas des modèles satisfaisants. Son principe est simple, il s'agit dans la plupart des cas d'imiter le comportement d'un opérateur humain dans la régulation d'un processus complexe, à l'aide de règles floues. Cette méthode de conduite de processus est mise en œuvre grâce à un dispositif comportant [Mamdani, 1977] :

- un certain nombre d'*entrées* (elles se rapportent à des variables d'état du système, dont les grandeurs physiques sont mesurées précisément, à l'aide de capteurs ou sondes) ;
- des *règles floues* assurant le passage des entrées aux sorties ;
- un certain nombre de *sorties* (les différents paramètres faisant l'objet d'une régulation).

Ce dispositif est appelé usuellement contrôleur flou. Un contrôleur flou est en fait un système expert qui fonctionne à partir d'une représentation des connaissances basée sur les sous-ensembles flous. Comme tout système expert, il dispose donc d'une *base de faits* (état des paramètres d'entrée du système, dont les grandeurs exactes instantanées sont connues), d'une *base de connaissances* (description des variables et règles floues), et d'un *moteur d'inférence* (c'est au niveau de ce dernier bloc que se déroule le mécanisme de déduction du

système, qui débouche sur la détermination des valeurs de sortie du contrôleur flou) [Mamdani, 1977].

Les contrôleurs flous se démarquent cependant des systèmes experts classiques par leur petit nombre de règles, par un traitement doux des contradictions (deux règles aux effets opposés auront un effet compensatoire l'une sur l'autre), et également par leur rôle qui consiste à modifier en permanence les paramètres de commande d'un processus [Ross, 2009].

Un système flou de commande, à *n* entrées et une sortie (MISO), est caractérisé par :

- une base de connaissances exprimée sous forme de règles du type :
  règle *i*: si x_i est A_{i1} ... et x_n est A_{in} alors *y* est B_i, où les x_i sont les variables d'entrée, *y* est la variable de sortie, les A_{ij} et les B_i sont des Sous-ensembles flous.
- une méthode d'évaluation des entrées par rapport aux sous-ensembles flous (fuzzification) et une méthode de passage du flou au réel pour la sortie Y (défuzzification).
- un mécanisme d'inférence permettant d'évaluer l'ensemble des règles et d'en tirer une conclusion. Ceci peut être illustré par le schéma de la figure II.2.

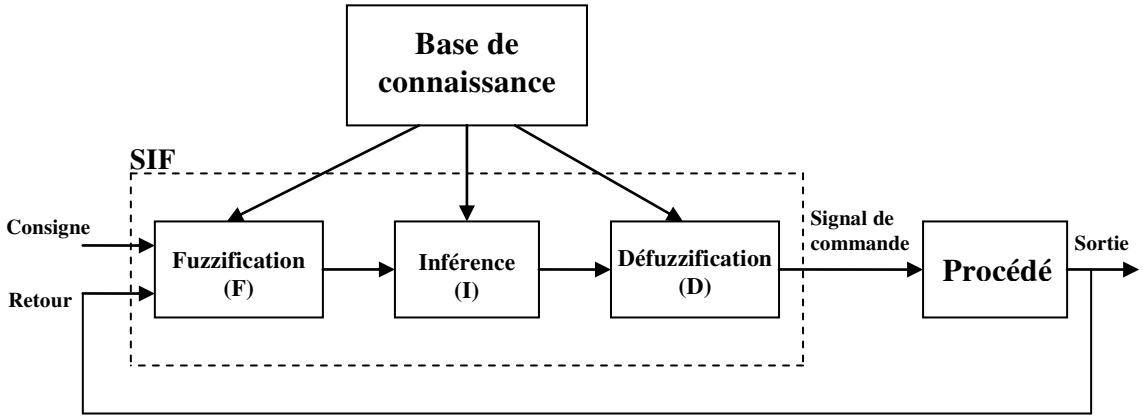
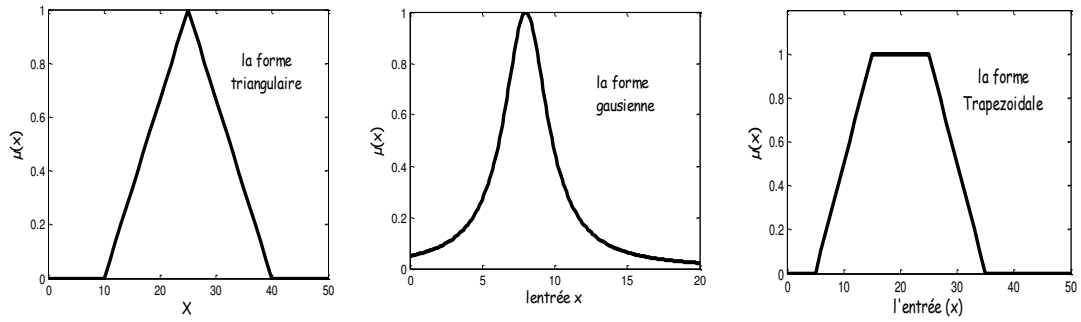


Figure II.2. Configuration de base d'un SIF

Donc, un régulateur par logique floue est constitué de trois parties : la fuzzification, l'inférence, et la défuzzification [Mamdani, 1977].

II.2.4. La Fuzzification

En général, on utilise des formes triangulaires ou trapézoïdales pour les fonctions d'appartenance, bien qu'il n'existe pas de règles précises sur ce choix. Cette étape s'occupe de la transformation des valeurs numériques aux entrées en valeurs floues en utilisant les bases de données. Comme le traitement de données dans un SIF est basé sur la théorie des ensembles flous, la fuzzification doit être faite a priori.



Forme triangulaire Forme gaussienne Forme trapézoïdale

Figure II.3. Les fonctions d'appartenance usuelles.

II.2.5. Inférence ou base de règles

Le bloc inférence est le cœur d'un SIF, qui possède la capacité de simuler les décisions humaines et de déduire (inférer) les actions de commande floue à l'aide de l'implication floue et des règles d'inférence dans la logique floue. Les règles proviennent donc des sources suivantes: expériences d'experts et connaissances de commande, actions des opérateurs de commande et apprentissage du régulateur [Dubois, 1980].

II.2.6. Différentes méthodes d'inférence

Il existe différentes possibilités d'exprimer les inférences, à savoir par description linguistique, par description symbolique, par matrices d'inférence ou par tableau d'inférence.

Plusieurs possibilités existent pour la réalisation des opérateurs de la logique floue qui s'appliquent aux fonctions d'appartenance. A partir de ces possibilités, on introduit la notion de méthodes d'inférence permettant un traitement numérique de ces inférences; en général, on utilise l'une des méthodes suivantes [Wang, 1999] :

- Méthode d'inférence Max - Min.
- Méthode d'inférence Max - Prod.
- Méthode d'inférence Somme - Prod.

II.2.7. Défuzzification

Les méthodes d'inférence fournissent une fonction d'appartenance résultante pour la variable de sortie. Il s'agit donc d'une information floue qu'il faut transformer en grandeur physique [Wang, 1999]. Nous distinguons 4 méthodes de défuzzification :

- Méthode du maximum.
- Méthode de la moyenne des maxima.
- Méthode du centre de gravité.
- Méthode de la somme pondérée.

II.2.8. Types de régulateurs flous

Il existe deux types de systèmes d'inférence floue (SIF), la méthode de Mamdani [Mamdani, 1977] et la méthode de Sugeno [Takagi, 1985], les deux méthodes diffèrent au niveau de la définition de la variable de sortie et, par conséquent, méthodes de défuzzification. Rien ne change au niveau de la fuzzification des variables d'entrée. Pour la méthode de Sugeno, la variable de sortie prend soit une valeur constante (singleton) indépendante des valeurs des entrées, soit une combinaison linéaire de celles-ci. Ce singleton sera, lors de l'étape de défuzzification, pondéré par les degrés d'appartenance des variables d'entrées. La règle générale d'une règle de type Sugeno [Sugeno, 1988] est :

Si X_1 est A ET X_2 est B ALORS sortie = $p*e_1+q*e_2+r$

Avec A et B désignant les fonctions d'appartenance, respectivement, de X_1 et X_2, et p, q, r des constantes choisies par l'utilisateur pour définir la combinaison linéaire des entrées.

Dans la conception d'un SIF, on n'a pas besoin de connaître le modèle mathématique précis du système à contrôler. Si toutefois le modèle existe, il peut être utilisé pour la simulation et pour le test de la stratégie de commande. On peut citer quelques autres caractéristiques des SIF :

- Il est possible d'implémenter les connaissances et l'expérience d'un opérateur, concepteur ou expert humain, en utilisant des règles linguistiques compréhensibles et faisant appel à des notions imprécises.
- La distinction entre systèmes linéaires et non linéaires n'existe pas.
- Grâce aux fonctions d'appartenance, on peut faire le lien entre la connaissance linguistique qualitative et les données quantitatives.

Mais face à ces avantages, le concepteur d'un SIF se trouve affronté à un certain nombre de problèmes. En effet, une base de règles floues est le résultat de l'expression d'une expertise. On cherche à modéliser un système physique, le savoir-faire d'un opérateur ou d'un expert. Or, il n'y a pas de méthode systématique et standard pour la transformation des connaissances humaines ou de l'expérience en base de règles et aucune procédure générale pour choisir le nombre optimal de règles. La connaissance des opérateurs humains est souvent incomplète et non systématique. En effet, un opérateur expérimenté prend des décisions qualitatives sur des

situations spécifiques dont il n'a qu'une connaissance imparfaite mais, grâce à son expérience, il peut s'adapter facilement et rapidement à des situations nouvelles ou imprévues [Wang, 1999]. En pratique, une démarche empirique de mise au point de SIF est coûteuse en temps et ne garantit pas des résultats satisfaisants sur la cohérence et la complétude de la base de règles.

Plus généralement la définition d'un SIF suppose des choix structurels et des choix paramétriques:

- Choix structurels :
    1- choix des variables d'entrée et de sortie ;
    2- choix du nombre des fonctions d'appartenance pour chaque variable;
    3- choix des relations entre entrées et sorties de la base de règles ;
    4- choix du nombre de règles.
- Choix paramétriques :
    1- paramètres définissant les fonctions d'appartenance d'entrée des règles ;
    2- paramètres définissant les sorties des règles.

Parmi ces choix, ceux susceptibles d'être effectués le plus facilement par une procédure d'optimisation automatique sont les paramètres des fonctions d'appartenance et éventuellement le nombre de règles.

II.3. SYSTEMES NEURO-FLOUS

Les systèmes neuro-flous sont des systèmes flous formés par un algorithme d'apprentissage inspiré de la théorie des réseaux de neurones. La technique d'apprentissage opère en fonction de l'information locale et produit uniquement des changements locaux dans le système flou d'origine [Nauck, 1997].

Les similarités et les complémentarités des techniques neuronales et floues ont été souvent discutées dans la littérature [Wang, 1999]. Les systèmes à réseaux de neurones et à logique floue sont des structures parallèles à caractère paramétrique. Les formes des fonctions d'appartenance des systèmes flous et les fonctions d'activation des réseaux de neurones peuvent être similaires. Les opérations de multiplication-addition des neurones artificiels sont très proches des opérateurs "Sup-Min" du raisonnement approximatif.

Le raisonnement utilisé dans un système d'inférence floue présente une structure que l'on peut qualifier de "pré-neuronale" : chaque étape bien distincte peut en effet être assimilée au calcul effectué au niveau d'une couche dans un réseau neuronal multicouche.

Cependant, dans les réseaux neuronaux, l'inclusion, l'extraction et la représentation des connaissances sont difficiles. Les avantages des systèmes flous sont que les connaissances sont représentées sous forme de règles linguistiques compréhensives et qu'ils sont capables de traiter des informations imprécises. Leurs points faibles sont causés essentiellement par la difficulté à définir avec précision les fonctions d'appartenance et le manque de procédure systématique pour la transformation des connaissances d'un expert en une base de règles. Ces différentes raisons ont conduit, à partir du début des années 90, à la création d'un domaine de recherche, communément désigné par "neuro-flou", cherchant à fusionner les deux techniques [Nguyen, 2005]. Il s'agissait de les "combiner" soit de façon à conserver les avantages des deux, tout en évitant les inconvénients, soit en exploitant les analogies entre les deux. Il y a deux principales méthodes :

- doter le système flou de fonctions d'apprentissage ou reproduire le modèle de traitement à partir du contrôleur flou ;
- transcrire le SIF directement dans une structure de réseau de neurones.

Les réseaux de neurones (*RN*) multicouches constituent des approximateurs universels. L'atout principal de ces réseaux réside dans leur capacité d'apprentissage. Par contre, leur structure et leurs paramètres n'ont pas toujours des justifications physiques. De plus, la connaissance humaine ne peut pas être exploitée pour les construire.

Les systèmes d'inférence flous sont également des approximateurs universels, ces systèmes possèdent deux points forts par rapport aux RN, d'une part, ils sont généralement construits à partir de la connaissance humaine, d'autre part, ils ont une capacité descriptive élevée due à l'utilisation de variables linguistiques. Il est donc apparu naturel de construire des systèmes hybrides qui combinent les concepts des systèmes d'inférence flous et des RN.

Diverses combinaisons de ces deux méthodes ont été développées depuis 1988. Elles ont donné naissance aux systèmes neuro-flous, qui sont le plus souvent orientées vers la commande de systèmes complexes et les problèmes de classification [Sugeno, 1988].

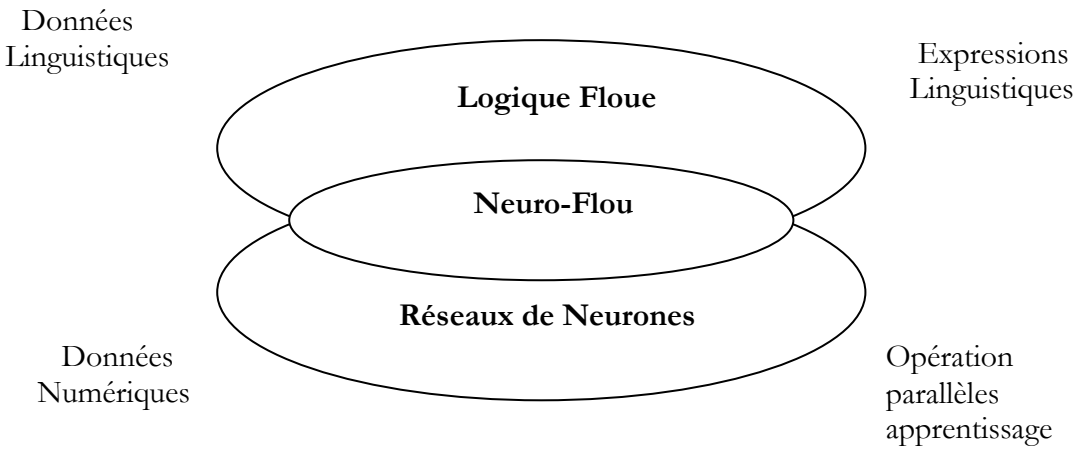


Figure II.4. Le système neuro-flou

La figure.II.1 résume le principe du système neuro-flou qui représente l'intersection entre la logique floue et les réseaux de neurones. Afin de résumer 1'apport du neuro-flou, le TableauII.1 regroupe les avantages et les inconvénients de la logique floue et des réseaux de neurones [Otilia, 2008] [Parizeau ,2004].

Les règles floues codées dans un système neuro-flou représentent les échantillons imprécis et peuvent être vues en tant que prototypes imprécis des données d'apprentissage. Un système neuro-flou ne devrait, par contre, pas être vu comme un système expert (flou), et il n'a rien à voir avec la logique floue dans le sens stricte du terme. On peut aussi noter que les systèmes neuro-flous peuvent être utilisés comme des approximateurs universels [Otilia, 2008].

Réseaux de Neurones	**Logique Floue**
Avantages	
-Modèle mathématique non requis **-** Aucune connaissance basée sur les règles - Plusieurs algorithmes d'apprentissage sont disponibles	-Modèle mathématique non requis -La connaissance antérieure sur les règles peut être utilisée - Une interprétation et une implémentation simples
Inconvénients	
- Boite noire (manque de traçabilité) - L'adaptation aux environnements différents est difficile et le réapprentissage est souvent obligatoire (sauf pour le RBF) - la connaissance antérieure ne peut pas être employée (apprentissage à partir de zéro) (sauf pour le RBF) - Aucune garantie sur la convergence de l'apprentissage	- Les règles doivent être disponibles - Ne peut pas apprendre - Adaptation difficile au changement de l'environnement - Aucune méthode formelle pour l'ajustement

Table II.1 : Comparaison entre la logique floue et les réseaux de neurones.

II.4. TYPES DE COMBINAISONS NEURO-FLOUES

Il existe quatre grandes catégories de combinaisons de réseaux de neurones avec la logique floue : réseau flou neuronal, système neuronal/flou simultanément, modèles neuro-flous coopératifs, et modèles neuro-flous hybrides [Ajith, 2001].

II.4.1. Réseau flou neuronal

Des techniques floues sont utilisées pour augmenter les possibilités d'apprentissage ou l'exécution d'un réseau neuronal. Ce genre d'approche ne doit pas être confondu avec les approches mixtes.

II.4.2. Système neuronal/flou simultanément

Le réseau neuronal et le système flou fonctionnent ensemble sur la même tâche, mais sans s'influencer. Habituellement le réseau neuronal traite les entrées, ou post-traite les sorties du système flou (voir figure II.5).

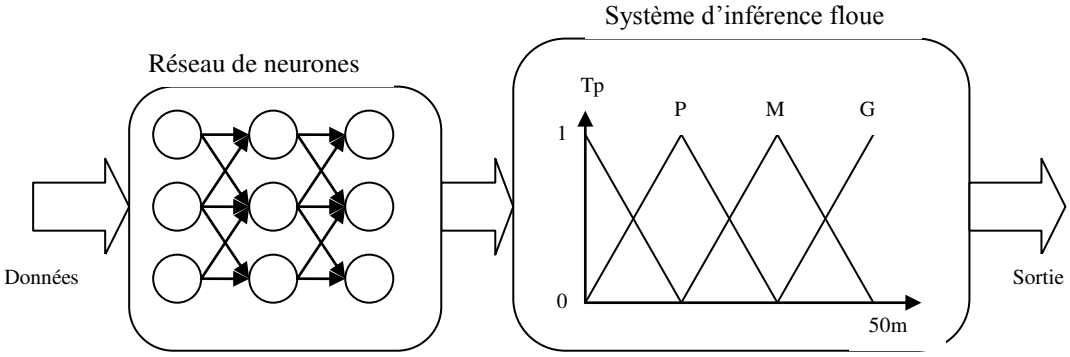


Figure II.5. Association en série d'un réseau de neurone et un système SIF

II.4.3. Modèles neuro-flous coopératifs

Le réseau neuronal est employé pour déterminer les paramètres (les règles et les ensembles flous) d'un système flou. Après la phase d'apprentissage, le système flou fonctionne sans le réseau neuronal. C'est une forme simple des systèmes neuro-flous.

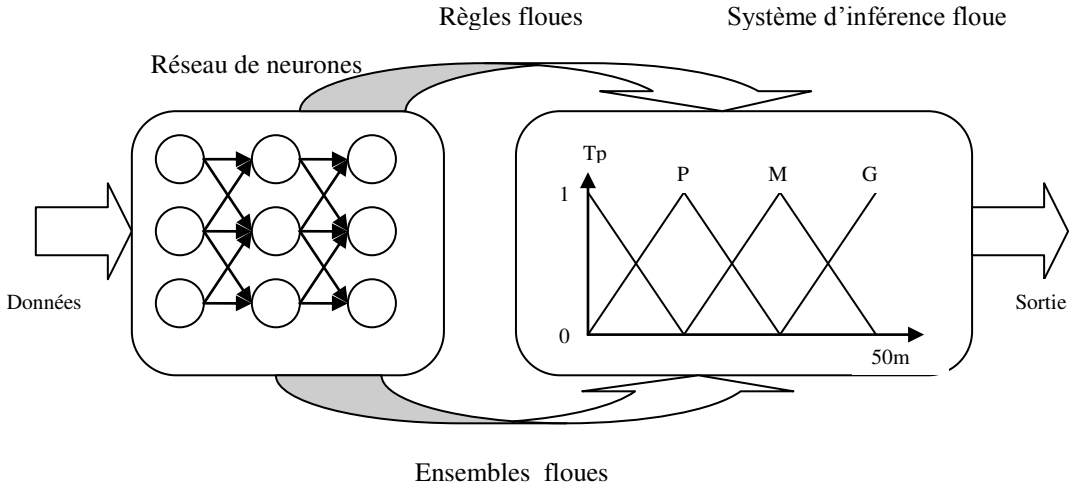


Figure II.6. Association en parallèle d'un réseau de neurone et un système SIF

II.4.4. Modèles neuro-flous Hybrides

Dans une architecture neuro-floue hybride, les réseaux de neurones artificiels sont utilisés pour déterminer les paramètres du système d'inférence flou. Les systèmes neuro-flous hybrides partagent les structures de données et la représentation de connaissance. Le système peut être interprété comme un réseau neuronal spécial avec des paramètres flous, et qui utilise un algorithme d'apprentissage supervisé. (figure.II.7).

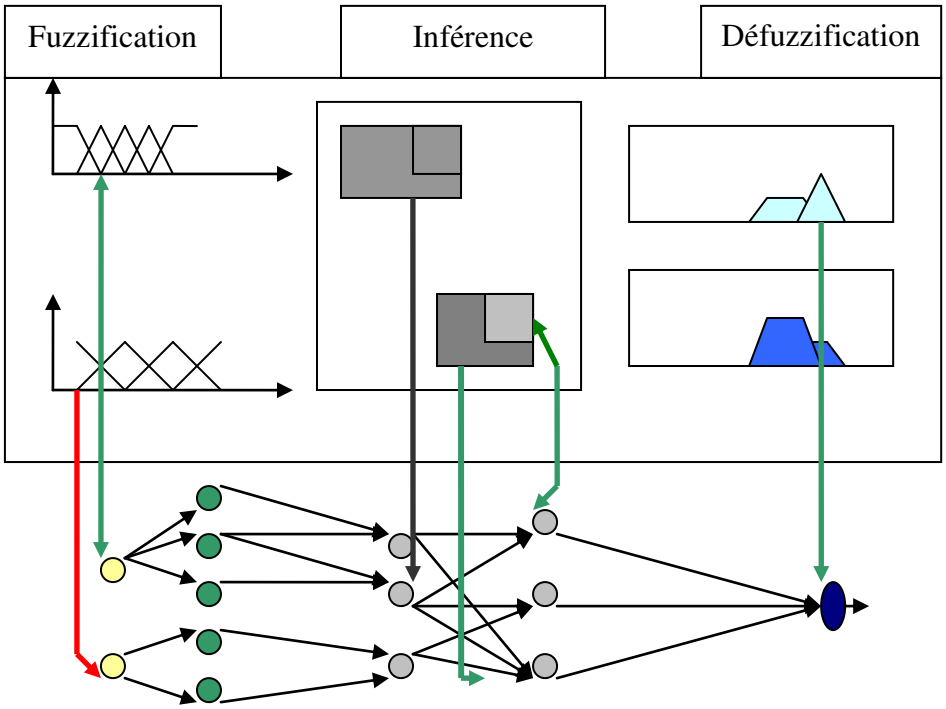


Figure II.7. Principe de fonctionnement d'un système neuro-flou.

II.5. LE RESEAU NEURO-FLOU (ANFIS)

Le réseau neuro-flou adaptatif (ANFIS : *Adaptif Neural Fuzzy Inference System*) est composé d'un ensemble de neurones connectés entre eux par des connexions directes. Chaque neurone modélise une fonction paramétrée; le changement des valeurs de ses paramètres entraîne le changement de la fonction, de même que le comportement total du réseau adaptatif. L'ensemble des paramètres d'un réseau adaptatif est distribué sur l'ensemble des neurones le constituant. Cependant, chaque neurone possède un ensemble de paramètres locaux: si cet ensemble est vide, alors le neurone associé est représenté par un cercle et sa fonction est fixe, neurone fixe, sinon, il est représenté par un carré et la fonction associée dépend des valeurs de ces paramètres, neurone adaptatif [Jang, 1993].

Dans un ANFIS, les connexions entre neurones sont seulement utilisées pour spécifier le sens de la propagation des stimulations provenant des autres neurones.

.

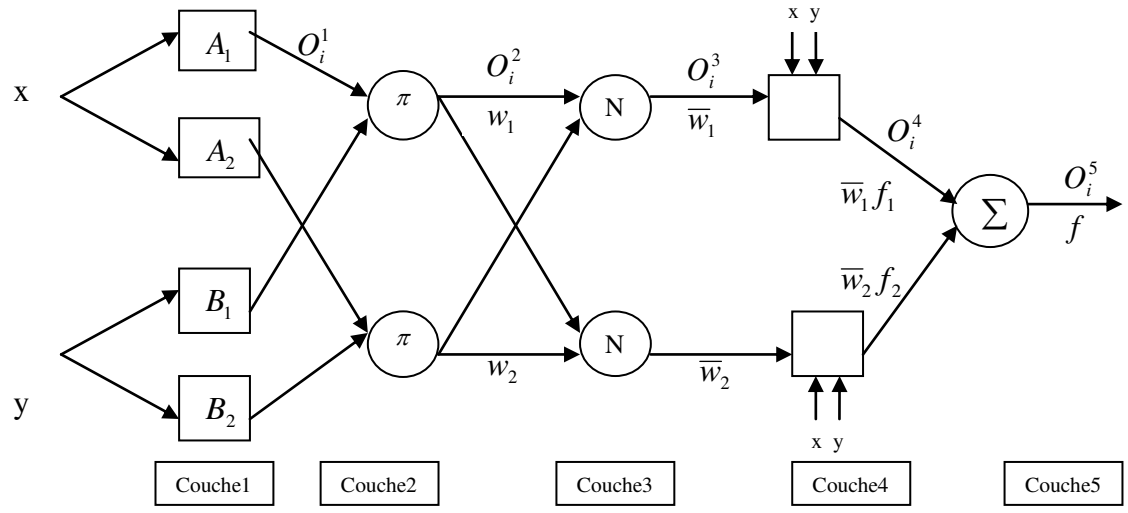


Figure II.8. Structure d'un ANFIS.

La figure ci-dessus représente l'architecture d'un ANFIS formalisant le raisonnement de Sugeno du premier ordre, à deux entrées et une sortie avec une base de règles constituée de deux règles, dont une règle est exprimée par:

$$\textit{Règle } i : \textbf{si } x \text{ est Ai et } y \text{ est Bi } \textbf{alors } f_i = p_i x + q_i x + r_i$$

Correspondant à l'architecture d'ANFIS qui se compose de cinq couches.

Couche .1 :

Les neurones adaptatifs A_i (Bi) calculent les degrés d'appartenance, l'ensemble des paramètres caractérise les fonctions Ai (Bi). Les paramètres correspondant sont appelés paramètres de la prémisse $\{a_i, b_i, c_i\}$

$$O_i^1 = U_{Ai}(x) \tag{II.5}$$

Généralement $U_{Ai}(x)$ est choisi sous forme de cloche avec son maximum égal à 1 et le minimum égal à 0.

$$U_{Ai}(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{\alpha_i}\right)^2\right]^b} \tag{II.6}$$

Où la fonction gaussienne est :

$$U_{Ai}(x) = \exp\left[-\left(\frac{x-c_i}{\alpha_i}\right)^2\right] \tag{II.7}$$

Couche.2 :

Les neurones fixes modélisent l'opérateur "Et" et calculent la valeur de vérité de chaque règle.

$$w_i = U_{Ai}(x) \times U_{Bi}(x) \tag{II.8}$$

***Couche.3* :**

Les neurones N sont fixes, effectuant la normalisation de la valeur de vérité de la règle (poids).

$$\overline{w} = \frac{w_i}{w_1 + w_2}, i = 1,2 \tag{II.9}$$

Couche.4 :

Chaque neurone de la couche 4 est un neurone adaptatif dont la fonction est:

$$O_i^4 = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i) \tag{II.10}$$

Les paramètres $\{\ p_i, q_i, r_i\ \}$ sont appelés les paramètres conséquents.

Couche.5 :

Le neurone de la couche 5 est un neurone fixe, à une entrée donnée; il délivre la réponse du réseau donnée par :

$$O_i^5 = \sum_i \overline{w_i}\ f_i \tag{II.11}$$

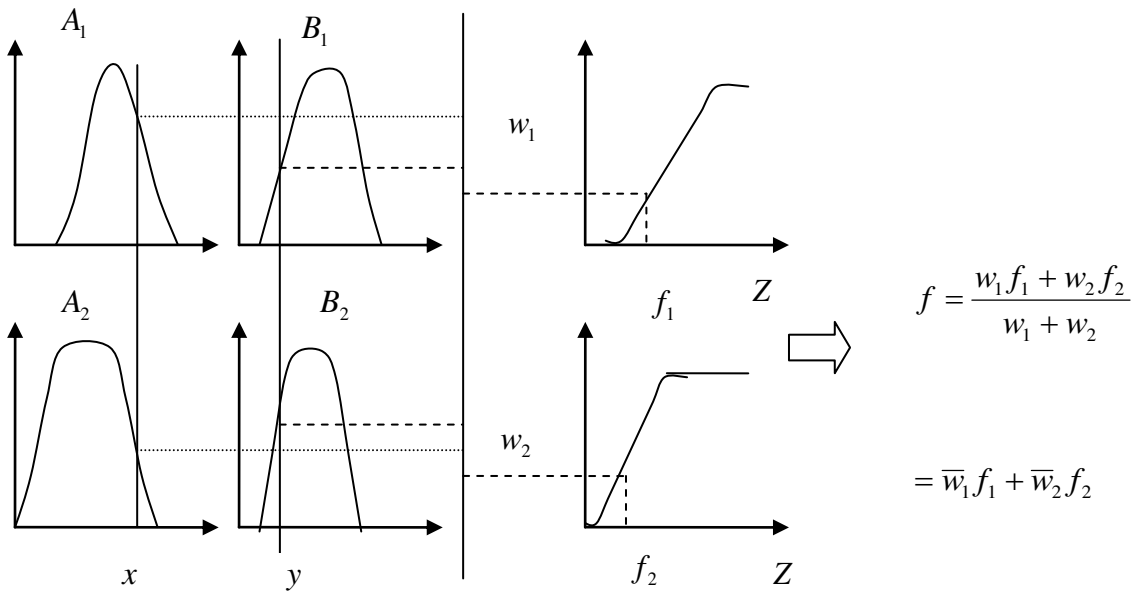


Figure II.9. Partie de défuzzification

Les paramètres prémisses sont identifiés par la méthode de descente du gradient, et les paramètres conséquents sont calculés par la méthode des moindres carrés. Les paramètres conséquents ainsi identifiés sont optimaux à la condition que les paramètres prémisses sont fixés [Ajith, 2001]. Le réseau ANFIS est l'un des tous premiers systèmes neuro-flous qui existent. Il est très cité dans la littérature, car il a prouvé son efficacité au fil du temps avec son algorithme d'apprentissage simplifié : la méthode de descente de gradient et celle des moindres carrés [Jang, 1993], [Belhachat, 2007].

II.5.1 Les avantages de L'ANFIS

Les avantages de cette technique sont :

- Exploitation de la connaissance disponible, grâce à la base de règles.
- Réduction de la taille de la base de règles : il suffit d'avoir des règles générales, les détails seront fournis par le RN.
- Réduction de la complexité de l'apprentissage : le RN doit simplement apprendre les cas particuliers ou les exceptions, pas le problème complet.
- Efficacité immédiate dès le début de l'apprentissage et possibilité d'éviter des comportements initiaux erratiques.

II.6. LE RESEAU NEURO-FLOU (STFIS)

Le réseau STFIS présente une complète analogie avec un système d'inférence floue (SIF) de type Takagi-Sugeno d'ordre zéro [Maaref, 2001]. Ce SIF peut être schématisé sous la forme d'un réseau de neurones à quatre couches (figure II.10):

- La première couche reçoit les entrées (x_1, x_2);
- la seconde calcule les degrés d'appartenance de ces entrées à leurs sous-ensemble flous A_{i1}, A_{i2}; les poids du réseau entre la première couche et cette dernière correspondent aux paramètres définissant les fonctions d'appartenances;
- La troisième couche calcule les valeurs de vérité α_i telle que :
  α_i = ET((x_1 est A_1), (x_2 est A_2)); dans notre cas la fonction '*min*' définissent l'opérateur ET choisi;

Et enfin, la quatrième couche est celle de sortie, les poids w_i des parties conclusion des règles sont pondérés par les valeurs de vérité des prémisses: $u(x_1, x_2) = \sum^m \alpha_i w_i / \sum^m \alpha_i$, les poids w du réseau entre la troisième et la quatrième couche correspondent aux parties conclusions des règles [Zemalache, 2008], [Chaouch, 2011].

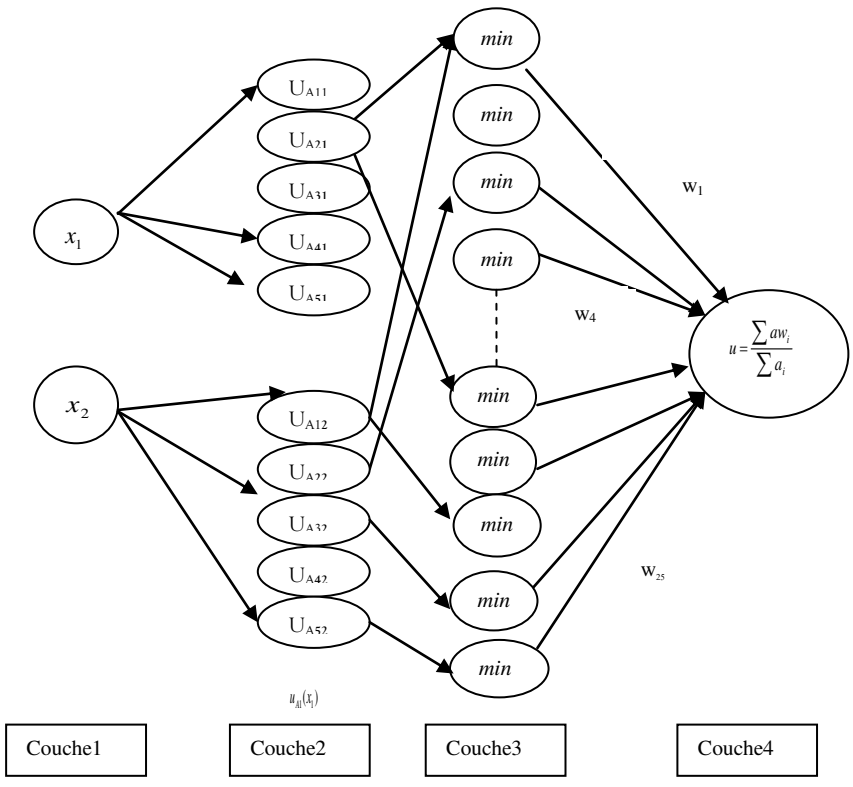


Figure II.10. Structure STFIS

II.6.1. Architecture de contrôle 'JEAN ' et 'mini-JEAN'

Dans le contrôle de processus dynamiques, le contrôleur doit reproduire la fonction de transfert inverse du système pour déterminer la commande à fournir. Cela est aisé dans certains cas simples, par exemple quand le système peut être modélisé sous une forme linéaire, mais le plus souvent la structure du système est inconnue, ou elle présente des non-linéarités non modélisables. Nous nous plaçons dans cette hypothèse, c'est-à-dire qu'on suppose n'avoir aucune connaissance a priori sur le processus à contrôler. Dans ce cas, Jordan propose la méthode du '*distal control'* sous le nom de JEAN (*Jordan method Extended for Adaptive Neuro-control*) [Zemalache, 2006]. Cette architecture (figure.II.11) nécessite la présence de deux réseaux de neurones :

- un premier réseau pour identifier le processus (modèle).
- un second réseau pour contrôler le processus (Contrôleur).

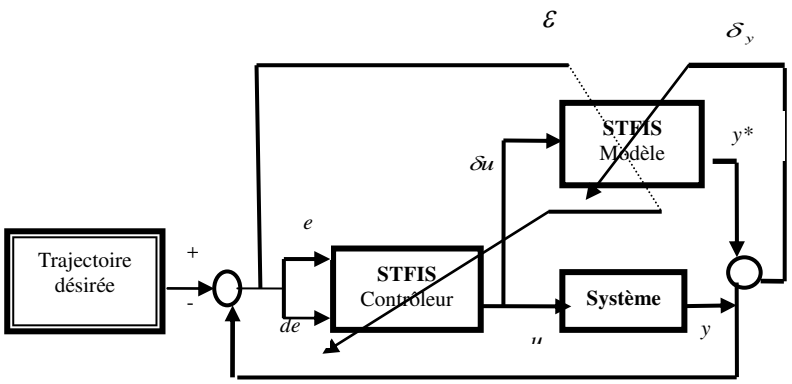


Figure II.11. Architecture JEAN

Cette méthode nécessite deux étapes successives: l'identification du processus par le réseau modèle, puis l'optimisation du réseau à contrôler. L'ajustage des poids synaptiques (ou des conclusions des règles) de ce dernier est effectué en ''rétro-propageant'' à travers le réseau la valeur *u* obtenue par rétro-propagation à travers le modèle d'une fonction coût basée sur l'erreur en sortie $e = y-y^*$.

Mais en fait, souvent il n'est pas nécessaire de connaître précisément le modèle du processus (ou plus exactement son Jacobien) pour obtenir un contrôle correct. En effet, un modèle imprécis du système n'altère pas la minimisation de la fonction coût, il modifie seulement le trajet pris par l'algorithme d'optimisation dans l'espace des actions : on ne suivra sans doute pas la plus forte pente, mais on restera sur un chemin descendant. Le moyen consistant à utiliser l'approximation du Jacobien par son signe est une approche viable pour la commande de systèmes simples. Nous arrivons à une architecture ''mini-JEAN'' avec un seul

réseau contrôleur, dont l'optimisation s'effectue en rétro-propageant directement l'erreur de sortie; cette architecture comparée à l'architecture JEAN, des performances équivalentes ont été obtenues, tant pour la vitesse d'optimisation que pour l'erreur moyenne en généralisation. Par contre, le temps de calcul est nettement en faveur de mini-JEAN.

Ils ont étendu ces résultats en remplacent dans ces architectures le réseau de neurones par un SIF dont certains paramètres sont à optimiser. Ils ont également observé que, malgré quelques différences dans la conduite de l'optimisation et dans la commande fournie, le recours à la version simplifiée n'induit pas de dégradation de performances sensible. C'est pourquoi, notre préférence va à cette méthode efficace, rapide et facile à mettre en œuvre. Il est évident néanmoins, qu'elle ne peut s'appliquer qu'à des systèmes dans lesquels le signe du Jacobien est constant dans le domaine de fonctionnement.

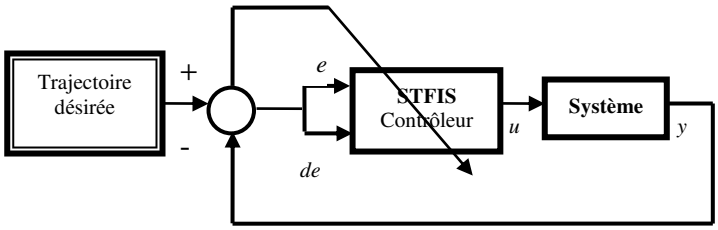


Figure II.12. Architecture mini-JEAN

II.6.2. Algorithme d'optimisation

Plusieurs algorithmes sont utilisables pour optimiser les paramètres ajustables du réseau. Dans notre application, nous avons utilisé la méthode de rétro-propagation du gradient pour ajuster les poids de la dernière couche du réseau.

Le principe général de cette méthode peut être résumé comme suit : à chaque itération, nous modifions les poids de la couche de sortie. Cette modification se fait dans le sens opposé du gradient de la fonction coût. On répète le processus jusqu'à ce que les poids de la couche de sortie ait convergé, c'est-à-dire que l'écart entre la sortie du réseau et celle désirée devient acceptable.

L'optimisation est effectuée entièrement en ligne en minimisant une fonction coût J (intégrant une erreur quadratique et un terme de régression des paramètres) pour générer les paramètres w_i caractérisent la partie conclusion des règles et les ajuster. L'algorithme de descente de gradient avec la régression des paramètres optimisant seulement la partie conclusion des règles a été adopté pour satisfaire les objectifs fixés [Maaref, 2001].

$$J = E + \lambda \sum w_i^2 \tag{II.12}$$

$$E = \frac{1}{2} e^2 \tag{II.13}$$

λ : constante de contrôle d'augmentation des paramètres.

e : terme d'erreur.

w_i : poids ou paramètre.

A l'aide de l'algorithme de rétro-propagation du gradient, les paramètres sont modifiés suivant la relation suivante :

$$w_{ij}^n(t) = w_{ij}^n(t-1) + \Delta w_{ij}^n(t) \tag{II.14}$$

Aussi nous avons :

$$\Delta w_{ij}^n(t) = -\eta \delta_i^n \alpha_j^{n-1} + b \Delta w_{ij}^n(t-1) \tag{II.15}$$

t : représente les itérations.

w_{ij}^n : poids entre le $i^{ème}$ neurone de la couche n et le $j^{ème}$ neurone de la couche n-1.

η : gain d'optimisation (ou pas d'optimisation), mesurant la vitesse avec laquelle les poids vont converger vers leur valeur finale.

b : moment, ce paramètre est compris entre 0 et 1; l'introduction du terme $b\Delta w_{ij}^n(t-1)$ permet d'éviter les minima locaux en faisant intervenir les variations des poids.

α_j^{n-1} : sortie du neurone de la couche n-1.

δ_i^n : terme de dérivée de la fonction coût (*ième* neurone de la couche *n*).

A l'aide de l'algorithme classique de rétro-propagation du gradient, les paramètres sont modifiés suivant la relation suivante :

$$w(t+1) = w(t) + \eta \left(\frac{-\partial J}{\partial w} \right) \tag{II.16}$$

Cet algorithme intègre facilement l'effet du terme de régression dans la fonction coût. En effet, il suffit de dériver la fonction coût J par rapport à chaque paramètre w_i et en posant $\beta = 2\lambda\eta$, nous obtenons :

$$\Delta w_{1j}^4(t) = -\eta \delta_1^4 \alpha_j^3 + b \Delta w_{1j}^4(t-1) - \alpha_j^3 \beta w_{1j}^4(t-1) / \sum \alpha_j^3 \tag{II.17}$$

Avec β : Coefficient du terme de « *weight decay* » ou de régression. En se limitant que pour l'optimisation des conclusions des sorties, ainsi on obtient :

$$\delta_1^4 = y_e - y_d / \sum_j \alpha_j^3 \qquad \text{(II.18)}$$

Où:

y_e : valeur effective de sortie.

y_d : sortie désirée.

Comme il s'agit ici d'un SIF, nous adaptons cette relation en multipliant β par le terme de déclenchement de la règle, à savoir $\frac{\alpha_i}{\sum \alpha_i}$, α_i est la valeur de vérité de la partie prémisse de la règle déclenchée.

Le choix de β détermine le poids dans la fonction coût du terme de régression par rapport à celui de l'erreur quadratique : s'il est trop petit par rapport au gain, l'effet de la régression des poids ne se fera pas sentir. En revanche, si β est trop important, les poids restent proches de zéro, car l'effet de diminution des poids sera trop important par rapport à l'augmentation due à la rétro-propagation normale. Dans ce cas, le SIF n'apprendra rien.

Il est également important que le terme de régression soit proportionnel au déclenchement de la règle. Si on n'introduisait pas ce coefficient, la valeur du paramètre w_i baisserait quand la règle n'est pas déclenchée. Donc si, par exemple, le système évoluait suffisamment longtemps sans déclencher une règle, le paramètre correspondant tendrait vers 0, et on perdrait alors de l'information. Classiquement, les coefficients sont choisis par des essais successifs [Zemalache, 2006].

II.6.3. Architecture de la commande STFIS

Un réseau contrôleur est utilisé pour le contrôle de processus dynamiques, dont l'optimisation s'effectue en rétro-propageant directement l'erreur de sortie.

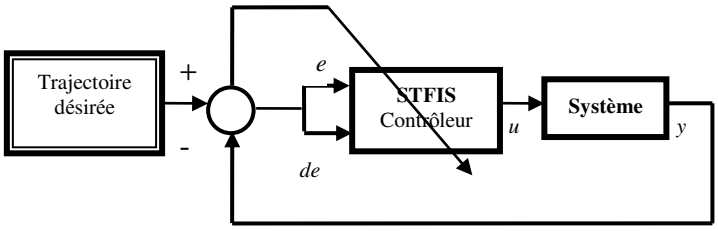


Figure II.13. Architecture de la commande STIFIS

II.6.4. Gains du contrôleur:

Les gains des entrées *Ge, GΔe et Gc* ont un rôle d'un facteur d'échelle. Ces gains affectent les performances (rapidité et précision) de la réponse du système en régime

transitoire. Le gain de la sortie *Gc* joue un rôle dans la stabilité du système et l'élimination des erreurs en régime permanent.

Le choix de ces gains peut se faire d'une manière subjective (essais/erreurs) de sorte à obtenir la meilleure performance possible [Chaouch, 2011].

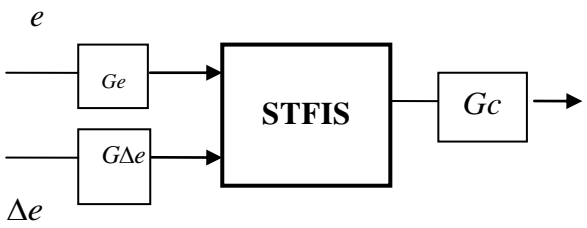


Figure II.14. Facteurs d'échelles.

II.7. SIMULATIONS ET RESULTATS

Les simulations présentées dans cette section sont réalisées sur un système non linéaire, le modèle masse-ressort. Nous avons utilisés les deux techniques neuro-flou ANFIS et STFIS.

II.7.1. Commande inverse ANFIS

Concernant l'application du modèle inverse ANFIS pour la commande, nous avons fait en premier lieu l'apprentissage du modèle inverse ANFIS, dont le signal d'entrée *u*(*k*) est un signal suffisamment riche en informations. Après plusieurs essais, un réseau ANFIS avec quatre entrées {y(k+1),y(k),y(k-1),u(k-2)} a été retenu (voir Figure II.15).

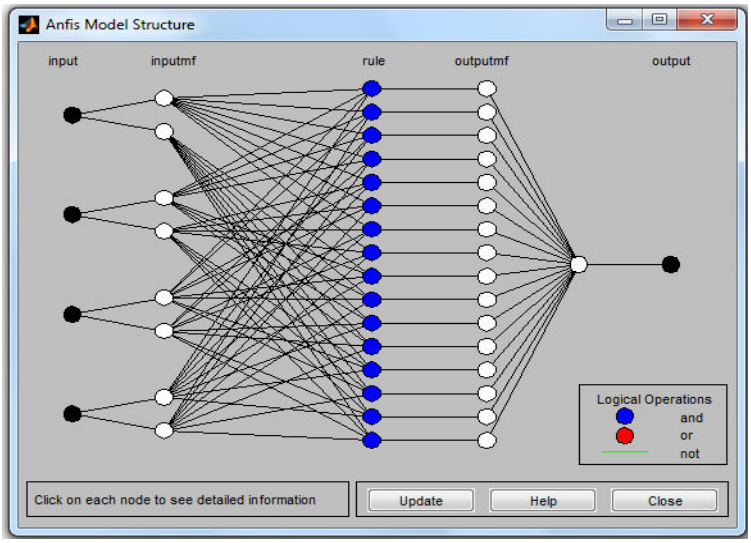


Figure II.15. Architecture du réseau ANFIS

Après, nous avons testé le contrôleur ANFIS obtenu sur le modèle masse-ressort pour une consigne aléatoire *yd* (voir Figure II.16).

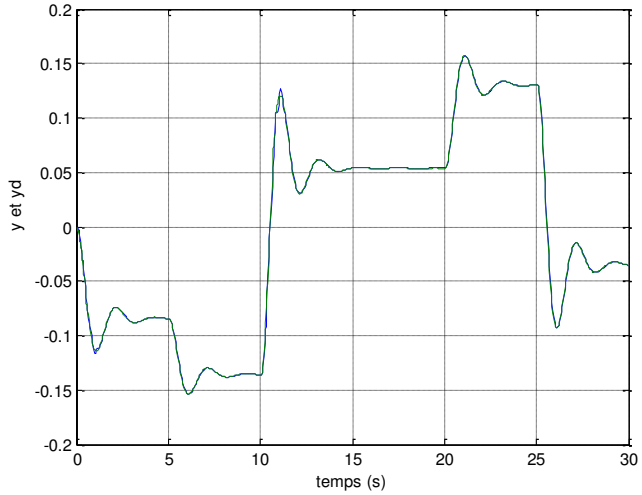


Figure II.16. Poursuite d'une trajectoire aléatoire.

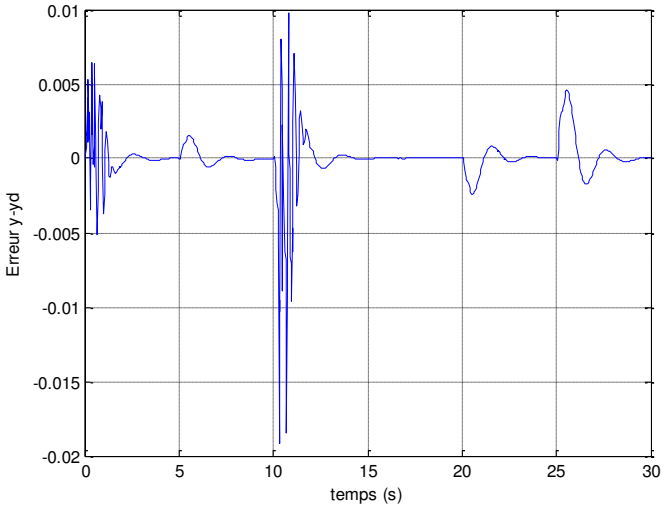


Figure II.17. Erreur de poursuite d'une trajectoire aléatoire.

D'après les figures II.16 et II.17, nous remarquons qu'il ya une bonne poursuite de trajectoire, et que l'erreur de poursuite est très réduite.

II.7.2. Commande STFIS

L'architecture utilisée (voir la figure II.13) est avec deux entrées, l'erreur *e* et sa dérivée. Nous avons cinq fonctions d'appartenance (GN, PN, Z, PP, PG). Les fonctions utilisées sont normalisées dans l'univers [-1,1] de type gaussien et sigmoïde (voir la figure. II.18).

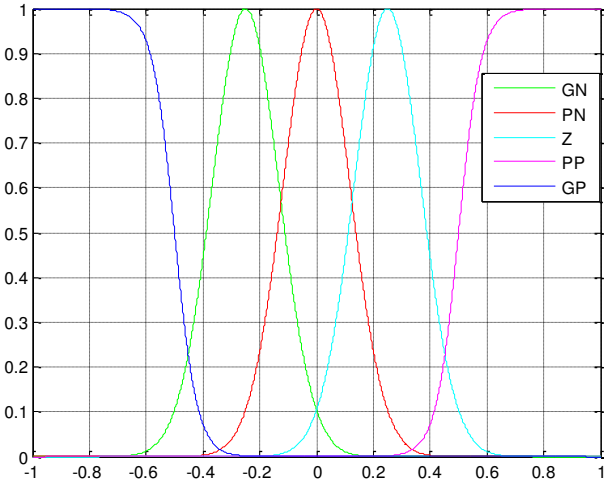


Figure II.18. Fonctions d'appartenances.

Nous avons procédé à des simulations sur le même modèle masse ressort. Les paramètres du contrôleur STFIS sont : $\eta = 0.6$, $b = 0.7, \beta = 0.00001$, et les gains $Ge = 10, G\Delta e = 10$ et $Gc = 1$.

Au début, l'apprentissage se fait en initialisant la table des poids par une table de zéro. Nous avons obtenu les résultats suivants :

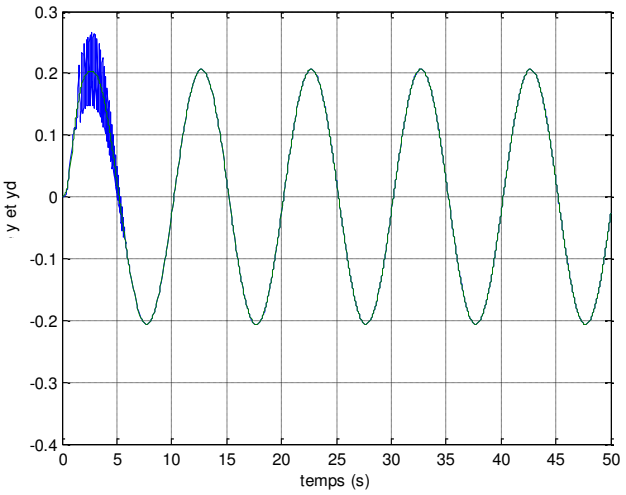


Figure II.19. Poursuite d'une trajectoire sinusoïdale.

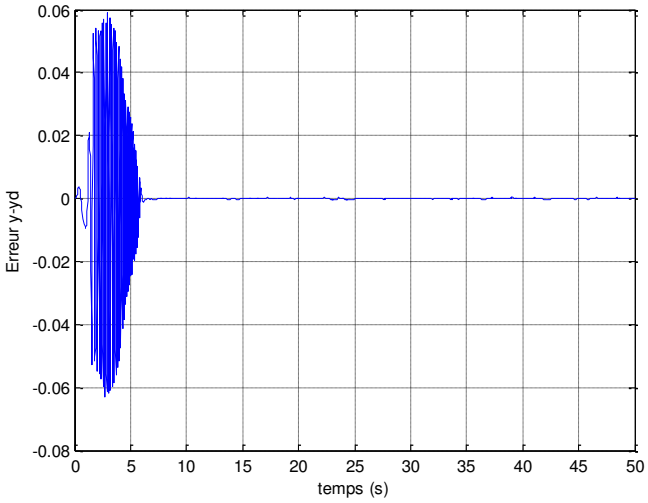


Figure II.20. Erreur de poursuite d'une trajectoire sinusoïdale.

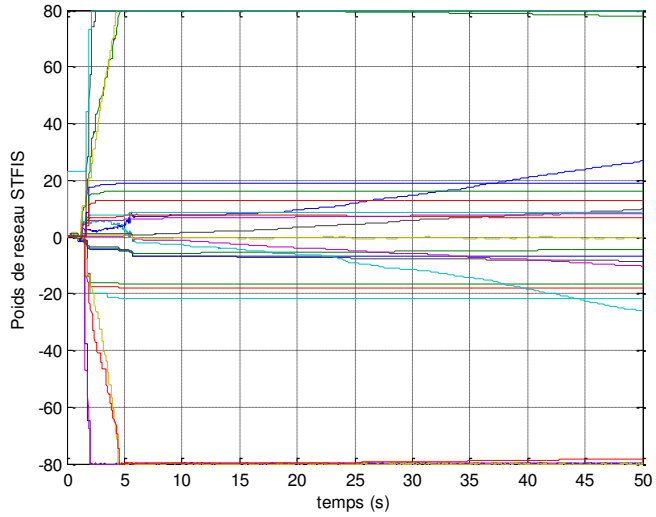


Figure II.21. Evolution des poids pour trajectoire sinusoïdale.

Suivant les résultats de la méthode STFIS obtenu, l'évolution de la sortie est chaotique au début d'apprentissage, puis elle converge au même temps que les poids convergent vers des valeurs asymptotiques (figures II.19 ; II.20 et II.21). Cette convergence est due à l'emploi de fonction coût regroupant une erreur quadratique et un terme de régression.

Après apprentissage, les poids sont convergés; nous avons testé la commande pour deux consignes (un signal sinusoïdal et un autre aléatoire).

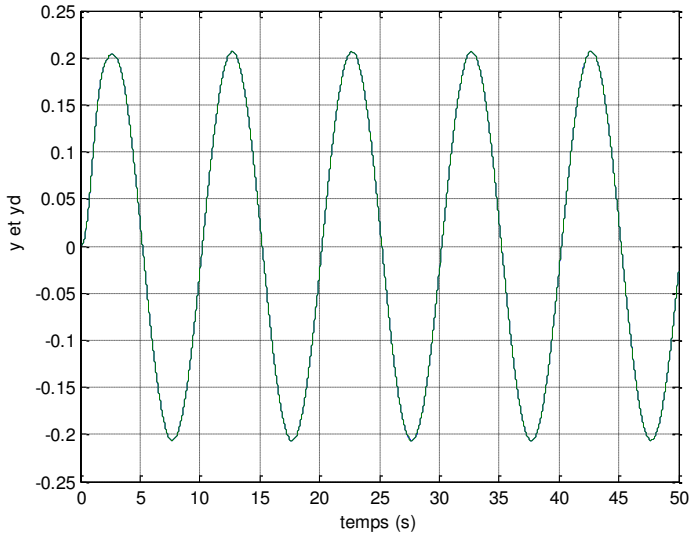


Figure II.22. Poursuite d'une trajectoire sinusoïdale.

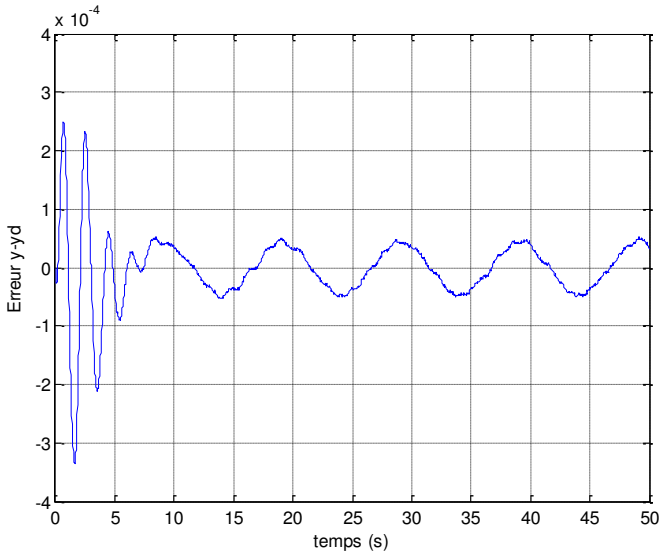


Figure II.23. Erreur de poursuite d'une trajectoire sinusoïdale.

D'après les figures II.22, II.23, II.24 et II.25, nous constatons que les poids restent stables. La sortie suit bien la consigne avec une erreur de poursuite très petite. Comme remarque, qu'on peut donner des connaissances appropriées concernant les poids au début d'apprentissage pour éviter l'évolution chaotique de la sortie.

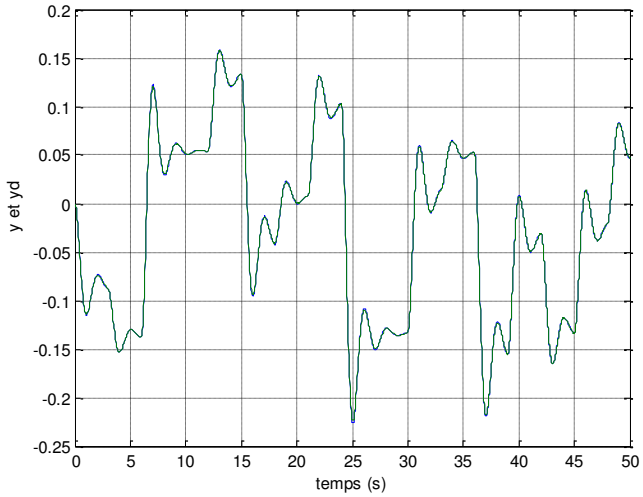


Figure II.24. Poursuite d'une trajectoire aléatoire.

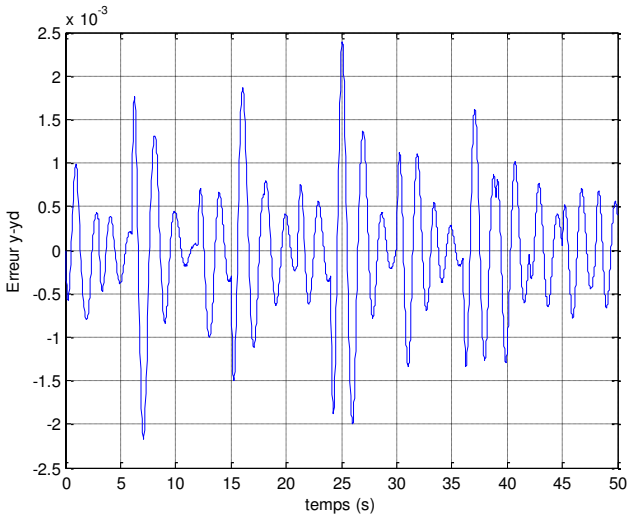


Figure II.25. Erreur de poursuite d'une trajectoire aléatoire.

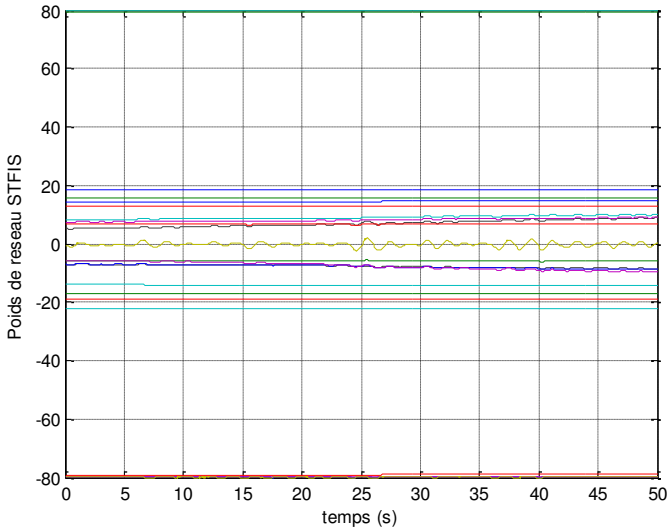


Figure II.26. Evolution des poids.

II.7.3. Comparaison entre la commande STFIS et les autres approches ANFIS et RN

Afin de vérifier l'amélioration de performance entre notre approche et les autres approches ANFIS et réseaux de neurones, nous avons appliqué les trois approches sur le modèle masse ressort, et nous avons obtenu les résultats suivants :

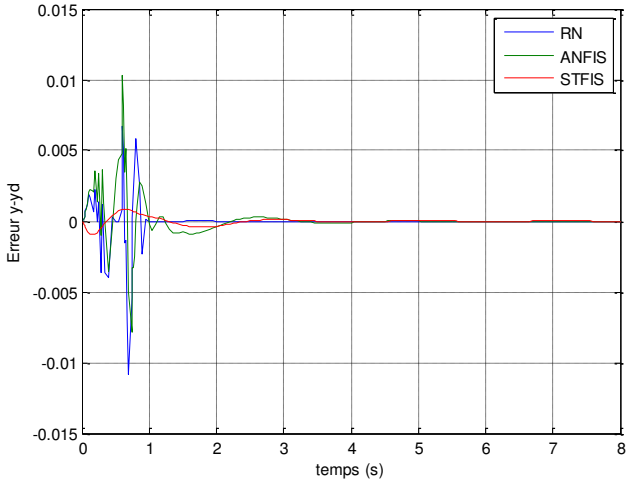


Figure II.27. Comparaison entre les trois méthodes : RN, ANFIS et STFIS.

D'après la figure II.27, nous constatons que notre approche utilisant le contrôleur STFIS donne une réponse meilleure, car l'erreur de poursuite de sortie est très petite et fine pour le cas de la commande STFIS par rapport aux autres commandes ANFIS et RN.

II.8. CONCLUSION

Nous avons présenté dans ce chapitre la théorie des ensembles flous ainsi que les outils mathématiques nécessaires à leurs manipulations. Le recours à des systèmes d'inférence floue pour générer les comportements élémentaires s'est révélé à la fois simple et efficace. Cependant, on peut toujours craindre que les règles déduites d'une simple expertise humaine soient plus ou moins largement sous-optimales. Nous avons opté d'utiliser des systèmes neuro-flous, comme le réseau neuro-flou ANFIS et le réseau neuro-flou STFIS. Le réseau STFIS utilise une méthode d'optimisation par descente de gradient pour effectuer l'optimisation des paramètres de système d'inférence flou caractérisant les conclusions des règles floues. L'optimisation est effectuée entièrement en ligne. Nous avons appliqué ces techniques pour la commande d'un système masse-ressort. Les résultats sont validés par des simulations sous MATLAB. Nous avons pu vérifier par comparaison entre les commandes (la commande par réseaux de neurones, la commande ANFIS et la commande STFIS) que le contrôleur neuro-flou STFIS présente de bons résultats, malgré la différence de stratégie

d'apprentissage. L'objectif du chapitre suivant est de proposer un réseau neuro-flou robuste, combinant les avantages des neuro-flous et ceux de la commande par mode de glissement.

CHAPITRE

III

LA COMMANDE NEURO-FLOUE ROBUSTE ‘STFSM’

III.1. INTRODUCTION

Connu par sa robustesse et sa simplicité de mise en œuvre, le mode glissant a été largement utilisé pour commander une large classe de systèmes non linéaires. Il s'agit de définir une surface dite de glissement en fonction des états du système de façon qu'elle soit attractive. La commande globale synthétisée se compose de deux termes : le premier permet l'approche jusqu'à cette surface et le second permet le maintien et le glissement le long de celle-ci vers l'origine du plan de phase. La commande globale ainsi construite permet d'assurer en plus de bonnes performances de poursuite, une dynamique rapide et un temps de réponse court [Utkin, 1977], [Khalil, 1996], [Perruquetti, 2002].

Cependant, cette loi de commande représente quelques inconvénients qui peuvent être résumés en deux points. Le premier réside dans la nécessité d'avoir des informations précises sur l'évolution du système dans l'espace d'état et les bornes supérieures des incertitudes et des perturbations. Or, la nature imparfaite des systèmes non linéaires rend difficile si ce n'est impossible de disposer d'une description analytique de la dynamique du système. Le second inconvénient réside dans l'utilisation de la fonction signe dans la loi de commande pour assurer le passage de la phase d'approche à celle du glissement. Ceci donne lieu au phénomène de broutement qui consiste en des variations brusques et rapides du signal de commande, ce qui peut exciter les hautes fréquences du processus et l'endommager [Slotine, 1991]. Pour remédier à ces problèmes, plusieurs approches ont été présentées dans la littérature [Spooner, 1996], [Wang, 1997], [Yoo, 1998], [Lin, 2002], [Yoshimura, 2015].

En effet, pour le premier inconvénient plusieurs travaux ont été focalisés sur la combinaison des modes glissants avec la commande adaptative où la dynamique du système imprécis est approximée à l'aide d'un réseau de neurones, d'un système flou ou une combinaison des deux (réseau neuro-flou) [Wang, 1997], [Yoo, 98], [Ha, 2001].

Pour le second inconvénient, l'introduction d'une bande de transition autour de la surface de glissement pour transformer la fonction signe en saturation, peut être une solution [Slotine, 1991], [Leu, 2013]. Néanmoins, une erreur statique subsiste, et un compromis entre la largeur de la bande et les variations de la commande s'impose. D'autres approches utilisant un système flou ont été également proposées dans la littérature [Wang, 1997], [Leu, 13].

A travers ce chapitre, nous allons proposer une commande pour traiter le cas où aucune connaissance de la dynamique du système n'est disponible. Les non linéarités inconnues sont approximées par un réseau STFIS (*Self Tuning Fuzzy Inference System*). Nous allons utiliser un réseau neuro-flou robuste, combinant les avantages des neuro-flous et ceux de la commande mode glissant [Chaouch, 2012].

Pour Cela, nous présenterons la commande par mode glissant; nous abordons quelques concepts de base ainsi que leur structure générale. Ensuite, nous présentons un aperçu sur le principe général de l'approche STFIS, qui sera introduite dans la commande des systèmes non linéaires.

Nous synthétisons une commande intelligente par mode glissant, par l'utilisation d'un réseau neuro-flou robuste (STFSM). Pour illustrer les performances de l'approche proposée, nous appliquerons cette technique à la commande d'un système non linéaire avec une application sur un pendule inversé.

III.2. CONTEXTE ET FORMULATION

La modélisation du processus est une étape primordiale dans la mise en œuvre d'un contrôleur dans les systèmes industriels. La nature non linéaire et la complexité de ces systèmes rendent leur description analytique avec des équations différentielles très difficile. En général, un système industriel ayant pour entrée (commande ou consigne) u et comme sortie y, peut être décrit par :

$$\begin{cases} x^{(n)} = F(x, \dot{x}....., x^{(n-1)}, u) \\ y = H(x, \dot{x}....., x^{(n-1)}) \end{cases} \quad \text{(III.1)}$$

où $F(x, \dot{x}....., x^{(n-1)}, u)$ et $H(x, \dot{x}....., x^{(n-1)})$ sont deux fonctions non linéaires continues, généralement partiellement ou totalement inconnues, $(x, \dot{x}....., x^{(n-1)})$ représente les états du système. Cette description ne permet pas la mise en œuvre de contrôleurs pour assurer la régulation ou l'asservissement. Afin de contourner ce problème, la linéarisation entrée-sortie a été largement utilisée; il s'agit de trouver, à l'aide des techniques de la géométrie différentielle, une relation explicite entre l'entrée du système et sa sortie [Slotine, 1991]. Dans ce cas, un système d'ordre n " affine dans la commande" peut être décrit par les équations différentielles suivantes :

$$\begin{cases} x^{(n)} = f(x, \dot{x}....., x^{(n-1)}) + g(x, \dot{x}....., x^{(n-1)})u \\ y = x \end{cases} \quad \text{(III.2)}$$

où $f(x, \dot{x}....., x^{(n-1)})$ et $g(x, \dot{x}....., x^{(n-1)})$ sont deux fonctions non linéaires continues.

Cette nouvelle description permet d'utiliser facilement les différentes approches basées sur la rétroaction pour résoudre les problèmes de poursuite de trajectoire ou de régulation. Dans la suite, on considère le système suivant :

$$\begin{cases} \ddot{x} = f(x,\dot{x}) + \ g(x,\dot{x})u + d \\ y = x \end{cases} \quad \text{(III.3)}$$

De plus, on supposera que les états x et $\dot{x}$ sont mesurables à l'aide de capteurs adéquats. Par ailleurs, pour garantir la contrôlabilité du système, on considère que la fonction $g(x,\dot{x})$ ne s'annule pas dans le domaine de fonctionnement ($g(x,\dot{x}) \neq 0$).

III.3. LA COMMANDE A STRUCTURE VARIABLE

Une autre approche à la commande robuste des systèmes non-linéaires incertains est la méthodologie de la commande à structure variable (CSV) avec le mode de glissement, cette approche est basée sur la théorie du système à structure variable (SSV). L'application de cette approche à la commande des systèmes non linéaires est l'originalité de plusieurs recherches [Utkin 1977], [Slotine 1991]. L'idée fondamentale est de conduire le signal d'erreur à une "surface de commutation"; après cela, le système est "en mode de glissement" et ne sera plus affecté par aucune incertitude de modélisation et/ou perturbations [Slotine 1991].

III.4. COMMANDE PAR MODE GLISSANT

Etant un cas particulier de la commande à structure variable, la commande par modes glissants (CMG) a été largement utilisée dans la littérature. Ce succès est dû à sa simplicité de mise en œuvre et à sa robustesse. Il s'agit de définir d'abord une surface dite de glissement qui représente la dynamique désirée, puis synthétiser une loi de commande qui doit agir sur le système en deux phases. Dans la première, on force le système à rejoindre cette surface, et dans la seconde phase on doit assurer le maintien et le glissement le long de cette surface pour atteindre l'origine du plan de phase, comme montré sur la figure (III.1).

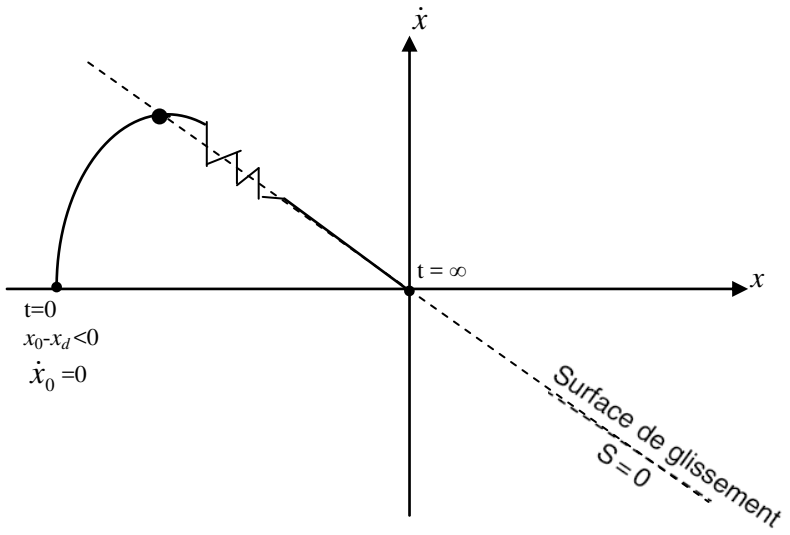


Figure III.1. Les deux phases de la CMG.

Slotine [Slotine ,1991] a proposé une forme générale qui consiste à définir une fonction scalaire des surfaces de glissement dans le plan de phase dans le but d'assurer la convergence d'une variable d'état x vers sa valeur de consigne x_d, cette fonction est donnée par l'équation:

$$S(x) = \left(\frac{\partial}{\partial t} + \lambda \right)^{r-1} e(x) \tag{III.4}$$

Avec :

$e(x) = x - x_d$: l'écart sur la variable à régler.

λ : une constante positive.

r: le degré relatif, qui représente le nombre de fois qu'il faut dériver la surface pour faire apparaître la commande.

Si l'on considère le système donné par l'équation (III.3), la surface de glissement peut être définie par :

$$S(x) = \dot{e} + \lambda e \tag{III.5}$$

Où λ est une constante positive représentant la pente de glissement. Il est à noter qu'en général, on donne une grande valeur à λ pour assurer l'attractivité ainsi que le maintien du système sur cette surface. Pour $f(x, \dot{x})$ et $g(x, \dot{x})$ parfaitement connues, la loi de commande par mode glissant peut être donnée par :

$$\begin{cases} u = u_{eq} + u_s \\ u_{eq} = g^{-1}(x, \dot{x}) \left[- f(x, \dot{x}) + \ddot{y}_d - \lambda \dot{e} \right] \\ u_s = -K_s \, sign(S) \end{cases} \tag{III.6}$$

Où u_{eq} est la commande équivalente permettant le maintien et le glissement le long de la surface S. u_s représente le signal de commutation assurant la convergence du système vers la surface. $\ddot{y}_d$ représente la dérivée seconde de la trajectoire de référence y_d.

La loi de commande (III.6) est certes robuste vis-à-vis des perturbations paramétriques et externes, mais présente quelques inconvénients majeurs :

i) l'utilisation du terme *sign(S)* dans le signal de commutation provoque le phénomène de broutement qui peut exciter les hautes fréquences et détériorer le système commandé. Pour résoudre ce problème, la fonction signe a été remplacée par la saturation. Néanmoins, cette substitution introduit une erreur statique qui persiste [Slotine, 1991].

ii) La mise en œuvre du signal de commutation nécessite la détermination de la constante K_s qui dépend des perturbations paramétriques et externes, ce qui est difficile, si ce n'est pas impossible. En général, on prend une valeur très grande pour assurer la stabilité, ce qui augmente les sollicitations au niveau de l'actionneur et amplifie gravement le phénomène de broutement. Parmi les solutions présentées dans la littérature, on remplace le signal de commutation par un système adaptatif flou ce qui a permis de résoudre à la fois le problème du gain K_s et celui du broutement [Kaiyu, 2000], [Zhang, 2006].

iii) Le troisième inconvénient concerne la nécessité de disposer d'une connaissance même partielle de la dynamique du système. Pour remédier à cet inconvénient, on peut approximer la dynamique du système pour synthétiser la loi de commande. Celle-ci peut être effectuée à l'aide d'un système flou [Wang, 1997], [Ha, 2001], [Noroozi, 2009], [Leu, 2013], [Chaouch, 2012].

III.5. CONDITION DE CONVERGENCE

La condition de convergence permet au système de converger vers la surface de glissement. Il s'agit alors de formuler une fonction scalaire positive $V(x)>0$ pour les variables d'états du système, qui est définie par la fonction de Lyapunov suivante:

$$V = \frac{1}{2}.S^2(x) \tag{III.7}$$

Pour que la fonction de Lyapunov soit décroissante, il suffit de s'assurer que sa dérivée soit négative. Ceci est vérifié par la relation suivante [Slotine, 1991]:

$$\dot{V} = \frac{1}{2}\frac{\partial}{\partial t}(S^2) \le -\eta.|S| \tag{III.8}$$

Ou $\eta > 0$

L'équation (III.7) exprime que le carré de la distance vers la surface, mesuré par $S^2(x)$, diminue tout le temps, contraignant la trajectoire du système à se diriger vers la surface dans les deux côtés.

III.6. COMMANDE NEURO-FLOUE ROBUSTE

On propose une commande neuro-floue mode glissante robuste (STFSM), en anglais '*Self Tuning Fuzzy Sliding Mode*', permettant de résoudre les problèmes du mode glissant classique cités précédemment et garantir de bonnes performances de poursuite même en présence de perturbations externes. Nous utiliserons un réseau neuro-flou 'STFIS' pour approximer la dynamique supposée totalement inconnue. Considérons les systèmes non linéaires où les équations dynamiques peuvent être écrites sous la forme canonique suivante :

$$\begin{cases} \ddot{x} = f(x,\dot{x}) + g(x,\dot{x})u + d \\ y = x \end{cases} \tag{III.9}$$

Où : $x \in R^n$ est le vecteur d'état,

$f(x,\dot{x})$ et $g(x,\dot{x})$ appartiennent à l'espace $R^n \to R$.

u et $d \in R$ sont respectivement, l'entrée de la commande et la perturbation extérieure.

III.6.1. Hypothèse

Les fonctions inconnues $f(x,\dot{x})$, $g(x,\dot{x})$ et $d(t)$ doivent satisfaire les conditions suivantes:

$$| f(x,\dot{x}) | \leq \mathrm{L}\,\gamma(x) \tag{III.10}$$

$$0 < g1 \leq g(x,\dot{x}) \tag{III.11}$$

$$d(t) \leq \delta \tag{III.12}$$

Avec L, $g1$ et δ des constantes positives inconnues. $\gamma(x)$ est une fonction positive connue, [Noroozi, 2009].

Supposons que la dynamique du système $f(x,\dot{x})$, $g(x,\dot{x})$ et les perturbations externes d, sont inconnues, la commande équivalente u_{eq} peut être estimée par le contrôleur neuro-flou (STFIS):

$$u_{eq} \hat{=} u_{stfis}(s,\dot{s}) \tag{III.13}$$

Le système neuro-flou STFIS est utilisé pour générer directement la commande u_{eq}. Ce contrôleur STFIS peut être représenté comme la cartographie des variables d'entrées $\dot{S}$ et S, et la variable de sortie u_{stfis} comme suit:

$$u_{stfis}(s,\dot{s}) = \sum^{m} \alpha_i w_i / \sum^{m} \alpha_i \tag{III.14}$$

La configuration proposée est représentée dans la figure III.2, où le contrôleur neuro-flou mode glissant (STFSM) est présenté par la loi de commande :

$$u = u_{stfis} + u_s \tag{III.15}$$

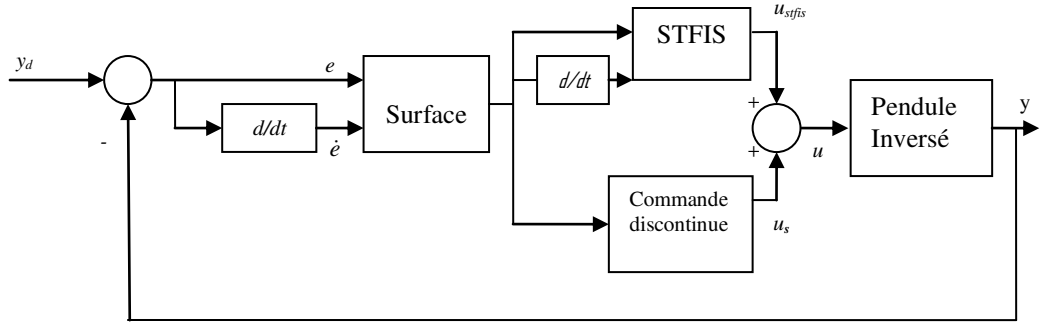


Figure III.2. Commande neuro-floue mode glissant.

Notons qu'il existe toujours une petite erreur ε entre la commande u_{eq} et la sortie u^*_{stfis}, la sortie réelle du STFIS. La commande de commutation u_s doit compenser cette erreur. Donc nous avons:

$$u_{eq} = u^{*}_{stfis}(s,\dot{s}) + \varepsilon = \sum^{m} \alpha_i w_i^{*} / \sum^{m} \alpha_i + \varepsilon \tag{III.16}$$

Où ε est l'erreur d'approximation; elle satisfait $|\varepsilon| < \upsilon$, avec υ une valeur faible et positive. u^{*}_{stfis} la sortie réelle du contrôleur STFIS; w_i^{*} sont les poids réels du STFIS.

Ici, nous définissons $\hat{u}_{stfis}$ l'estimation de u_{eq}:

$$\hat{u}_{stfis}(s,\dot{s}) = \sum^{m} \alpha_i \hat{w}_i / \sum^{m} \alpha_i \tag{III.17}$$

En prenant:

$$\zeta_i = \frac{\alpha_i}{\sum_{i=1}^{m} \alpha_i} \tag{III.18}$$

Nous obtenons:

$$u_{eq} = u^{*}_{stfis}(s,\dot{s}) + \varepsilon = w^{*T}\zeta + \varepsilon \tag{III.19}$$

$$\hat{u}_{stfis}(s,\dot{s}) = \hat{w}^{T}\zeta \tag{III.20}$$

Où $w=[w_1, w_2,..w_m]^T, \zeta=[\zeta_1, \zeta_2,...\zeta_m]^T$ et $\hat{w}$ l'estimation de w^*.

Le théorème ci-dessous nous permet de vérifier la stabilité de notre système en boucle fermée. De ce fait, il est possible de forcer le système non linéaire (III.9) dans la surface de mode de glissement $S = 0$, où $S=\dot{e}+\lambda e$. Ceci est possible si la condition $S\dot{S} < 0$ est vérifiée.

III.6.2. Théorème

Considérant le système dynamique incertain (III.9). Supposons que l'hypothèse (III.6.1) est satisfaite. La loi de commande sera sous la forme (III.15). Ainsi la stabilité asymptotique, de $S \to 0$ quand $t \to \infty$, pour le système commandé peut être vérifiée [Chaouch, 2012].

Preuve du théorème : Nous choisissons la fonction candidate de Lyapunov suivante:

$$V=\frac{1}{2}S^2+\frac{g(x,\dot{x})}{2\eta}\tilde{w}^T\tilde{w} \tag{III.21}$$

Nous avons:

$$\tilde{u}_{stfis}=\hat{u}_{stfis}-u_{eq}=\hat{u}_{stfis}-u^*_{stfis}-\varepsilon \tag{III.22}$$

On définit $\tilde{w}=\hat{w}-w^*$, nous pouvons obtenir:

$$\tilde{u}_{stfis}=\tilde{w}^T\zeta-\varepsilon \tag{III.23}$$

$$\begin{cases} u_{eq}=g^{-1}(x,\dot{x})\left[-f(x,\dot{x})+\ddot{y}_d-\lambda\dot{e}-d\right] \\ u_{eq}=g(x,\dot{x})^{-1}[g(x,\dot{x})u-\dot{S}] \end{cases} \tag{III.24}$$

$$\dot{S}=g(x,\dot{x})[\hat{u}_{stfis}+u_s-u_{eq}] \tag{III.25}$$

La dérivée dans le temps de V est:

$$\dot{V}=S\dot{S}+\frac{g(x,\dot{x})}{\eta}\tilde{w}^T\dot{\tilde{w}} \tag{III.26}$$

Remplaçons les équations (III.25) et (III.26), nous trouvons :

$$\dot{V}=g(x,\dot{x})\ \tilde{w}^T(S\zeta+\frac{1}{\eta}\dot{\tilde{w}})\ +S\,g(x,\dot{x})(u_s-\varepsilon) \tag{III.27}$$

La loi d'adaptation et la commande de commutation sont données :

$$\dot{\tilde{w}}=-\eta\,S\,\zeta \tag{III.28}$$

$$u_s=-K_s\,\mathrm{sgn}(S) \tag{III.29}$$

D'où:

$$\begin{aligned}\dot{V}&=-K_s|S|g(x,\dot{x})-\varepsilon\,S\,g(x,\dot{x}) \\ &\le -K_s|S|\le 0\end{aligned} \tag{III.30}$$

Ensuite, le choix du gain positif K_s peut être réalisé pour assurer la négativité de la dérivée temporelle de la fonction de Lyapunov candidate. Ce qui implique que la surface S =0 est atteinte en un temps fini. [Chaouch, 2012].

III.7. APPLICATION

Pour valider notre approche, nous avons choisi le modèle non linéaire du pendule inversé. Le système pendule inversé est représenté par la figure (III.3). En exerçant une force horizontale $u(t)$ sur le chariot, celui-ci se déplace en translation de x mètres et provoque une déviation du pendule de θ radians.

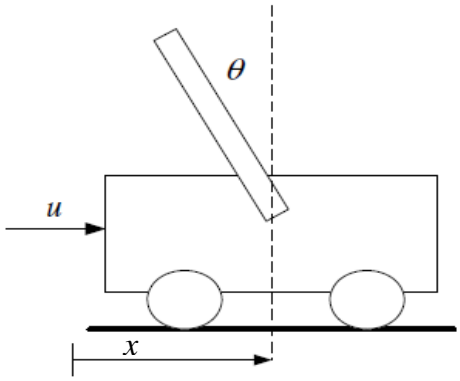


Figure III.3. Le pendule inversé

La modèle dynamique de pendule inversé est le suivant [Hung, 2007] :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \dfrac{(m_c+m)g\sin x_1 - mlx_2^2\cos x_1 \sin x_1}{l(4/3(m_c+m) - m\cos^2 x_1)} \\ + \dfrac{\cos x_1}{l(4/3(m_c+m) - m\cos^2 x_1)} u(t) + d(t) \end{cases} \quad \text{(III.31)}$$

$$\begin{cases} \dot{x}_3 = x_4 \\ \dot{x}_4 = \dfrac{-4/3mlx_2^2\sin x_1 + mg\cos x_1 \sin x_1}{4/3(m_c+m) - m\cos^2 x_1} \\ + \dfrac{4}{3.(4/3(m_c+m) - m\cos^2 x_1)} u(t) + d(t) \end{cases} \quad \text{(III.32)}$$

Où:

x_1 est la position angulaire θ,

x_2 est la vitesse angulaire,

x_3 est la position du chariot x,

x_4 est la vitesse linéaire du chariot,

$g = 9.8 m/s^2$,

m_c est la masse du chariot avec m_c =1kg,

m est la masse du pendule avec $m = 0.1kg$,

l = 0.5m est la longueur du pendule centre de masse,

$d(t)$ est la perturbation extérieure.

En premier lieu, on ne s'intéresse qu'au contrôle de l'angle du pendule, le système (III.31) est écrit sous la forme suivante [Liu, 2006] :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f_1(x,t) + g_1(x,t)u(t) + d(t) \\ y(t) = x_1(t) \end{cases} \quad \text{(III.33)}$$

Avec : $x = [x_1, x_2]^T$ le vecteur d'état, $f_1(x, t)$ et $g_1(x, t)$ des fonctions non linéaires du vecteur d'état décrivant la dynamique du système, et u la commande.

$$f_1(x,t) = \frac{(m_c + m)g \sin x_1 - mlx_2^2 \cos x_1 \sin x_1}{l(4/3(m_c + m) - m\cos^2 x_1)} \quad \text{(III.34)}$$

$$g_1(x,t) = \frac{\cos x_1}{l(4/3(m_c + m) - m\cos^2 x_1)} \quad \text{(III.35)}$$

Pour évaluer les performances de la commande STFSM, nous avons procédé à une série de simulations. Nous avons utilisé la loi de commande (III.15) présentée sur la figure (III.2), avec deux entrées, la surface $S = \dot{e} + \lambda e$ et sa dérivée, avec $e = y_d - y$ est l'erreur de poursuite.

Nous avons cinq fonctions d'appartenance (GN, PN, Z, PP, GP); les fonctions utilisées sont de types gaussienne et sigmoïde (voir figure III.4). les paramètres du contrôleur STFSM sont : $\eta = 0.3$, $b = 0.9$, $\beta = 0.00001$, $\lambda = 5$, ϕ=0.2, K_s = 50. Condition initiale $x_1 = pi/10$.

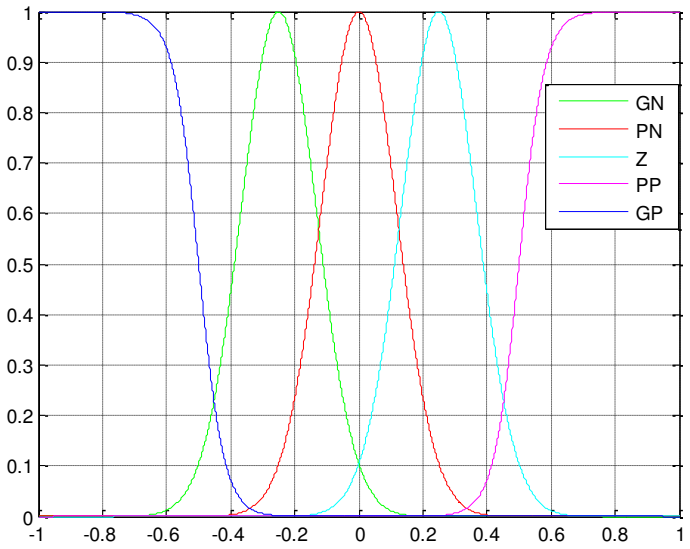


Figure III.4. Fonctions d'appartenances pour S et $\dot{S}$.

III.7.1. Résultats de simulation

Au début, en considère que le système est sans perturbation. Nous débutons l'apprentissage en initialisant la table des poids par une table de zéro, nous laissons les poids *w* évoluer avec la consigne "avec plusieurs créneaux" jusqu'à ce qu'ils convergent. Nous avons obtenu les résultats suivants :

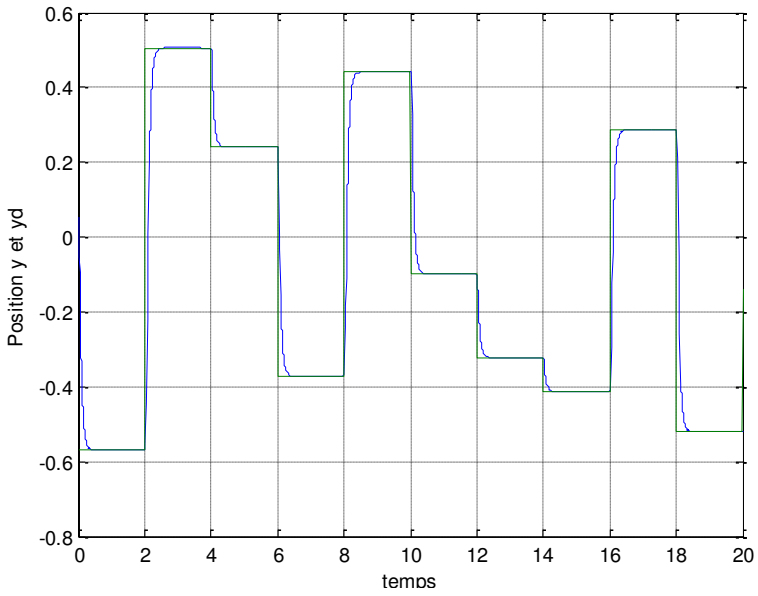


Figure III.5. La sortie du système *y* et la consigne *yd*.

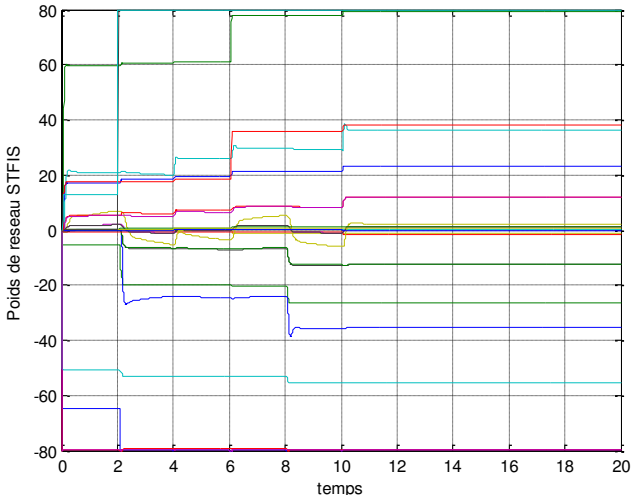


Figure III.6. L'évolution des poids (*w*).

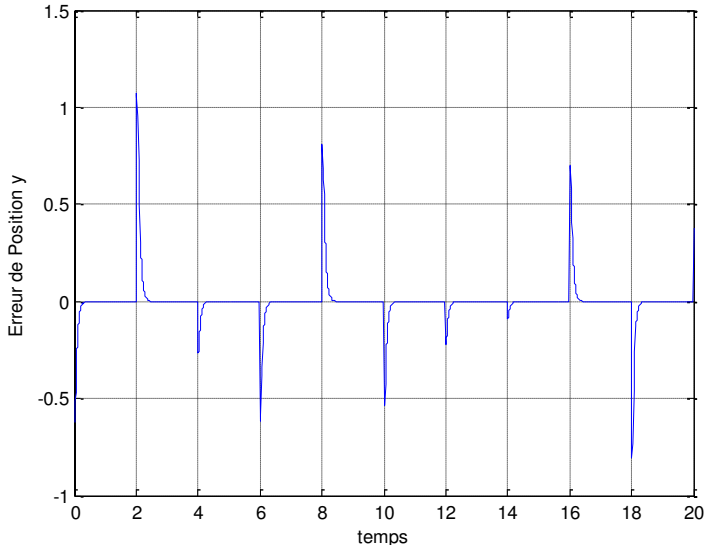


Figure III.7. L'erreur de sortie du système

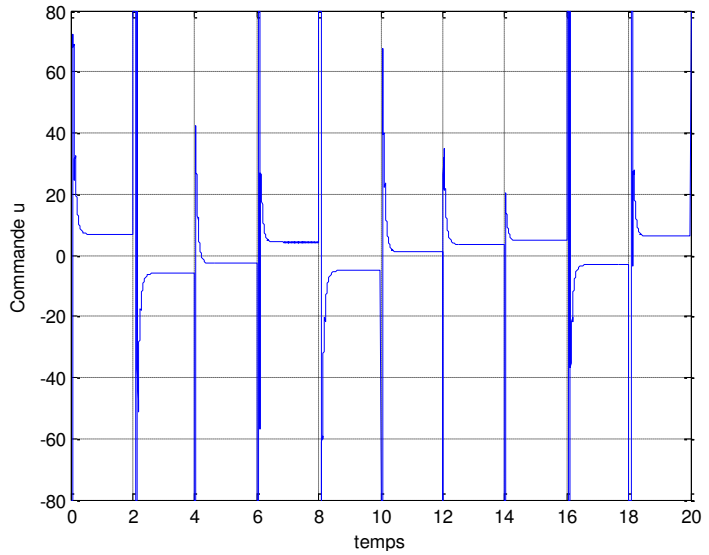


Figure III.8. La commande *u*.

En remarque que la sortie se régularise et converge vers la consigne (figure III.5). L'évolution des poids est quelque peu aléatoire en début d'apprentissage, puis elles convergent (figure III.6). L'erreur de position converge asymptotiquement vers zéro (figure III.7). D'après la figure III.8, L'allure de la commande ne dépasse pas les valeurs maximums (80 et -80). A la fin de l'apprentissage, nous avons obtenu la table suivante :

Table III.1. Les poids (w).

S / ΔS	GN	PN	Z	PP	PG
PG	-79.9991	-1.4526	-0.0151	0.8730	79.9977
PP	-55.3717	-12.5118	1.1205	12.1204	38.0113
Z	-79.6767	-35.2745	2.2893	36.3349	79.3691
PN	-26.3788	-12.5923	1.1433	11.8166	23.3659
GN	-79.9995	-1.3422	0.2268	1.3651	79.9896

D'après la table (III.1), on peut donner une interprétation linguistique en faisant pour cela convenir d'une échelle de traduction numérique-symbolique. Nous convenons d'attribuer le concept PL (Positive Large) aux valeurs numériques dans l'intervalle [65, 80], le concept PG (Positive Grand) aux valeurs numériques dans l'intervalle [2, 55], le concept PM (Positive Moyen) aux valeurs numériques dans l'intervalle [9, 15], le concept PS (Positive Petit) aux valeurs dans l'intervalle [3, 7], le concept PZ (Positive Zéro) aux valeurs dans l'intervalle[0.8, 2] et le concept Z (Zéro) aux valeurs dans l'intervalle[-0.5, 0.5], et de même dans le sens négatif. On obtient la table III.2.

S / $\dot{S}$	GN	PN	Z	PP	GP
GP	NL	NZ	Z	NZ	PL
PP	NG	NM	PZ	PM	PG
Z	NL	NG	PP	PG	PL
PN	NG	NM	PZ	PM	PG
GN	NL	NZ	Z	PZ	PL

Table III.2. Traduction des poids (w).

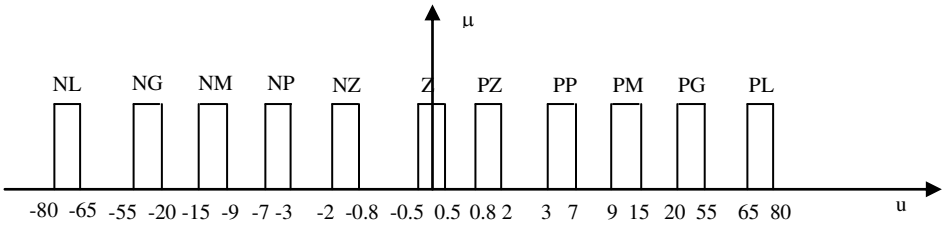


Figure III.9. Expression linguistique de la commande

III.7.2. Test de Robustesse

Après apprentissage et convergence des poids, nous avons testé la robustesse en injectant une perturbation $d(t) = 20\sin(2\pi t)$.

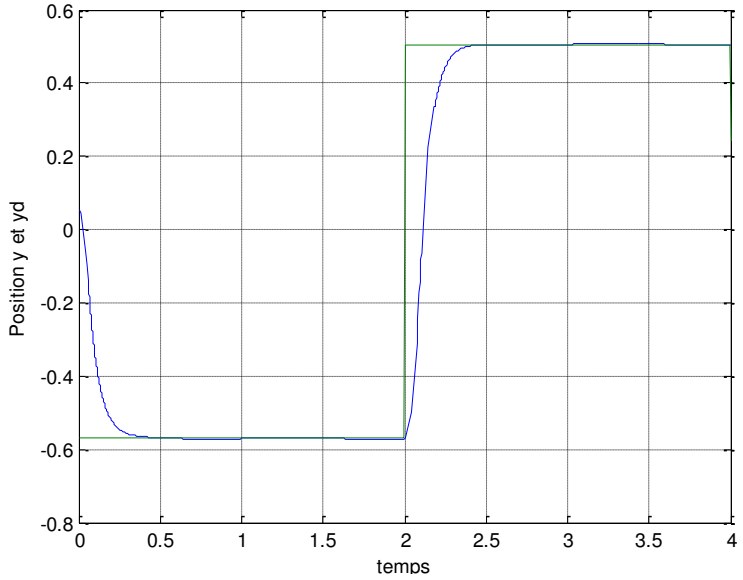


Figure III.10. La sortie *y* et la consigne *yd*, avec $d(t) = 20\sin(2\pi t)$

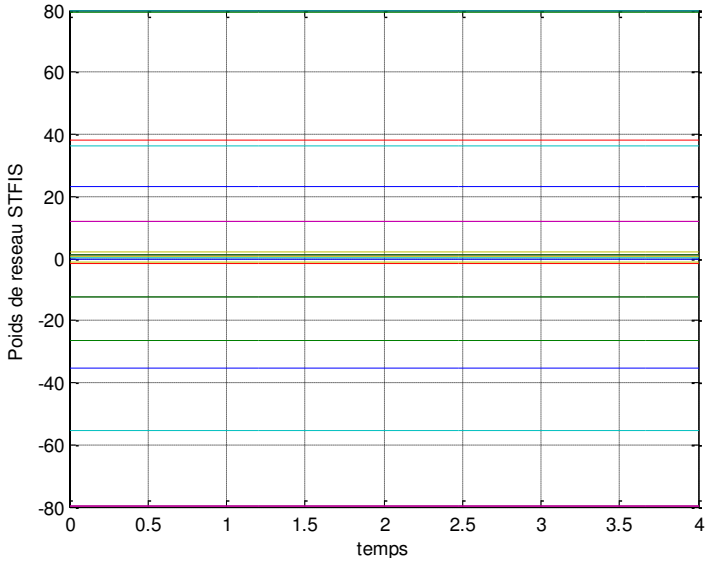


Figure III.11. L'évolution des poids (*w*).

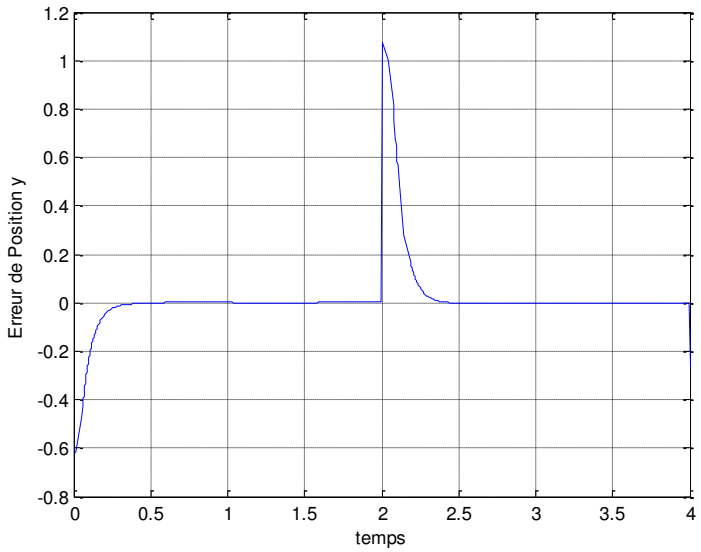


Figure III.12. L'erreur de sortie du système

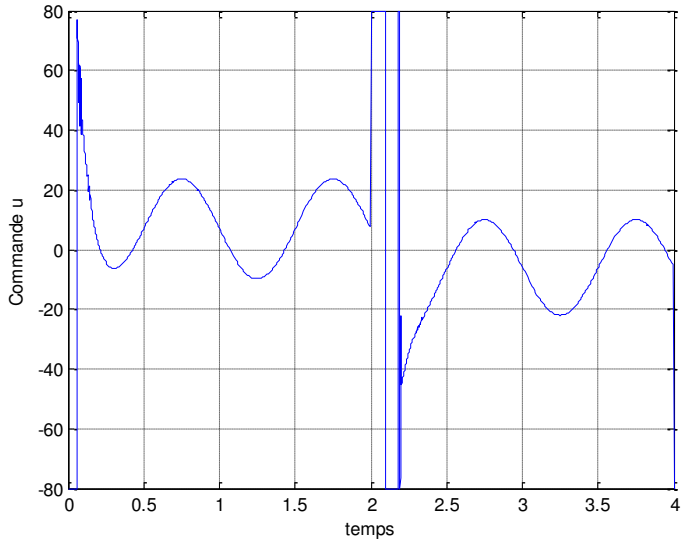


Figure III.13. La commande u.

En appliquant une perturbation extérieure, et d'après les figures III.11, III.12 III.13 et III.14, la sortie est stable et converge vers la consigne; les poids restent figés. Les résultats obtenus montrent bien que cette commande est robuste vis-à-vis des perturbations externes.

Avec un autre test de robustesse, vis-à-vis des variations paramétriques, nous avons appliqué une perturbation paramétrique avec ΔMc=0.1; Δl=0.1; Δm=0.1, et en plus une perturbation externe $d(t) = 10\sin(2\pi t)$.

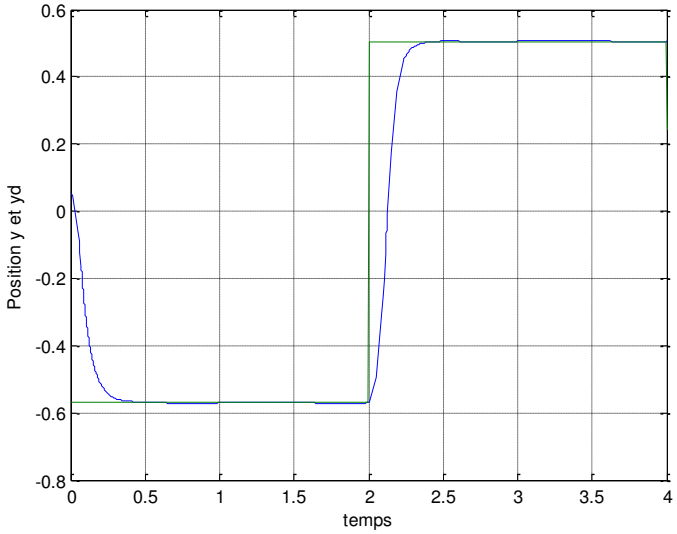


Figure III.14. La sortie *y* et la consigne *yd*

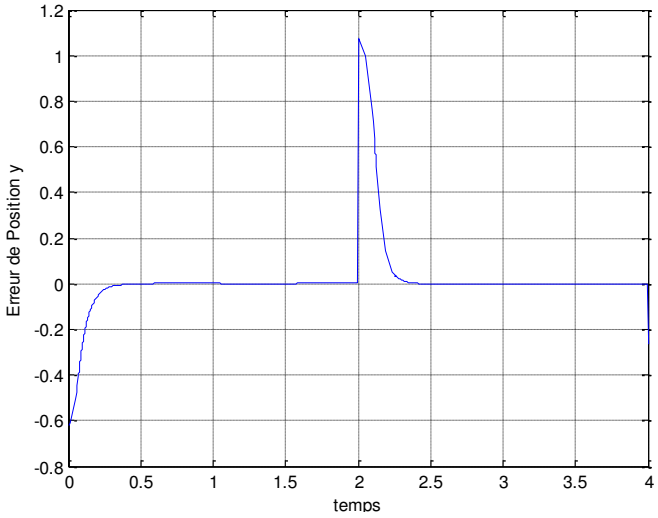


Figure III.15. L'erreur de sortie du système

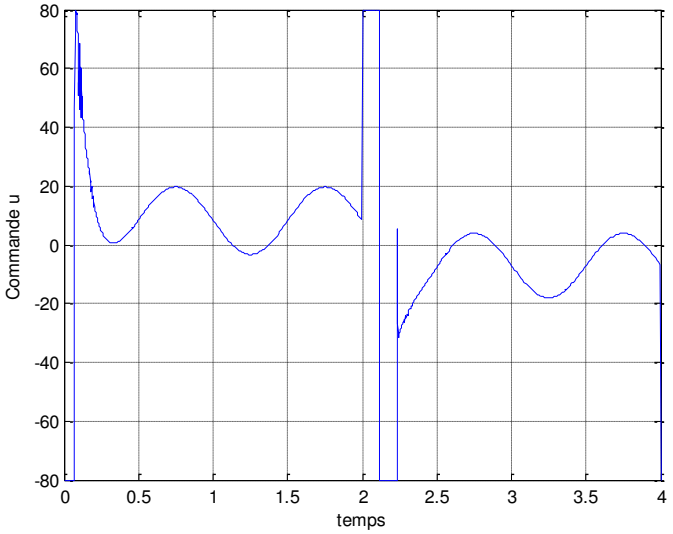


Figure III.16. La commande *u*.

D'après les figures III.14, III.15 et III.16, nous constatons que la sortie est stable et converge vers la consigne. Les figures obtenues montrent bien que cette commande est robuste vis-à-vis des perturbations paramétriques et extérieures.

III.8. COMMANDE NEURO-FLOUE MODE GLISSANT DECOUPLE

Considérons le système non linéaire représenté par le modèle suivant :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f_1(x,t) + g_1(x,t)u(t) + d_1(t) \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = f_2(x,t) + g_2(x,t)u(t) + d_2(t) \end{cases} \quad \text{(III.36)}$$

Avec : $x = [\, x_1, x_2\, x_3\, x_4]^T$ le vecteur d'état, $f_1\,(x, t)$, $f_2\,(x, t)$, $g_1(x, t)$ et $g_2(x, t)$ des fonctions non linéaires, u_1 et u_2 sont les commandes, d_1 et d_2 des perturbations externes. Les perturbations sont supposées être bornées.

III.8.1. Commande mode glissant découplé

Pour la conception du contrôleur, on définit première une surface de glissement :

$$s_1 = c_1\ x_1 + x_2 \quad \text{(III.37)}$$

Et une autre surface de glissement :

$$s_2 = c_2 x_3 + x_4 \quad \text{(III.38)}$$

Puis, sur la base de la théorie de mode de glissement, présentée dans la section précédente, nous pouvons choisir une loi de commande suivante :

$$u_1 = u_{eq1} + K_1\, sat(S_1) \quad \text{(III.39)}$$

Ou

$$u_2 = u_{eq2} + K_2\, sat(S_2) \quad \text{(III.40)}$$

De toute évidence, si l'on utilise la commande $u = u_1$ pour contrôler le système, seulement les états x_1 et x_2 s'approcheront de zéro asymptotiquement le long du hyperplan S_1. Et, si la commande $u = u_2$ est utilisée pour contrôler le système, seulement les états x_3 et x_4 vont glisser asymptotiquement. En d'autres termes, le dispositif de commande de mode de glissement peut seulement contrôler le pendule ou le chariot.

Pour traiter ce problème, Ji-Chang Lo a développé un contrôleur appelé contrôleur à mode glissant découplé (SMD), qui peut contrôler à la fois le pendule et le chariot [Ji-Chang Lo]. L'objectif de ce contrôle est de réaliser une commande basée sur le mode de glissement découplé, de sorte que le système en boucle fermée est globalement stable; pour ce faire toutes les variables d'état sont uniformément bornées et chacune converge asymptotiquement

vers son point d'équilibre. Ce contrôleur à mode de glissement découplé est réalisé comme suit.

La surface S1 est réécrite sous la forme:

$$S_1 = c_1(x_1 - z) + x_2 \quad \text{avec } c_1 > 0 \tag{III.41}$$

Où, z est une valeur transférée de S_2 :

$$S_2 = c_2 x_3 + x_4 \quad \text{avec } c_2 > 0 \tag{III.42}$$

La commande est donnée par :

$$u = u_1 = u_{eq1} + K_1 \, sat\left(S_1 / \Phi_1\right) \tag{III.43}$$

Et

$$z = sat\left(S_2 / \Phi_2\right).\, z_{upper} \text{ avec } \; 0 < z_{upper} < 1 \tag{III.44}$$

L'objectif de la loi de commande est changé, au lieu de converger asymptotiquement vers $x_1 = 0, x_2 = 0$, la convergence sera vers $x_1 = z, x_2 = 0$.

III.8.2. Commande neuro-floue mode glissant découplé

Le contrôleur neuro-flou mode glissant découplé (STFSMD), donné par la figure (III.17), est présenté par la loi de commande suivante [Khelfi, 2013]:

$$u = u_1 = u_{stfis} + K_1 \, sat\left(S_1 / \Phi_1\right) \tag{III.45}$$

Le système neuro-flou STFIS est utilisé pour générer directement la commande équivalente u_{eq1} donnée dans l'équation (III.43).

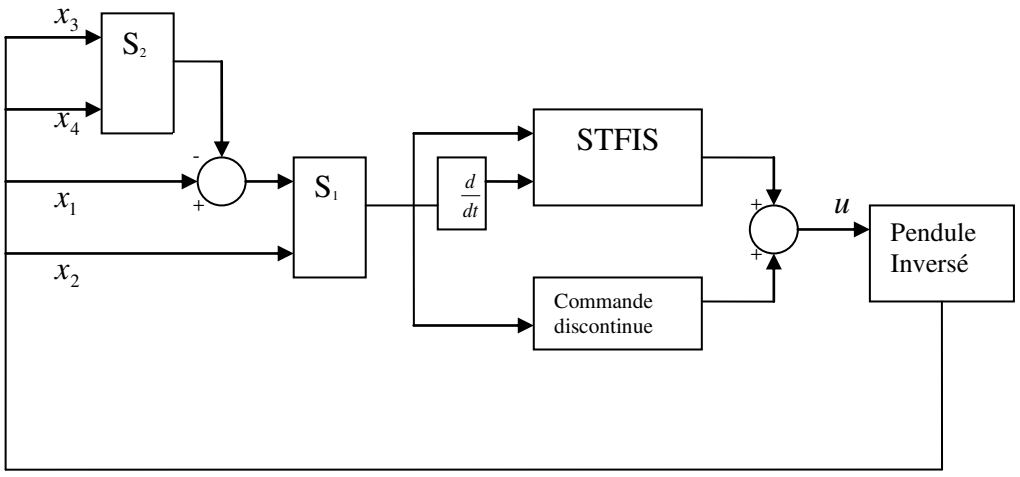


Figure III.17. Commande neuro-floue mode glissant découplé (STFSMD).

III.8.3. Résultats de simulations

Pour valider nos travaux, des simulations ont été réalisées sur le modèle dynamique complet du pendule inversé. Le modèle est écrit sous la forme (III.36) :

$$f_1(x,t) = \frac{(m_c+m)g\sin x_1 - mlx_2^2\cos x_1\sin x_1}{l(4/3(m_c+m)-m\cos^2 x_1)} \quad \text{(III.46)}$$

$$g_1(x,t) = \frac{\cos x_1}{l(4/3(m_c+m)-m\cos^2 x_1)} \quad \text{(III.47)}$$

$$f_2(x,t) = \frac{-4/3mlx_2^2\sin x_1 + mg\cos x_1\sin x_1}{4/3(m_c+m)-m\cos^2 x_1} \quad \text{(III.48)}$$

$$g_2(x,t) = \frac{4}{3.(4/3(m_c+m)-m\cos^2 x_1)} \quad \text{(III.49)}$$

L'architecture utilisée est présentée sur la figure.III.17, et les paramètres du contrôleur STFSMD choisis sont:

- Cinq fonctions d'appartenances {GN, PN, Z, PP, GP}, voir la figure III.4 ;
- Les entrées de STFIS sont $s_1(t)$ et $\dot{s}_1(t)$.

Les paramètres de la commande sont présentés dans le tableau III.3.

c_1	c_2	Φ_1	Φ_z	z_{upper}	k	$x_{1initial}$	$d(t)$	β	η	b
5	0.24	5	15	0.94	20	$\pi/3$	≤ 0.08	0.0001	0.3	0.7

Table III. 3. Les paramètres de la commande.

Les résultats obtenus sont représentés dans les figures ci-dessous.

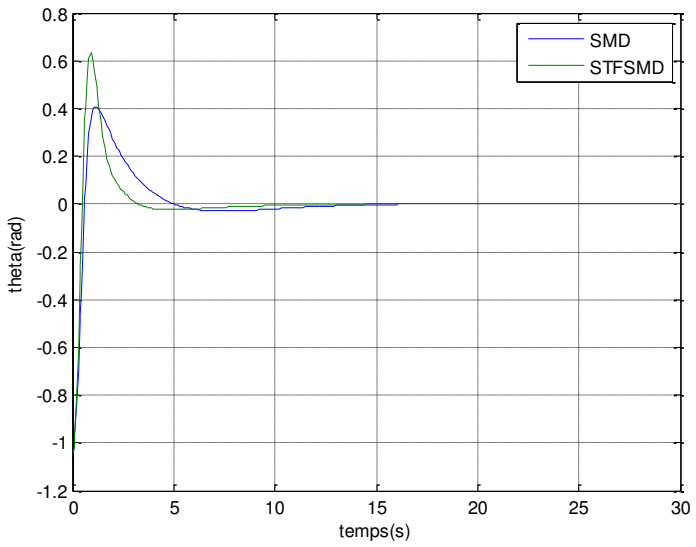


Figure III.18. La position angulaire du pendule

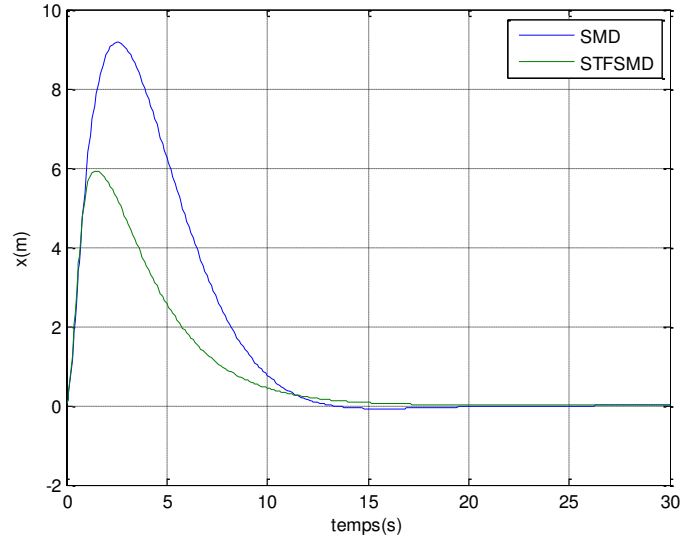


Figure III.19. La position de chariot

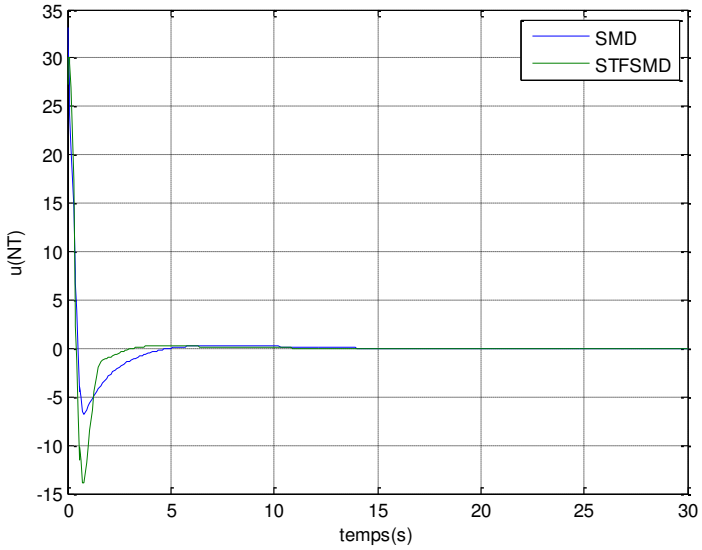


Figure III.20. La loi de commande.

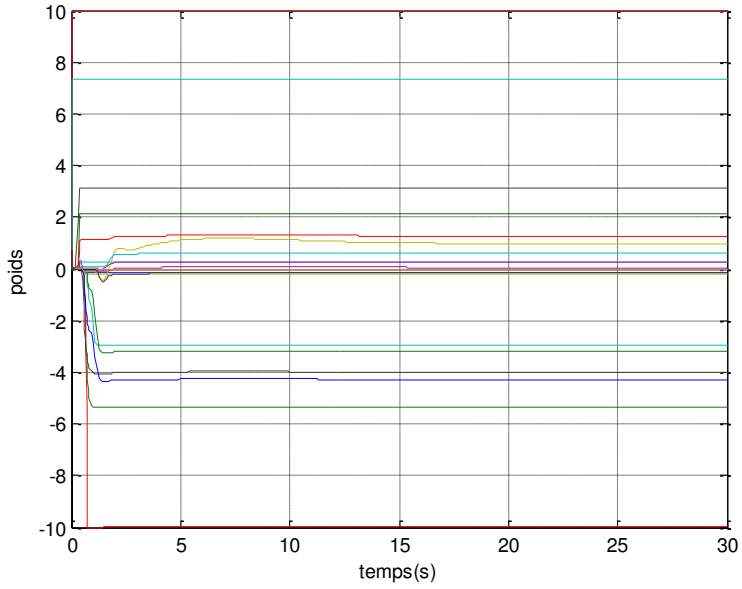


Figure III.21. Evolution des poids.

Les résultats obtenus sont issus d'un ensemble de simulations du système en boucle fermée, en utilisant une structure de commande neuro-floue mode glissant découplé (STFMD) ainsi que la commande mode glissant découplé (SMD), figures (III.18, 19, 20 et 21).

A travers ces résultats, nous constatons que le pendule et le chariot se stabilisent au point d'équilibre et les poids du STFIS convergent. En outre, la commande proposée peut non seulement diminuer le temps de réponse et atténuer le dépassement, mais aussi faire cela avec des performances similaires à la commande SMD.

Remarque : Pour éviter la situation où le chariot ne s'arrête jamais, le paramètre c_2 doit être bien choisi. En fait, la valeur de c_2 ne doit pas être trop grande, sinon le chariot sera toujours oscillant autour de l'origine.

III.9. CONCLUSION

Dans ce chapitre, nous avons utilisé la technique neuro-floue qui est introduite dans la commande des systèmes non linéaires. Nous avons présenté la commande par mode glissant. Nous avons abordé quelques concepts de base ainsi que leur structure générale. Ensuite, nous avons présenté le principe général de l'approche STFIS (Self Tuning Fuzzy Inference System). Nous avons synthétisé une commande intelligente par mode glissant, en utilisant un réseau neuro-flou (STFSM). L'approche STFIS avec ses capacités de modélisation des connaissances à priori utilisant une technique d'optimisation permettant l'ajustement en ligne de ces paramètres flous, et la commande par mode de glissement avec sa robustesse, ont été combinées pour la commande des systèmes non-linéaires. Nous avons appliqué cette technique pour la commande d'un pendule inversé.

Les résultats sont validés par des simulations sous MATLAB. Nous avons pu vérifier par ailleurs que le contrôleur neuro-flou robuste STFSM ainsi obtenu, avait une très bonne robustesse vis à vis des perturbations sur les signaux d'entrée. A notre connaissance, il n'existe pas de travaux sur la commande neuro-floue robuste STFSM appliquée aux systèmes non-linéaires, ce qui constitue l'une des originalités de notre travail.

Ensuite, nous avons traité le problème de contrôle pour le modèle complet du pendule inversé, soumis aux perturbations externes. Nous avons proposé une commande neuro-floue mode glissant découplé, en utilisant l'approche de la commande par mode glissant découplé. La loi de commande proposée permet de donner un meilleur résultat qu'avec la commande par mode glissant découplé.

CHAPITRE

IV

APPLICATION AUX ROBOTS MANIPULATEURS

IV.1. INTRODUCTION

L'objectif de ce chapitre est d'appliquer la commande neuro-floue robuste aux systèmes non linéaires d'ordre *n*, présentant des entrées et des sorties multiples (MIMO) soumises aux variations paramétriques et à des perturbations externes. La différence la plus évidente entre un système de type MIMO et un autre de type SISO, se trouve dans l'interaction complexe entre les paramètres, ce qui rend l'estimation et l'adaptation du premier plus compliquées. Cette complexité explique pourquoi certaines des approches traditionnelles du contrôle pour un système SISO ne sont pas appropriées à être exécutées directement pour le cas de MIMO. Pour illustrer cette approche, nous présenterons la commande Slotine appliquée aux robots manipulateurs.

La première partie de ce chapitre est consacrée à la présentation du modèle dynamique des robots manipulateurs [Canudas, 1992], [Sciavicco, 2000] et à la commande de Slotine *et* Li, basée sur la théorie de Lyapunov. Cette loi ne cherche ni la linéarisation ni le découplage de la dynamique du système; elle cherche seulement la stabilité asymptotique globale (parfois exponentielle) du système. Ensuite, nous présenterons la commande de Slotine neuronale, où nous avons utilisé les réseaux de neurones pour trouver les estimés du modèle dynamique du robot manipulateur. Après, nous proposerons la commande STFSM, où le réseau neuro-flou STFIS est proposé en association avec la commande du mode de glissement pour la commande de chaque articulation du robot.

Dans la deuxième partie et afin de valider l'étude théorique de ce chapitre, nous simulerons les différentes lois de commande en les appliquant au modèle dynamique du robot à deux degrés de liberté, ensuite sur un robot SCARA à trois degrés de liberté.

IV.2. MODELE DYNAMIQUE DES ROBOTS MANIPULATEURS

Le modèle dynamique exprime les couples des différents bras du robot en fonction des positions, vitesses et accélérations articulaires, et les efforts exercés par l'organe terminal sur l'environnement [Siciliano, 2009], [Spong, 1989], [Lewis, 1993]. Ce modèle est représenté par la relation suivante :

$$\Gamma = M(q).\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F_s\dot{q} + F_v(\dot{q}) + \Gamma_d \qquad \text{(IV.1)}$$

avec :

Γ : le vecteur des couples appliqué à l'articulation $i=1,\ldots\ldots n$;

q : le vecteur des variables articulaires du robot manipulateur ;

$\dot{q}$: le vecteur des vitesses articulaires du robot manipulateur;

$\ddot{q}$: le vecteur des accélérations articulaires du robot manipulateur;

$M(q)$: la matrice symétrique d'inertie du robot,

$C(q, \dot{q})$: le vecteur des forces non linéaires de Coriolis et centrifuges,

$G(q)$: le vecteur des forces de gravité,

$F_s\,\dot{q}$ et $F_v(\dot{q})$ désignent respectivement les matrices des coefficients de frottement sec et visqueux des articulations du robot.

$\Gamma_d \in R^n$: le vecteur des couples des articulations correspondant à l'effort extérieur.

IV.3. PROPRIÉTÉS STRUCTURELLES DU MODÈLE DYNAMIQUE

Dans cette partie, nous allons présenter les différentes propriétés des matrices qui constituent le modèle dynamique du robot manipulateur, car la connaissance de ces propriétés nous permet l'élaboration d'une commande simple et efficace du robot.

IV.3.1. Propriétés de la matrice d'inertie M(q)

La matrice M(q) est symétrique et définie positive, [Lewis. 93]. Le fait que, l'énergie du bras manipulateur est :

$$K = \frac{1}{2}\dot{q}^T M(q).\dot{q} \tag{IV.2}$$

La deuxième propriété est que $M(q)$ est donnée comme suit ;

$$M_m I \leq M(q) \leq M_M I \tag{IV.3}$$

Avec M_m et M_M des scalaires positifs, et I la matrice identité.

Si les articulations sont rotoïdes, les bornes M_m et M_M sont des constantes, car la variable articulaire q apparaît dans $M(q)$ uniquement à travers des sinus ou des cosinus, qui sont bornés par 1. Par ailleurs, si les articulations sont prismatiques, M_m et M_M peuvent être des fonctions scalaires de la variable articulaire q.

IV.3.2. Propriétés de la matrice des forces centrifuges et de Coriolis

La matrice des forces centrifuges et de Coriolis a quatre propriétés, qui sont :

1- La matrice $N(q, \dot{q}) = \dot{M}(q) - 2C(q,\dot{q})$ est antisymétrique [Spong, 1989], ce qui veut dire que les éléments n_{jk} de la matrice $N(q, \dot{q})$ vérifient l'égalité $n_{jk} = -n_{kj}$.

2- La matrice $C(q, \dot{q})$ vérifie la relation suivante [Lewis. 93] :

$$C(q,x)\, y = C(q,y)x \tag{IV.4}$$

Avec x, y les vecteurs de vitesse.

3- La norme de la matrice $C(q, \dot{q})$ vérifie la relation suivante :

$$\|C(q, \dot{q})\| \leq v_0 \|\dot{q}\| \tag{IV.5}$$

Où : $v_0 > 0$ est indépendant de q.

4- La matrice $C(q, \dot{q})\dot{q}$ vérifie la relation suivante [Lewis, 93] :

$$\|C(q, \dot{q})\dot{q}\| \leq C_0 \|\dot{q}\|^2 \tag{IV.6}$$

Pour une constante bornée $C_0 > 0$.

IV.3. 3. Propriétés du vecteur des forces de gravité

La norme du vecteur de gravité est bornée supérieurement ce que veut dire :

$$\| G(q) \| \leq g_b(q) \tag{IV.7}$$

Où : g_b est une fonction scalaire.

Pour une articulation rotoïde, g_b est une constante indépendante de la variable articulaire q, mais pour une articulation prismatique, g_b peut dépendre de q.

IV.3. 4. Propriétés du vecteur des frottements $F_t(\dot{q})$

Le frottement dans l'équation de mouvement du robot manipulateur, comme indiqué dans la formule (IV.1), est :

$$F_t(\dot{q}) = F_v \dot{q} + F_d(\dot{q}) \tag{IV.8}$$

Avec F_v la matrice des coefficients du frottement visqueux, et F_d le terme du frottement dynamique (frottement sec).

Ces coefficients sont parmi les paramètres les plus difficiles à les déterminer; de ce fait, la formule (IV.8) ne représente qu'une approximation mathématique pour leur influence sur le système [Lewis, 93].

IV.3. 5. Propriétés du vecteur Γd

Le vecteur Γ_d correspond aux incertitudes de modélisation dynamique et aux perturbations; il est borné par :

$$\|\Gamma d\| \leq d \tag{IV.9}$$

Où : d représente une constante scalaire.

IV.4. LA COMMANDE DE SLOTINE

Dans cette technique, Slotine [Slotine, 1991] a présenté une modification de la commande originale SV pour résoudre le problème de poursuite pour des robots manipulateurs décrits par l'équation (IV.1). Une erreur dépendante du terme de commutation est ajoutée à une commande non-linéaire qui utilise les valeurs estimées de la dynamique du robot. La stabilité asymptotique du système en boucle fermée est garantie sous certaines conditions sur les gains. La structure de la commande non-linéaire est fondamentalement celle proposée en [Slotine, 1987]. La commande proposée est définie comme :

$$\Gamma = \hat{M}\ddot{q}_r + \hat{C}\dot{q}_r + \hat{G} - Ksign(s) \tag{IV.10}$$

Avec :

$$\dot{q}_r = \dot{q}_d - \Lambda\tilde{q},\ \Lambda = diag(\lambda_1, ...\lambda_n) \tag{IV.11}$$

$$s = \dot{q} - \dot{q}_r = \dot{\tilde{q}} + \Lambda\tilde{q} \tag{IV.12}$$

Et :

$$K = diag(k_i),\ k_i > 0 \tag{IV.13}$$

Où $\hat{M}, \hat{C}$, et $\hat{G}$ sont respectivement les estimés constants des matrices $M(q), C(q,\dot{q})$ et $G(q)$. La variable s de l'équation (IV.12) est une combinaison linéaire des éléments d'erreur de poursuite. Le terme $sign(s)$ représente la fonction signe de la variable s, il est donné par:

$$sign\ (s) = (sign\ (s_1), \ldots, sing\ (s_n)) \tag{IV.14}$$

Et :

$$sign(s_i) = \begin{cases} 1 & \text{si } s_i > 0 \\ -1 & \text{si } s_i < 0 \end{cases} \tag{IV.15}$$

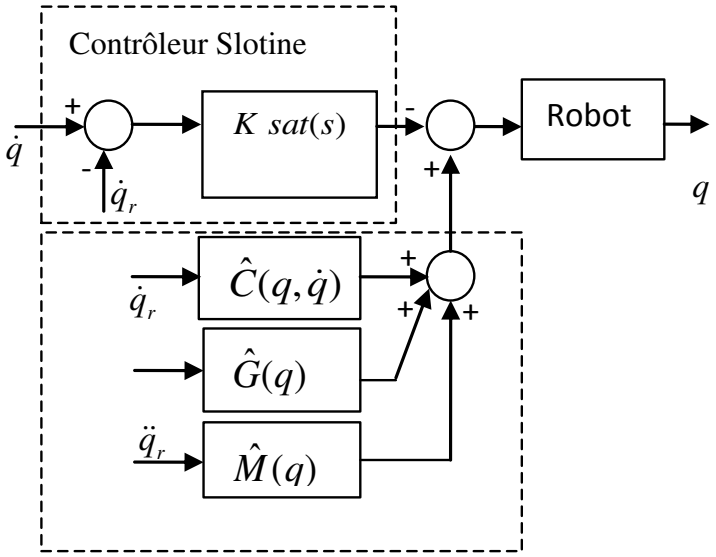


Figure IV.1. La structure de la commande Slotine.

L'application de cette loi de commande sur le dynamique du robot donne la dynamique du système en boucle fermée suivante:

$$M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=\hat{M}\ddot{q}_r+\hat{C}\dot{q}_r+\hat{G}-Ksign(s) \tag{IV.16}$$

Ensuite, et après quelques manipulations, nous obtenons:

$$M(q)\dot{s}+C(q,\dot{q})s=-\tilde{M}(q)\ddot{q}_r-\tilde{C}(q,\dot{q})\dot{q}_r-\tilde{G}(q)-Ksign(s) \tag{IV.17}$$

Où, $\tilde{M}(q)=M(q)-\hat{M}$, $\tilde{C}(q,\dot{q})=C(q,\dot{q})-\hat{C}$, et $\tilde{G}(q)=G(q)-\hat{G}$ sont les erreurs d'estimation de la dynamique du système. Elles sont considérées comme les incertitudes du système.

A ce stade, il est nécessaire de vérifier que la loi de commande proposée assure la stabilité asymptotique du système en boucle fermée.

Choisissons la fonction candidate de Lyapunov suivante:

$$V(s)=\frac{1}{2}s^T M(q)s \tag{IV.18}$$

La dérivée temporelle de cette fonction est :

$$\dot{V}(s)=s^T M(q)\dot{s}+\frac{1}{2}s^T \dot{M}(q)s \tag{IV.19}$$

L'évaluation de cette fonction le long de la trajectoire du signal d'erreur donne :

$$\dot{V}(s)=s^T\left(-C(q,\dot{q})s-\tilde{M}(q)\ddot{q}_r-\tilde{C}(q,\dot{q})\dot{q}_r-\tilde{G}(q)-Ksign(s)\right)+\frac{1}{2}s^T \dot{M}(q)s \tag{IV.20}$$

En utilisant la propriété antisymétrique de $\dot{M}(q)-2C(q,\dot{q})$, nous obtenons:

$$\dot{V}(s)=s^T\left(-\tilde{M}(q)\ddot{q}_r-\tilde{C}(q,\dot{q})\dot{q}_r-\tilde{G}(q)\right)-\sum_{i=1}^{n}k_i|s_i| \tag{IV.21}$$

Alors, un choix du gain k_i peut assurer la négativité de la dérivée de la fonction candidate de Lyapunov. Un simple choix de ce gain de commande peut être pris comme:

$$k_i \geq \left|\left(\tilde{M}(q)\ddot{q}_r+\tilde{C}(q,\dot{q})\dot{q}_r+\tilde{G}(q)\right)_i\right|+\eta_i\ , \quad (\eta_i>0) \tag{IV.22}$$

Par conséquent,

$$\dot{V}(s)\leq-\sum_{i=1}^{n}\eta_i|s_i| \tag{IV.23}$$

Ce qui implique que la valeur de la surface s=0 est atteinte dans un temps fini. Pour remédier au problème de broutement, Slotine [Slotine 1985] a modifié la commande originale en remplaçant la fonction signe dans la loi de commande (IV.10) par la fonction de saturation $sat(s/\varepsilon)$ avec $\varepsilon>0$, ainsi :

$$\text{sat}(s/\varepsilon)=\begin{cases} sign(s) & \text{si } |s|>\Phi \\ s/\varepsilon & \text{si } |s|\leq\Phi \end{cases} \tag{IV.24}$$

A partir de la commande ci-dessus, nous pouvons voir que le compensateur non-linéaire ne cherche pas à linéariser le système du robot, mais utilise les estimés du modèle dynamique du robot pour compenser les non-linéarités présentes dans le système. De ce fait, nous proposons d'utiliser les réseaux de neurones pour trouver les estimés du modèle dynamique du robot [Chaouch, 2012].

IV.5. LA COMMANDE DE SLOTINE NEURONALE

L'approche d'identification est basée sur les travaux de [Salem, 2007] et [Babuska, 2003], où une certaine connaissance sur le robot est exploitée. Les paramètres dynamiques du modèle du manipulateur sont modélisés par trois réseaux de neurones (MLP), le premier pour estimer la matrice d'inertie $M(q)$, le second réseau est utilisé pour estimer la matrice des forces centrifuges et de Coriolis $C(q,\dot{q})$, tandis que le dernier pour estimer le vecteur de gravité $G(q)$. La commande slotine neuronale peut être écrite sous la forme suivante:

$$\Gamma=\hat{M}(q)\ddot{q}_r+\hat{C}(q,\dot{q})\dot{q}_r+\hat{G}(q)-K\ \text{sgn}(s) \tag{IV.25}$$

Où $\hat{M}$, $\hat{C}$ et $\hat{G}$ sont les modèles neurales estimés; nous avons :

$$\hat{M}(q) = g_{nn1}(\varphi_M(k), \theta_M) \tag{IV.26}$$

$$\hat{G}(q) = g_{nn3}(\varphi_G(k), \theta_G) \tag{IV.27}$$

$$\hat{C}(q,\dot{q}) = g_{nn2}(\varphi_C(k), \theta_C) \tag{IV.28}$$

Avec :

$$\varphi_C(k) = (q_1,.....q_a, \hat{C}(q,\dot{q})_1,....., \hat{C}(q,\dot{q})_b) \tag{IV.29}$$

$$\varphi_M(k) = (q_1,.....q_a, \hat{M}(q)_1,....., \hat{M}(q)_b) \tag{IV.30}$$

$$\varphi_G(k) = (q_1,.....q_a, \hat{G}(q)_1,....., \hat{G}(q)_b) \tag{IV.31}$$

Où, $g_{nn}(\varphi(k),\theta)$ est un réseau neuronal de paramètres θ, et $\varphi(k)$ est un vecteur de régression qui contient les entrées du réseau.

Bien que les réseaux de neurones sont des approximateurs universels idéaux, ils sont limités dans le cas où nous n'aurions pas bien préparé la base de données d'apprentissage; dans ce cas-ci l'apprentissage est offline. Pour cela, nous avons utilisé une autre approche online basée sur le réseau neuro-flou (STFIS).

IV.6. LA COMMANDE NEURO-FLOUE ROBUSTE 'STFSM'

Dans cette section, le réseau neuro-flou STFIS est proposé en association avec la commande de mode de glissement pour chaque articulation du robot. Le contrôleur neuro-flou mode glissant (STFSM) est présenté par la loi de commande :

$$\Gamma = u_{stfis} + u_s \tag{IV.32}$$

Avec :

$$u_s = -K\,sat(s) \tag{IV.33}$$

$$s = \dot{\tilde{q}} + \Lambda\tilde{q} \tag{IV.34}$$

Et :

$$K = diag(k_i),\ k_i > 0 \tag{IV.35}$$

Le terme *sat*(*s*) représente la fonction saturation de la variable *s*; il est donné par :

$$sat\,(s) = (sat\,(s_1),\ldots,\, sat\,(s_n)) \tag{IV.36}$$

Le réseau neuro-flou (STFIS) est utilisé pour générer directement la commande équivalente pour chaque articulation du robot.

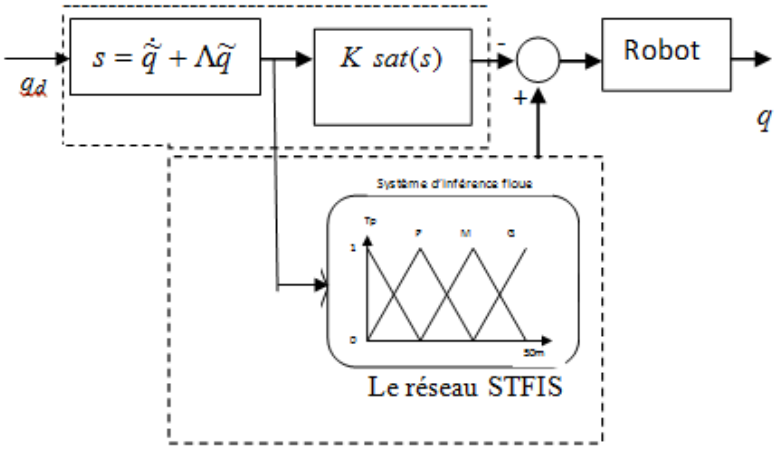


Figure IV.2. La structure de la commande neuro-floue robuste.

Afin de valider l'étude théorique que nous avons présentée dans ce chapitre, plusieurs simulations vont être effectuées dans cette section, où nous allons comparer les différentes lois de commande. Les simulations ont été réalisées sur deux robots, le premier est un robot manipulateur à deux degrés de liberté, et le second est un robot SCARA à trois degrés de liberté.

IV.7. APPLICATION AU ROBOT A DEUX DEGRES DE LIBERTE

IV.7.1. Modèle et trajectoire utilisés

Le premier modèle du robot manipulateur rigide utilisé dans la simulation est le modèle du robot à deux degrés de liberté, dont le modèle dynamique est donné par [Berguis 1993], [khelfi, 1996]:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \Gamma \tag{IV.38}$$

Avec:

$$M(q) = \begin{bmatrix} 8.77 + 1.02\cos(q_2) & 0.76 + 0.51\cos(q_2) \\ 0.76 + 0.51\cos(q_2) & 0.62 \end{bmatrix} \tag{IV.39}$$

$$C(q,\dot{q}) = \begin{bmatrix} -0.5\sin(q_2)\dot{q}_2 & -0.5\sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ 0.5\sin(q_2)\dot{q}_1 & 0 \end{bmatrix} \tag{IV.40}$$

$$G(q) = 10\begin{bmatrix} 7.6\sin(q_1) + 0.63\sin(q_1+q_2) \\ 0.63\sin(q_1+q_2) \end{bmatrix} \quad \text{(IV.41)}$$

La trajectoire utilisée dans la simulation est une interpolation polynomiale de degré cinq, dont la position est de la forme suivante :

$$q_d(t)=q^i+(10(\frac{t}{t_f})^3-15(\frac{t}{t_f})^4+6(\frac{t}{t_f})^5)D \quad \text{(IV.42)}$$

$$D=q^f-q^i \quad \text{(IV.43)}$$

Où, q^i, q^f et t_f sont respectivement la position initiale, la position finale et le temps final.

Avec :

$$q^i = (0 \quad 0)^T \quad \text{(IV.44)}$$

$$q^f = (1{,}1 \quad 1{,}5)^T \quad \text{(IV.45)}$$

Le vecteur d'état utilisé dans les simulations est :

$$x = [q_1 \quad q_2]^T \quad \text{(IV.46)}$$

IV.7.2. La commande de Slotine

La loi de commande est donnée par la relation suivante :

$$\Gamma=\hat{M}\ddot{q}_r+\hat{C}\dot{q}_r+\hat{G}-Ksat(s/\varepsilon) \quad \text{(IV.47)}$$

Où :

$$\dot{q}_r=\dot{q}_d-\Lambda\tilde{q},\ \Lambda=diag(\lambda_1,.....\lambda_n) \quad \text{(IV.48)}$$

$$s=\dot{q}-\dot{q}_r=\dot{\tilde{q}}+\Lambda\tilde{q} \quad \text{(IV.49)}$$

Les paramètres de simulation sont :

$$\hat{M}=I,\ \hat{C}=0,\ \hat{G}=0,\ \varepsilon=0.1,\ \Phi=0.8 \quad \text{(IV.50)}$$

Les matrices des gains sont :

$$\Lambda = \text{diag}\{60,\ 50\} \ \text{ et } \ K = \text{diag}\ \{650,\ 600\} \quad \text{(IV.51)}$$

Le vecteur d'état initial est :

$$x(0) = \begin{bmatrix} 0.1 & -0.1 \end{bmatrix}^T \tag{IV.52}$$

IV.7.3. La commande de Slotine Neuronale

La loi de commande est donnée comme suit :

$$\Gamma = \hat{M}\ddot{q}_r + \hat{C}\dot{q}_r + \hat{G} - Ksat(s/\varepsilon) \tag{IV.53}$$

Où, les trois matrices M(q), G(q) et $C(q,\dot{q})$ du modèle du robot sont identifiés par trois réseaux de neurones avec une couche cachée de 7, 10 et 5 neurones. Après apprentissage, les trois matrices sont utilisées dans la structure de la commande de Slotine comme le montre la Figure IV. 1.

Les matrices des gains sont :

$$\Lambda = \text{diag}\{50,\ 20\} \quad \text{et} \quad K = \text{diag}\ \{450,\ 400\} \tag{IV.54}$$

IV.7.4. Résultats de simulations

La poursuite et les erreurs de poursuite de la position et de vitesse pour les deux articulations sont présentées aux figures ci-dessous. Un bruit a été introduit dans la boucle de commande.

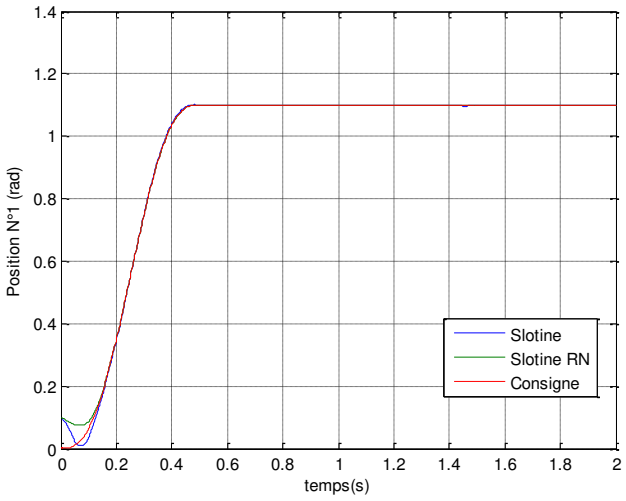


Figure IV.3. Poursuite de position de l'articulation 1.

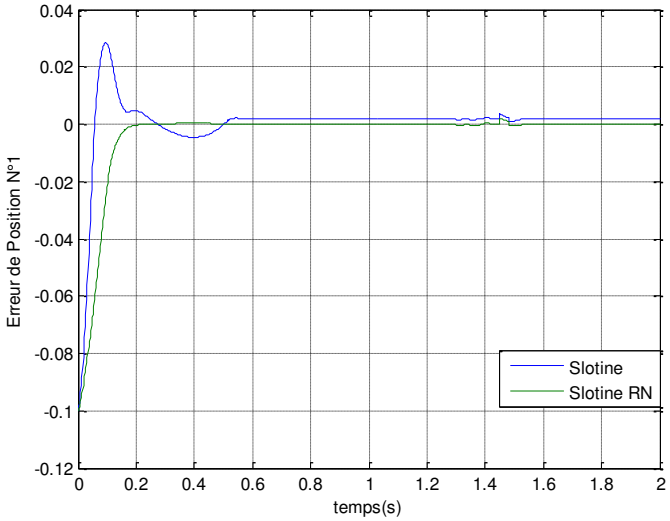


Figure IV.4. Erreur de poursuite de l'articulation 1.

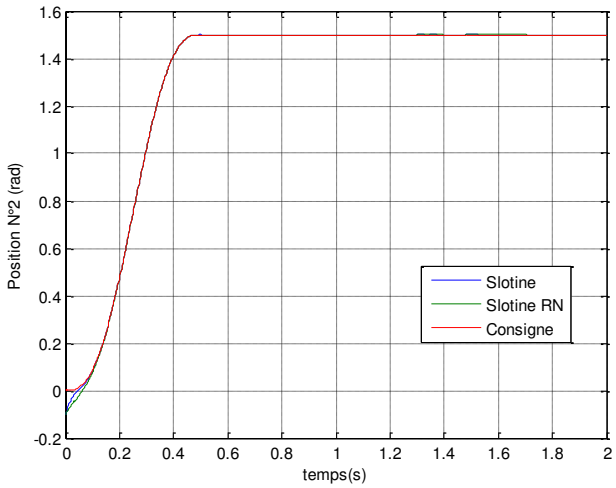


Figure IV.5. Poursuite de position de l'articulation 2.

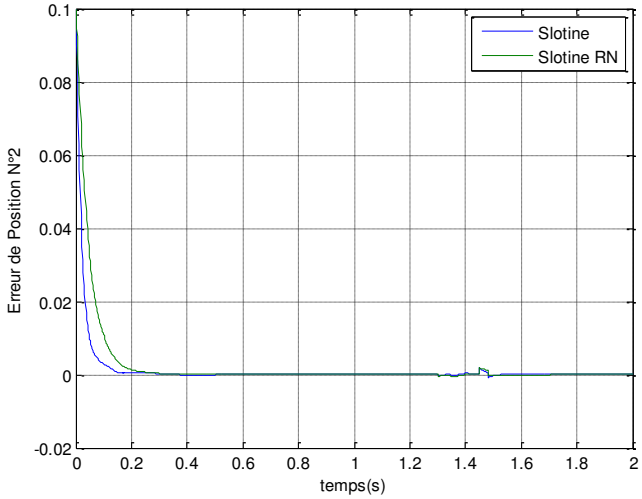


Figure IV.6. Erreur de poursuite de l'articulation 2.

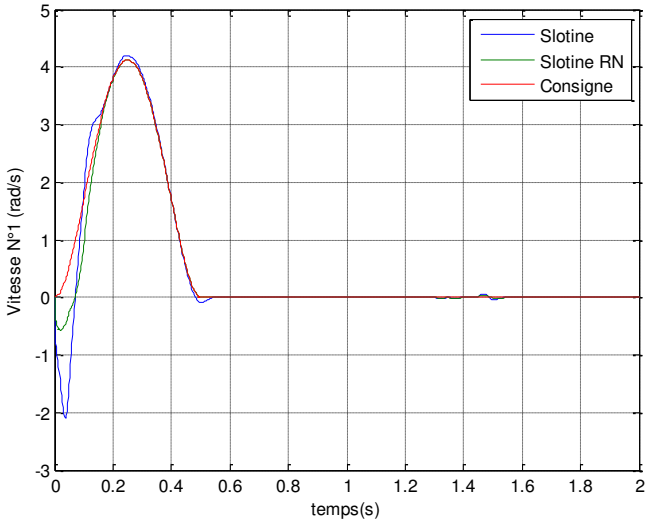


Figure IV.7. Poursuite en vitesse de l'articulation 1.

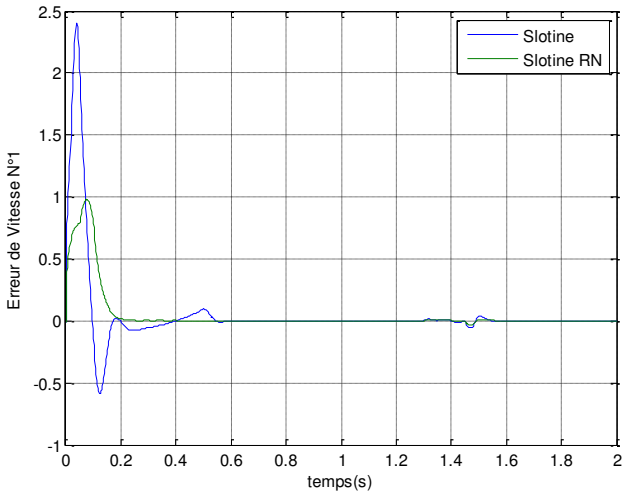


Figure IV.8. Erreur poursuite en vitesse de l'articulation 1.

On voit sur les Figures IV.3 et IV.5 que les positions suivent la trajectoire souhaitée. Les erreurs de poursuite convergent asymptotiquement vers zéro dans le cas de la commande Slotine neuronale, comme le montrent les figures IV.4 et IV.6. Les erreurs de poursuite présente une erreur statique dans le cas de la commande Slotine.

Nous remarquons que la commande Slotine neuronale présente une amélioration nette sur la réponse par rapport la commande de Slotine. La robustesse des deux structures de contrôle proposées est clairement indiquée sur les figures ci-dessous.

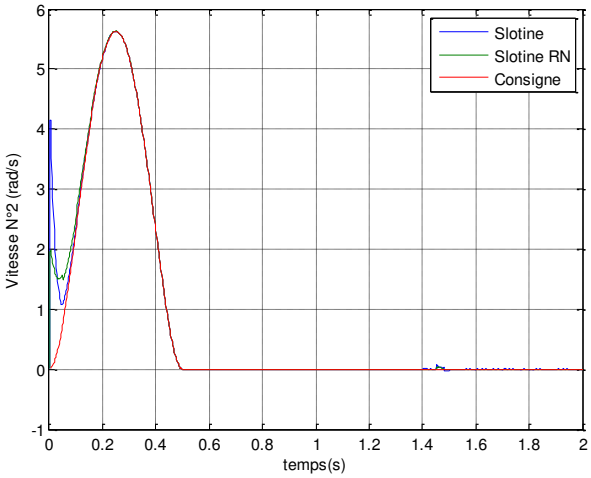


Figure IV.9. Poursuite en vitesse de l'articulation 2.

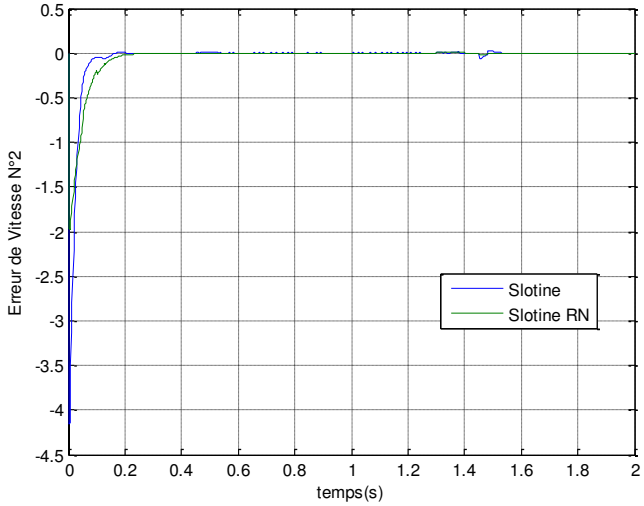


Figure IV.10. Erreur poursuite en vitesse de l'articulation 2.

Aussi et d'après les figures IV.7, IV.8, IV.9 et IV.10, nous pouvons remarquer l'amélioration nette sur la réponse pour le cas de la commande Slotine neuronale.

IV.7.5. Commande neuro-floue robuste STFSM

Concernant la commande neuro-floue robuste 'STFSM', nous avons utilisé deux réseaux STFIS séparés. Pour chacun, nous avons utilisé cinq fonctions d'appartenance de type sigmoïde et gaussiennes figure (IV.11). La loi de commande est donnée comme suit :

$$\Gamma = u_{stfis} + u_s \qquad \text{(IV.55)}$$

Avec

$$u_s = -K\,sat(s) \tag{IV.56}$$

$$s = \dot{\tilde{q}} + \Lambda\tilde{q} \tag{IV.57}$$

Les paramètres de la loi de commande sont les suivants :

$\beta = 0.00001$, η =0.9, et $b = 0.3.$, λ =diag{25, 20} et K =diag {250, 200}.

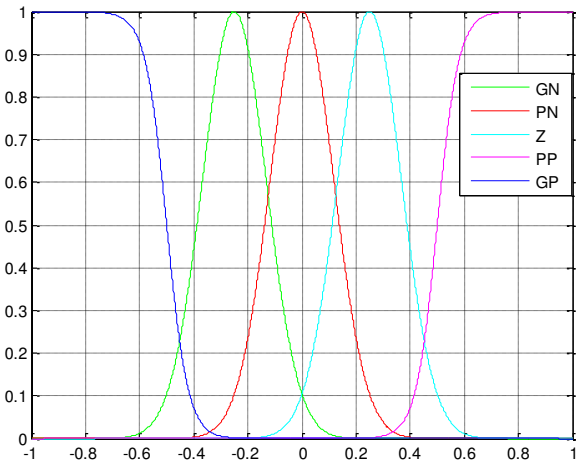


Figure IV.11. Fonctions d'appartenances assignée aux variables d'entrée s et $\dot{s}$.

IV.7.6. Résultats de simulations

La poursuite et les erreurs de poursuite de position et de vitesse pour les deux articulations sont présentées. Un bruit a été introduit dans la boucle de commande.

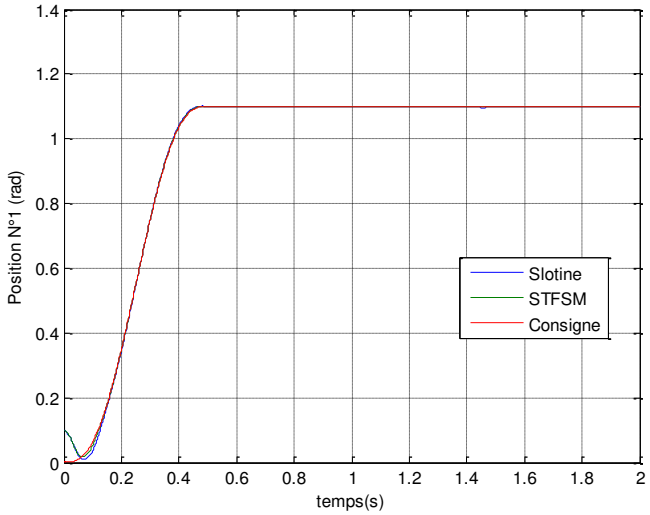


Figure IV.12. Poursuite de position de l'articulation 1.

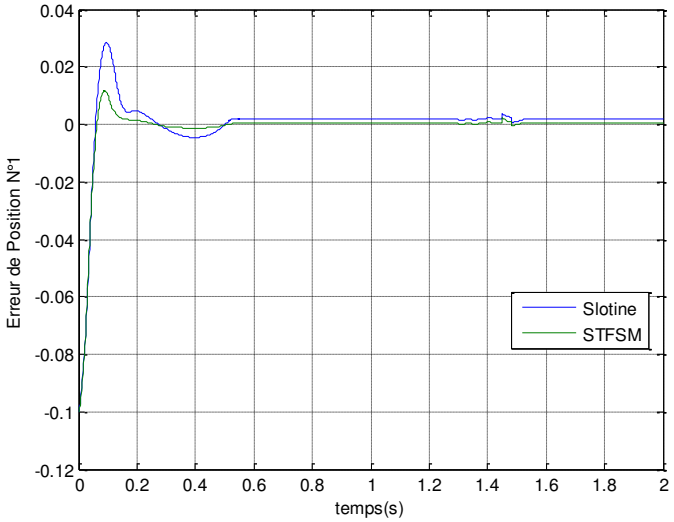


Figure IV.13. Erreur de poursuite de l'articulation 1.

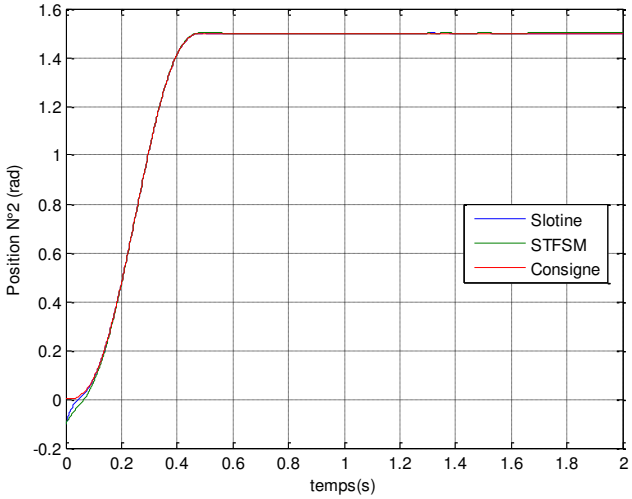


Figure IV.14. Poursuite de position de l'articulation 2.

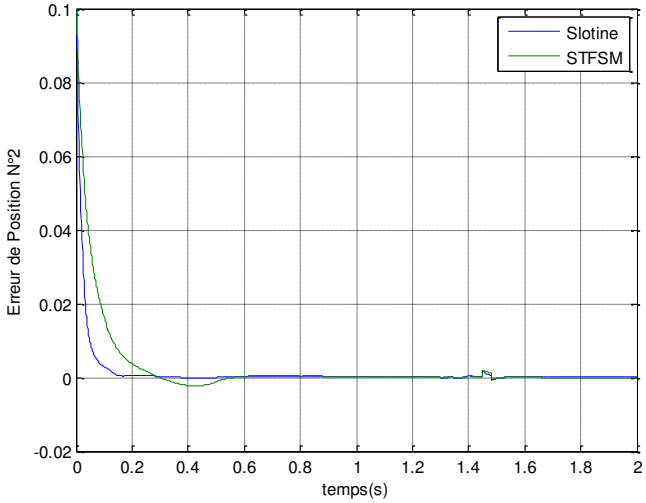


Figure IV.15. Erreur de poursuite de l'articulation 2.

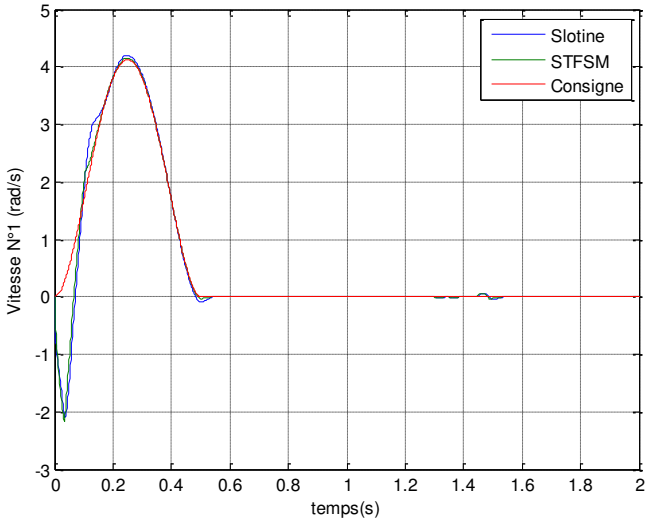


Figure IV.16. Poursuite en vitesse de l'articulation 1.

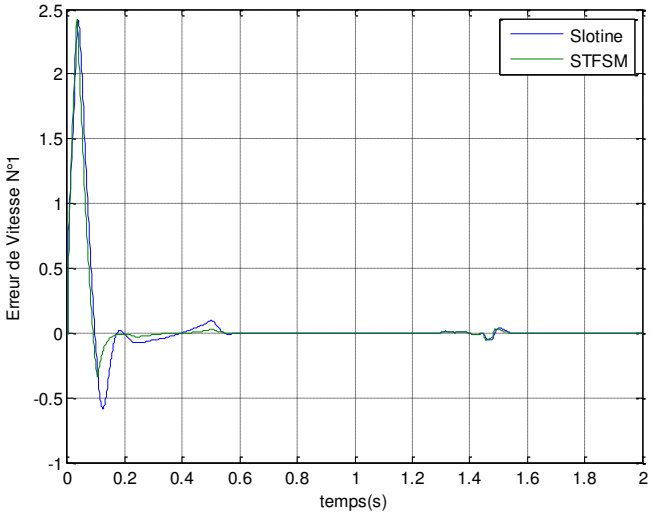


Figure IV.17. Erreur poursuite en vitesse de l'articulation 1.

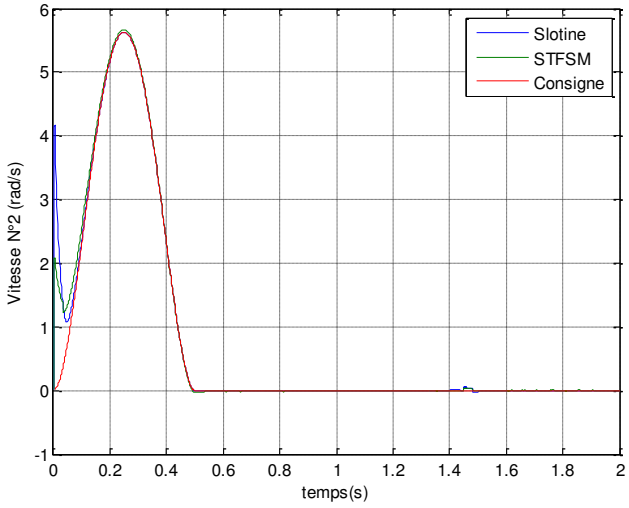


Figure IV.18. Poursuite en vitesse de l'articulation 2.

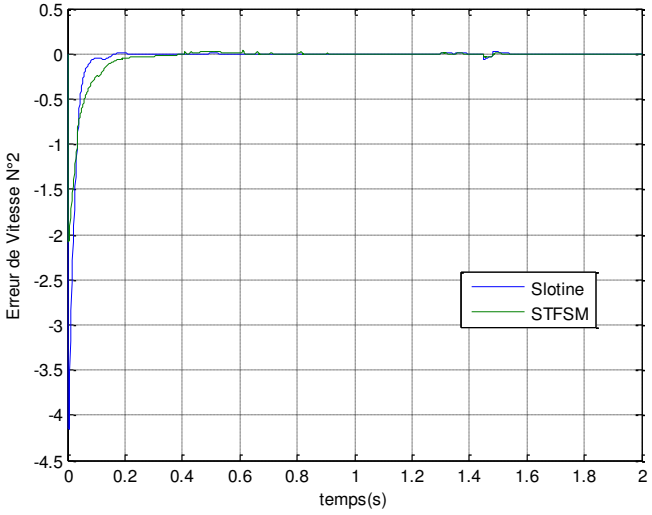


Figure IV.19. Erreur poursuite en vitesse de l'articulation 2.

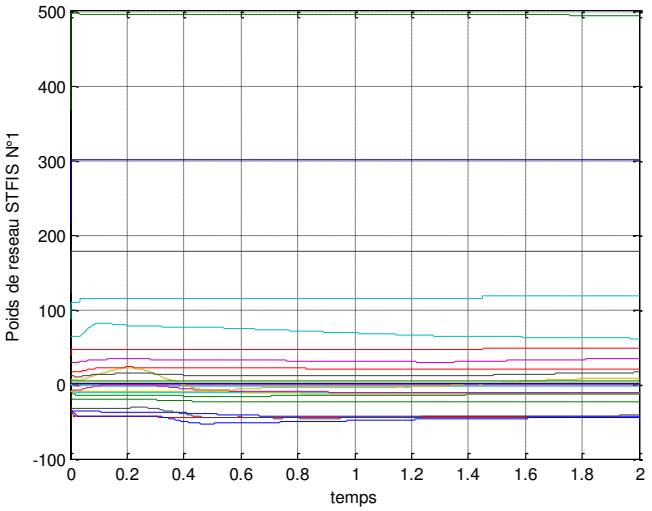


Figure IV.20. Convergence des poids de réseaux STFIS N°1.

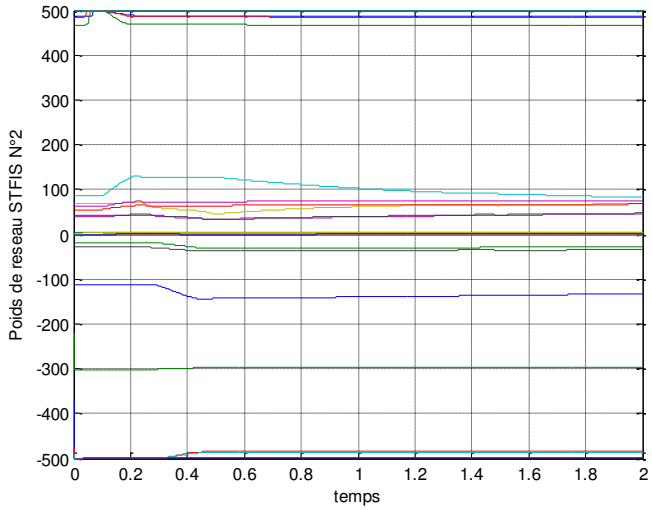


Figure IV.21. Convergence des poids de réseaux STFIS N°2.

Les résultats de la simulation sont présentés sur les figures (IV.12 jusqu'à IV.19), lesquelles montrent que la convergence de l'erreur de poursuite est garantie. Les erreurs de poursuite convergent vers zéro dans le cas de la commande STFSM, comme le montrent les figures IV.13 et IV.15.

Les figures IV.20 et IV.21 montrent la convergence des poids des deux réseaux STFIS. Nous remarquons que la robustesse des deux structures de contrôle est clairement indiquée sur les figures ci-dessus.

IV.8. APPLICATION AU ROBOT SCARA

Pour montrer l'efficacité et la performance de la commande STFSM proposée, nous avons utilisé un deuxième robot; c'est le modèle du robot SCARA à trois articulations [Wai, 2002]. Les équations dynamiques dérivées par la méthode d'Euler-Lagrange sont présentées comme suit:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + f(\dot{q}) + d(t) = \Gamma \tag{IV.58}$$

La matrice d'inertie $M(q)$ est :

$$M(q) = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{13} & M_{23} & M_{33} \end{bmatrix} \tag{IV.59}$$

Avec:

$$M_{11} = l_1^2\left(\frac{m_1}{3} + m_2 + m_3\right) + l_1 l_2\,(m_2 + 2m_3)\cos(q_2) + l_2^2\left(\frac{m_2}{3} + m_3\right)$$

$$M_{12} = M_{21} = -l_1 l_2\left(\frac{m_2}{2} + m_3\right)\cos(q_2) - l_2^2\left(\frac{m_2}{3} + m_3\right)$$

$$M_{22} = l_2^2\left(\frac{m_2}{3} + m_3\right)\ ,\ M_{33} = m_3\ ,\ M_{13} = M_{23} = M_{31} = M_{32} = 0$$

La matrice des forces centrifuges et de Coriolis $C(q,\dot{q})$ est :

$$C(q,\dot{q}) = l_1 l_2 \sin(q_2)\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{13} & C_{23} & C_{33} \end{bmatrix} \tag{IV.60}$$

Avec

$$C_{11} = -\dot{q}_2(m_2 + 2m_3)\ ,\ C_{12} = -\dot{q}_2\left(\frac{m_2}{2} + m_3\right)$$

$$C_{21} = -\dot{q}_1(\frac{m_2}{2} + m_3) \quad C_{13} = C_{22} = C_{23} = C_{31} = C_{32} = C_{33} = 0$$

Le vecteur de gravité $G(q)$ est :

$$G(q) = \begin{bmatrix} 0 \\ 0 \\ -m_3 g \end{bmatrix} \tag{IV.61}$$

Dans lequel q_1, q_2 et q_3 sont les angles des articulations de 1, 2 et 3. m_1, m_2 et m_3 sont les masses. l_1, l_2 et l_3 sont la longueur des liaisons 1, 2 et 3; g est l'accélération de la pesanteur.

Des paramètres importants influant sur le rendement de contrôle du système robotique, sont la perturbation externe $d(t)$ et le terme de frottement $f(\dot{q})$. Les paramètres du robot SCARA sont donnés comme suit:

$l_1 = 1.0m$, $l_2 = 0.8m$, $l_3 = 0.6m$, $m_1 = 1.0\text{kg}$, $m_2 = 0.8\text{kg}$, $m_3 = 0.5\text{kg}$, $g = 9.8$.

Les perturbations externes d et les forces de frottement f sont choisies comme suit:

$$d(t) = \begin{bmatrix} 3\sin(3t) \\ 3\sin(3t) \\ 3\sin(3t) \end{bmatrix} \tag{IV.62}$$

$$f(\dot{q}) = \begin{bmatrix} 12\dot{q}_1 + 0.2\,sign(\dot{q}_1) \\ 12\dot{q}_2 + 0.2\,sign(\dot{q}_2) \\ 12\dot{q}_3 + 0.2\,sign(\dot{q}_3) \end{bmatrix} \tag{IV.63}$$

La trajectoire utilisée dans la simulation pour les trois articulations est une interpolation polynomiale de degré cinq.

avec :

$$q^i = \begin{pmatrix} 0 & 0 \end{pmatrix}^T \tag{IV.64}$$

$$q^f = \begin{pmatrix} 1 & 1.5 & 2 \end{pmatrix}^T \tag{IV.65}$$

Les paramètres de simulation sont :

$\beta = 0.00001$, $\eta = 0.9$, $b = 0.3$, $\Lambda = \text{diag}\{25, 20; 15\}$ et $K = \text{diag}\ \{450, 400; 350\}$.

Le vecteur de l'état initial est :

$$x(0) = \begin{bmatrix} -0.1 & 0.01 & 0.2 \end{bmatrix}^T$$

IV.8.1. Résultats de simulations

La poursuite de position et l'erreur de poursuite de l'articulation 1 sont représentées respectivement dans les figures IV.22, IV.23. La poursuite de position et l'erreur de poursuite de l'articulation 2 sont représentées respectivement dans la figure IV.24 et IV.25. Les figures IV.26 et IV.27 représentent la poursuite de position et l'erreur de poursuite de l'articulation 3. On voit sur les figures IV.22, IV.24 et IV.26 que les positions suivent bien la trajectoire souhaitée. Les erreurs de poursuite convergent vers zéro, comme montrent les figures IV.23, IV.25 et IV.27.

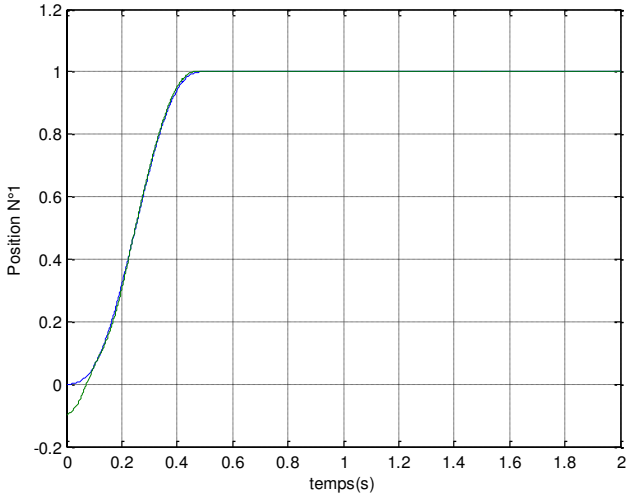


Figure IV.22. Poursuite de position de l'articulation 1.

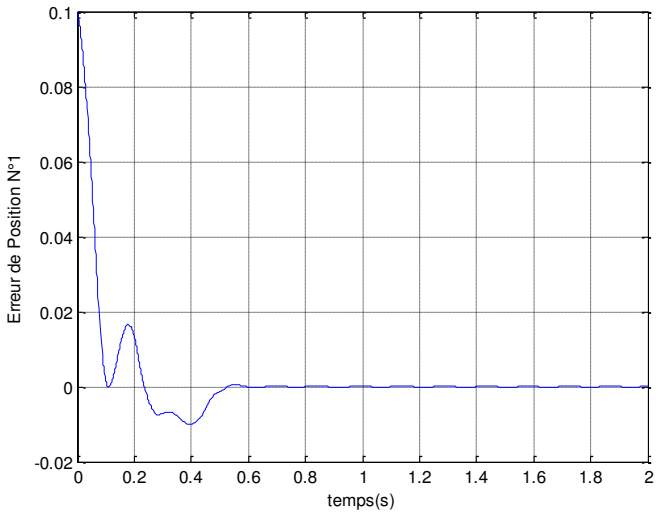


Figure IV.23. Erreur de poursuite de l'articulation 1.

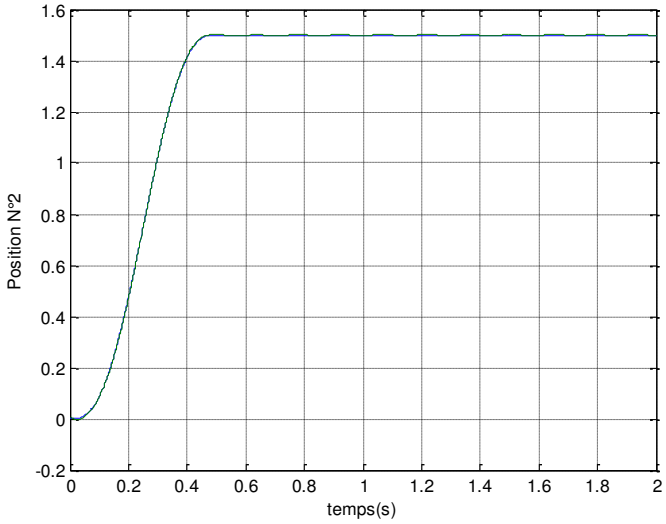


Figure IV.24. Poursuite de position de l'articulation 2.

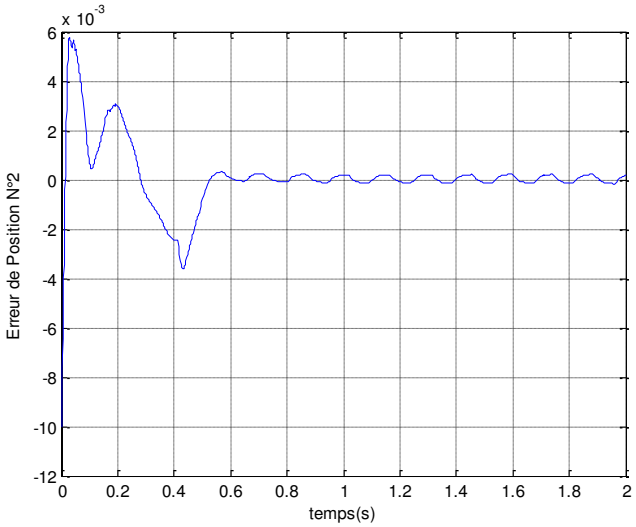


Figure IV.25. Erreur de poursuite de l'articulation 2.

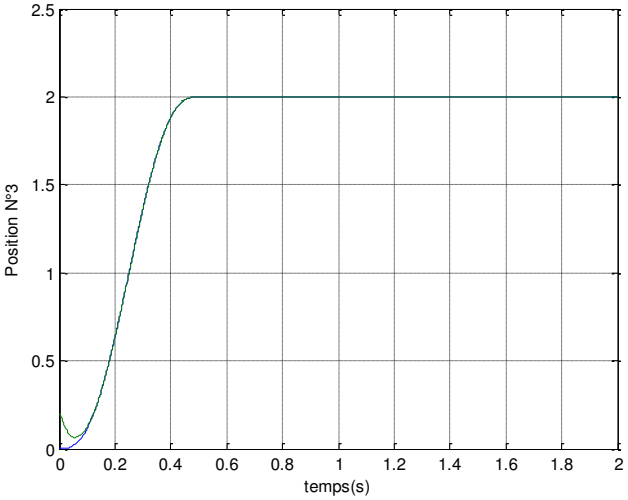


Figure IV.26. Poursuite de position de l'articulation 3.

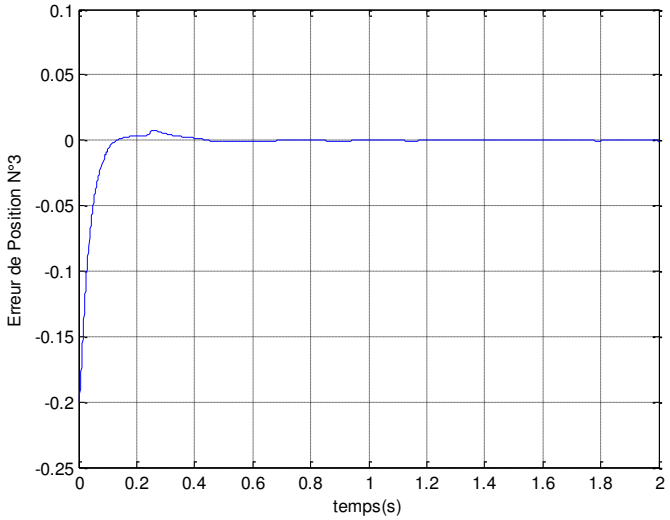


Figure IV.27. Erreur de poursuite de l'articulation 3.

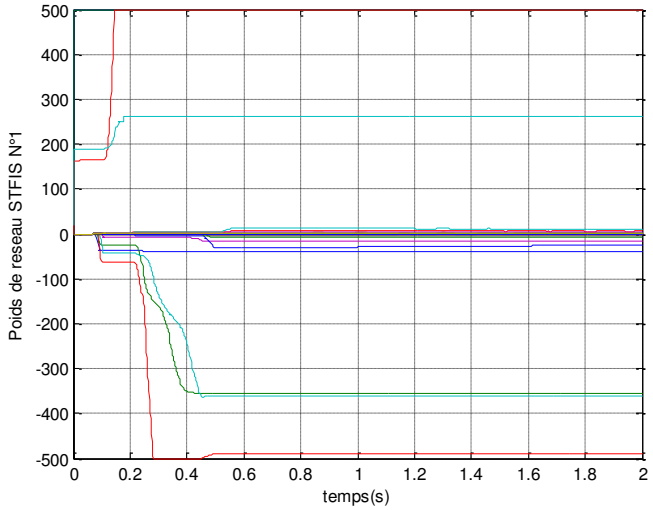


Figure IV.28. Convergence des poids de réseaux STFIS N°1.

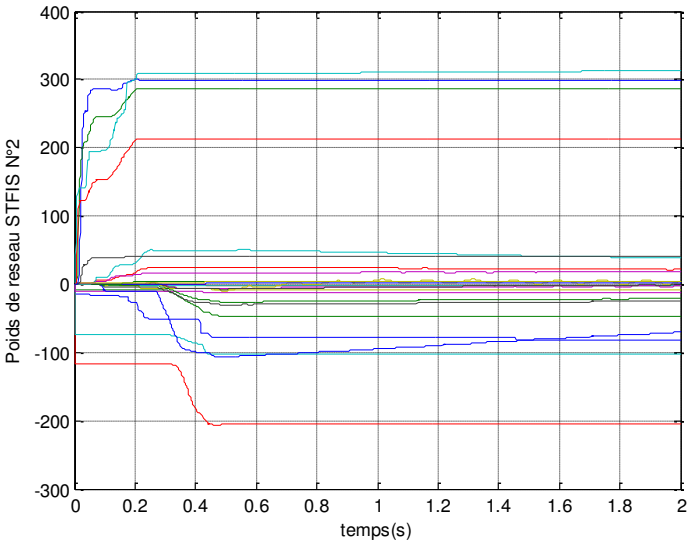


Figure IV.29. Convergence des poids de réseaux STFIS N°2.

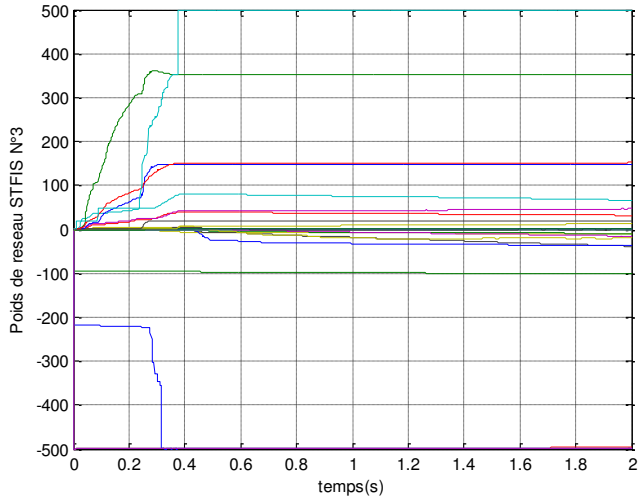


Figure IV.30. Convergence des poids de réseaux STFIS N°3.

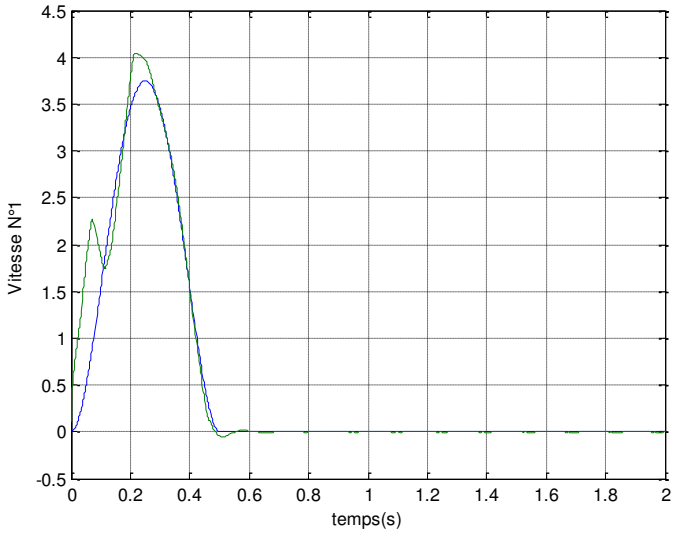


Figure IV.31. Poursuite en vitesse de l'articulation 1.

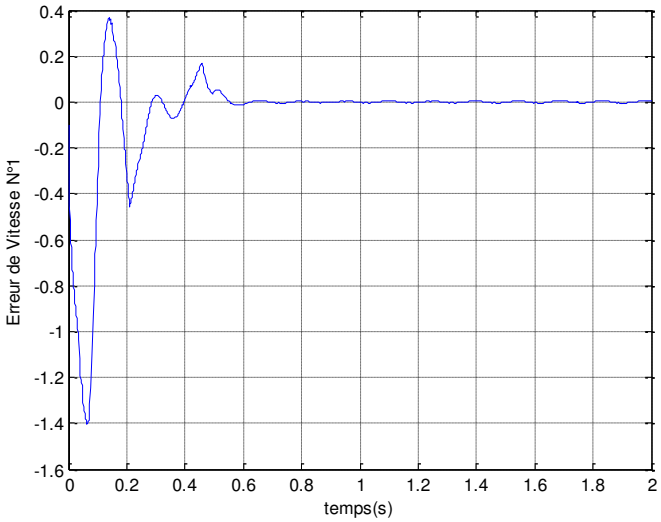


Figure IV.32. Erreur poursuite en vitesse de l'articulation 1.

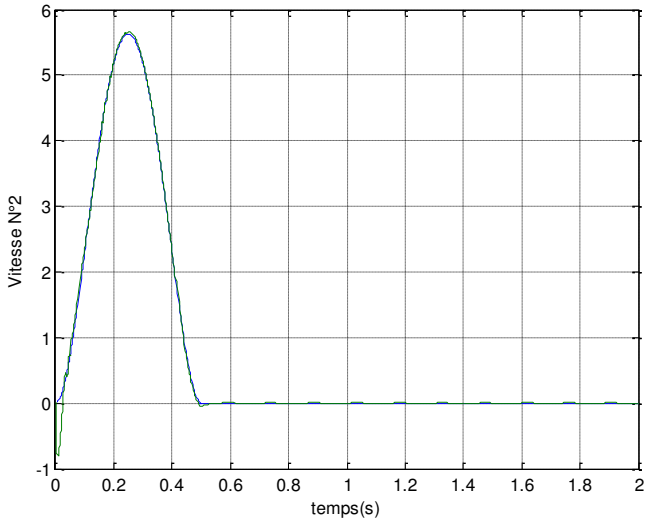


Figure IV.33. Poursuite en vitesse de l'articulation 2.

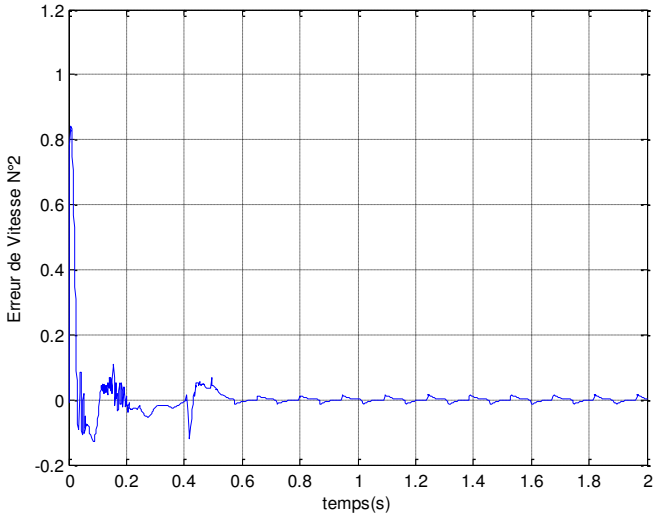


Figure IV.34. Erreur poursuite en vitesse de l'articulation 2.

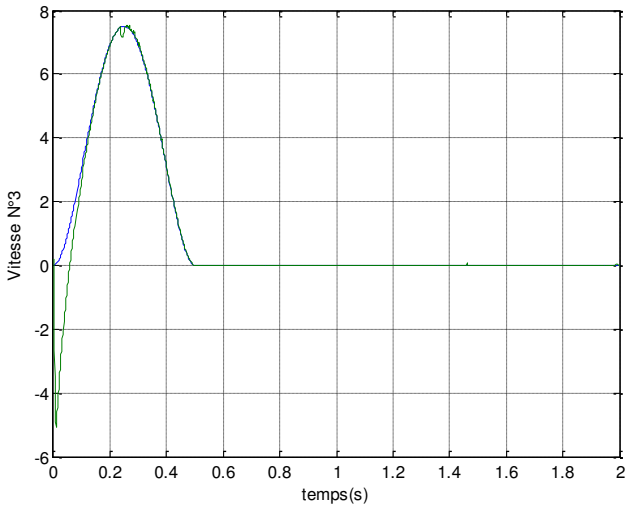


Figure IV.35. Poursuite en vitesse de l'articulation 3.

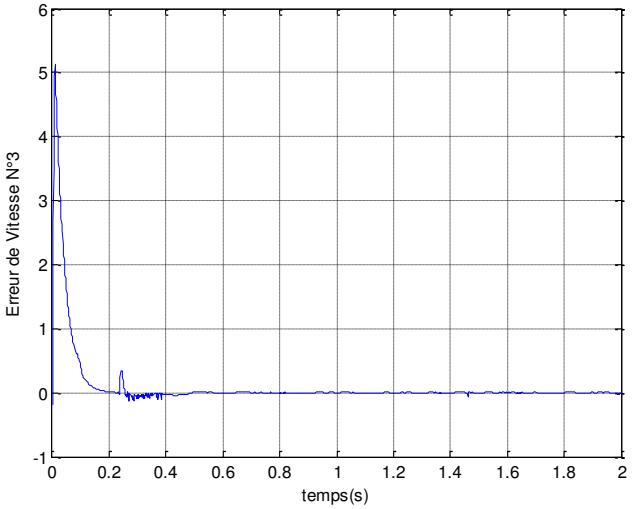


Figure IV.36. Erreur poursuite en vitesse de l'articulation 3.

Les figures IV.28, IV.29 et IV.30 montrent la convergence des poids de la dernière couche de chaque réseau STFIS obtenue par optimisation en ligne. La poursuite de vitesse et l'erreur de vitesse pour l'articulation 1 sont représentées respectivement dans les figures IV.31 et IV.32. La poursuite de vitesse et l'erreur de vitesse pour l'articulation 1 sont représentées respectivement dans les figures IV.33 et IV.34. Les figures IV.35 et IV.36 représentent la poursuite de vitesse et erreur de vitesse pour l'articulation 3.

On remarque sur les figures IV.31, IV.33 et IV.35 que les vitesses sont très proches des trajectoires désirées. Les erreurs de vitesse deviennent évidemment plus petites et convergent vers zéro, comme indiqué sur les figures IV.32, IV.34 et IV.36.

IV.9. CONCLUSION

Dans ce chapitre, nous avons abordé le problème du suivi de trajectoire d'une classe de systèmes non linéaires, et ceci en tenant compte des performances de poursuite en présence de perturbations externes. La stratégie de commande proposée est principalement destinée aux systèmes non linéaires et multi variables. Elle consiste à décomposer le système MIMO en un ensemble de sous-systèmes MISO, ce qui permettra de réguler chaque sous-système indépendamment des autres. Cette décomposition a l'avantage d'assurer l'identification des paramètres sans aucune hypothèse sur la richesse de l'excitation, de réaliser facilement le découplage, et de garantir une bonne poursuite.

La stratégie de commande adoptée consiste à combiner les réseaux de neurones en premier lieu, le réseau neuro-flou 'STFIS' en deuxième lieu, à la technique de mode glissant ce qui permet de résoudre les problèmes des perturbations externes et les incertitudes paramétriques sans aucune hypothèse sévère sur la nature des non-linéarités.

D'excellents résultats, en terme de poursuite, ont été obtenus dans les deux cas de commande, ce qui est d'ailleurs justifié par les résultats des simulations présentés et effectués sur deux différents robots, le premier est un robot manipulateur à deux degrés de liberté et le deuxième est un robot SCARA à trois articulations.

CONCLUSION GENERALE

Le travail présenté dans cette thèse a pour objectif d'apporter une contribution aux travaux déjà menés dans le cadre de l'association de l'intelligence artificielle à la rigueur du mode glissant; il s'agit de développer des lois de commande intelligente par modes glissants pour résoudre les problèmes de poursuite des systèmes non linéaires incertains et perturbés.

Après avoir donné un bref aperçu sur le principe des réseaux de neurones et la procédure générale à suivre pour identifier un système, nous avons d'abord présenté les divers algorithmes d'apprentissage et les structures de commande des systèmes non linéaires par réseaux de neurones basées sur le modèle neuronal inverse. L'intérêt de la détermination de l'architecture optimale a été mise en évidence; ensuite nous avons utilisé le modèle retenu après apprentissage dans une structure de commande directe en boucle ouverte mais ceci suppose que le modèle inverse est presque parfait et que le système est non bruité, ce qui est loin de la réalité. Pour essayer de remédier à ce problème, nous avons utilisé le modèle inverse dans une structure hybride avec un PID classique, où nous avons abouti à des résultats très encourageants.

Après avoir présenté un bref aperçu sur la théorie des ensembles flous ainsi que les outils mathématiques nécessaires à leurs manipulations, dans le chapitre 2, nous avons opté d'utiliser des systèmes neuro-flous, tels que le réseau neuro-flou ANFIS et le réseau neuro-flou STFIS pour la commande. Le réseau STFIS utilise une méthode d'optimisation par descente de gradient pour effectuer l'optimisation des paramètres de système d'inférence flou caractérisant les conclusions des règles floues. L'optimisation est effectuée entièrement en ligne. Nous avons appliqué ces techniques pour la commande d'un système masse-ressort. Les résultats sont validés par des simulations sous MATLAB; ensuite nous avons pu vérifier par comparaison entre les commandes (commande par réseaux de neurones, commande ANFIS et commande STFIS) que le contrôleur neuro-flou STFIS présente de bons résultats malgré la différence de stratégies d'apprentissage.

Pour améliorer la robustesse du système bouclé, nous avons présenté dans le chapitre 3, une commande intelligente par mode glissant (STFSM) en utilisant un réseau neuro-flou. L'approche STFIS avec ses capacités de modélisation des connaissances utilisant une technique d'optimisation permettant l'ajustement en ligne de ses paramètres flous, et la commande par mode de glissement avec sa robustesse, ont été combinées pour la commande des systèmes non-linéaires. Nous avons appliqué cette technique pour la commande d'un pendule inversé, puis nous avons pu vérifier, par ailleurs, que le contrôleur neuro-flou robuste STFSM ainsi obtenu, avait une très bonne robustesse vis à vis des perturbations sur les

signaux d'entrée. Ensuite, nous avons traité le problème de poursuite pour le modèle complet du pendule inversé, soumis aux perturbations externes. Nous avons proposé une commande neuro-floue mode glissant découplé, en utilisant l'approche de la commande par mode glissant découplé. La loi de commande synthétisée permet de donner un meilleur résultat qu'avec la commande par mode glissant découplé.

Le chapitre 4 a été consacré au problème du suivi de trajectoire d'une classe de systèmes non linéaires et multi-variables. Ceci consiste à décomposer le système MIMO en un ensemble de sous-systèmes MISO, ce qui nous a permis de réguler chaque sous-système indépendamment des autres. Cette décomposition a l'avantage d'assurer l'identification des paramètres sans aucune hypothèse sur la richesse de l'excitation, de réaliser facilement le découplage, et de garantir une bonne poursuite. La stratégie de commande adoptée consiste à combiner les réseaux de neurones en premier lieu, le réseau neuro-flou 'STFIS', en deuxième lieu, à la commande mode glissante de Slotine, ce qui permet de résoudre les problèmes des perturbations externes et les incertitudes paramétriques. D'excellents résultats, en terme de poursuite, ont été obtenus dans les deux cas de commande, ce qui est d'ailleurs justifié par les résultats de simulations présentés et effectués sur deux différents robots, le premier est un robot manipulateur à deux degrés de liberté et le deuxième est un robot SCARA à trois articulations.

Afin d'éprouver un peu plus nos commandes, nous envisageons d'appliquer ces approches sur un banc d'essai d'un pendule inversé réel ou un robot réel. Au niveau de la loi de commande, nous envisageons la combinaison de la commande neuro-floue STFIS avec d'autres commandes robustes comme la commande H∞, et la commande back stepping. L'introduction d'un observateur d'état dans le cas d'un système d'ordre n est envisagée.

ANNEXES

A. STABILITE AU SENS DE LYAPUNOV

A.1. Théorème : Lyapunov

Etant donné le système non-linéaire : $\dot{x} = f(t,x)$, avec $x(0) = x_0$ un point d'équilibre à l'origine [i.e., $f(t,0)=0$], et soit N un voisinage de l'origine ; c'est-à-dire,

$$N=\left[x;\|x\| \text{ est petit}\right]$$

Alors nous avons, [Lewis, 1993] :

Stabilité: l'origine est stable au sens de Lyapunov si pour $x\in N$ il existe une fonction scalaire $V(t,x)$avec un dérivé partiel continu tels que

(1) $V(t,x)$ est définie positive

(2) $\dot{V}(t,x)$ est semi-définie négative

Stabilité Uniforme: l'origine est uniformément stable si en plus de (1) et de (2)

(3) $V(t,x)\le\beta(\|x\|)$ pour tous $t\ge 0$ et tout $x\in N$ et $N=R^n$

Stabilité Asymptotique: L'origine est asymptotiquement stable si $V(t,x)$ satisfait (1) et

(4) $\dot{V}(t,x)$ est définie négative

Stabilité asymptotique globale: l'origine est globalement, asymptotiquement stable si $V(t,x)$ vérifie (1) et (4) pour tous $x\in R^n$ (i.e., si $N=R^n$).

Stabilité Asymptotique Uniforme: l'origine est *UAS* si $V(t,x)$ satisfait (1), (3) et (4).

Stabilité Asymptotique Uniforme Globale: l'origine est *GUAS* si $N=R^n$ et si $V(t,x)$ satisfait (1), (3), (4) et (5) $V(t,x)$ va uniformément à l'infini dans le temps quand $\|x\|\to\infty$.

Stabilité Exponentielle: L'origine est exponentiellement stable s'il existe des constantes positives α, β et γ tels que.

(6) $\alpha\|x\|^2\le V(t,x)\le\beta\|x\|^2$ et $\dot{V}(t,x)\le-\gamma\|x\|^2$ pour tous $x\in N$

Stabilité Exponentielle Globale : L'origine est globalement exponentiellement stable si (6) est vrai pour tout le $x\in R^n$.

A.2. Lemme de Barbalat

Soit une fonction $f(t)$ dérivable.*Première version* [Lewis, 1993]:

Si $\dot{f}(t)=\frac{df}{dt}$ est uniformément continue et $\lim_{t\to+\infty} f(t)<\infty$, alors $\lim_{t\to+\infty}\dot{f}(t)=0$.

Deuxième version : Si $f(t)\ge 0$, $\dot{f}(t)$ est bornée, alors $\lim_{t\to+\infty}\dot{f}(t)=0$.

A.3. Théorème deLaSalle

Considérons un système non linéaire décrit par $\dot{x}=f(t)$. Supposons qu'il existe une fonction V de Lyapunov définie positive vérifiant $\lim_{\|x\|\to+\infty}\|V(x)\|\to\infty$ et $\dot{V}(x)\leq 0$ pour tout $x\in R^n$. Définissons un voisinage $D=\{x\in R^n/\dot{V}(x)=0\}$ et supposons que la seule trajectoire contenue dans D soit la trajectoire triviale, alors le point d'équilibre x=0 est globalement asymptotiquement stable, [Lewis, 1993].

B. Génération de mouvement entre deux points

Nous considérons un robot à n degrés de liberté. Soit q^i et q^f les vecteurs des coordonnées articulaires correspondants aux configurations initiales et finale. Le mouvement interpolé entre q^i et q^f en fonction du temps t, est décrit par l'équation suivante [Dombre, 1988]:

$$q_d(t)=q^i+r(t)D \quad 0\leq t\leq t_f \tag{A.1}$$

Avec :

$$D=q^f-q^i \tag{A.2}$$

Les valeurs aux limites de la fonction d'interpolation $r(t)$ sont données par :

$$\begin{cases} r(0)=0 \\ r(t_f)=1 \end{cases} \tag{A.3}$$

Plusieurs fonctions permettent de satisfaire le passage par q^i à t=0et par q^f à $t=t_f$. Nous étudierons successivement l'interpolation polynomiale et la loi Bang-Bang.

B.1. Interpolation polynomiale de degré trois

Si nous imposons une vitesse nulle au point initial et au point final, nous ajouterons deux contraintes aux deux contraintes précitées dans l'équation de position de la méthode interpolation linéaire. Le polynôme qui satisfait ces quatre contraintes est de degré trois et il a la forme générale suivante :

$$q_d(t)=q^i+(3(\frac{t}{t_f})^2-2(\frac{t}{t_f})^3)D \tag{A.4}$$

Avec les conditions initiales et finales :

$$q_d(0)=q^i,\ q_d(t_f)=q^f,\ \ \dot{q}_d(0)=0,\ \dot{q}_d(t_f)=0. \tag{A.5}$$

La figure A.1 montre l'évolution de la position, la vitesse et l'accélération avec le temps en utilisant une interpolation polynomiale de degré trois. Cette loi de mouvement assure la continuité de vitesse mais pas celle d'accélération.

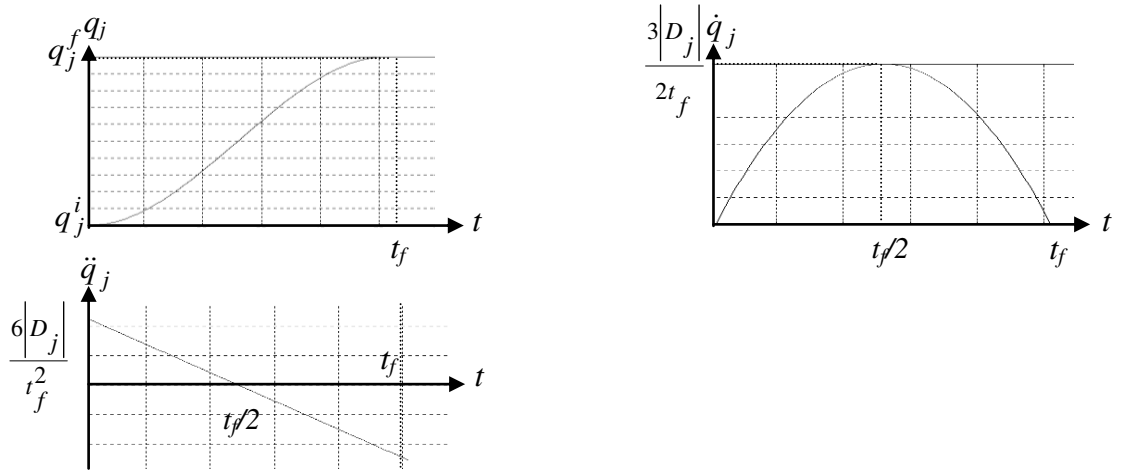


Figure A.1. Interpolation polynomiale de degré trois

B.2. Interpolation polynomiale de degré cinq

Si nous cherchons la continuité de l'accélération, nous somme obligé de satisfaire six contraintes et l'interpolation polynomiale sera de degré cinq. Choisissons en plus à (A.5), les conditions :

$$\ddot{q}_d(0)=0,\ddot{q}_d(t_f)=0 \tag{A.6}$$

L'équation de mouvement s'écrit comme suit :

$$q_d(t)=q^i+(10(\frac{t}{t_f})^3-15(\frac{t}{t_f})^4+6(\frac{t}{t_f})^5)D \tag{A.7}$$

L'évolution de la position, la vitesse et l'accélération du bras *i* avec une interpolation polynomiale de degré cinq est représentée dans la figure A.2.

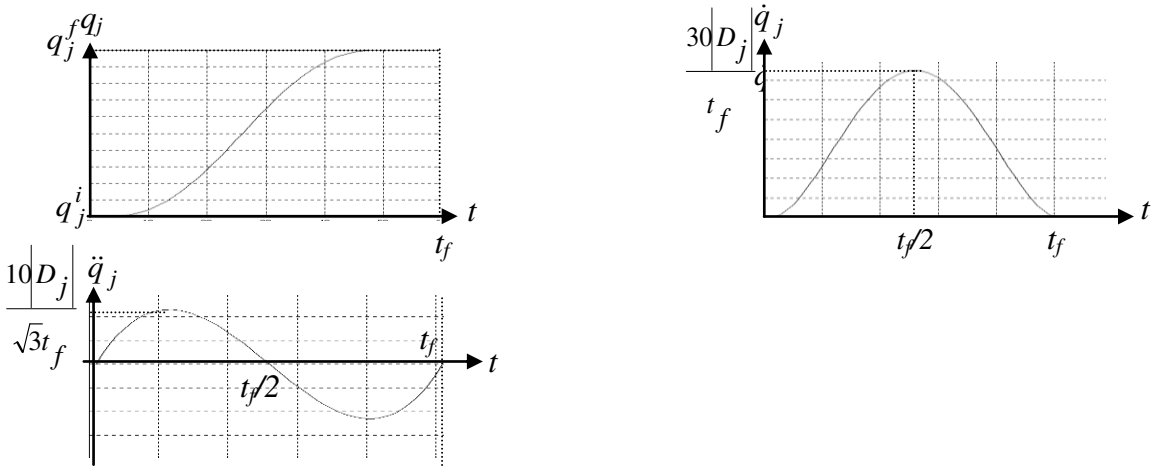


Figure A.2. Interpolation polynomiale de degré cinq

C. IDENTIFICATION DES PROCESSUS

C.1. Principe de l'identification

Quand nous manipulons un système, nous avons besoin de connaître la façon dont ses variables sont reliés entre eux. Avec une large définition, nous appellerons une telle relation: le modèle du système, (voir Figure A.3).

Dans la plupart des cas, Le but de l'identification est de permettre la mise en œuvre d'une bonne régulation du procédé étudié. Afin de mener à bien cette identification, un minimum de connaissances sur le système est indispensable. C'est l'étape qualitative: elle permet de déterminer la structure du modèle. Le plus souvent il s'agit de connaissance sur l'ordre du système, ses conditions initiales, l'amplitude maximum de ses entrées et sorties. Nous pouvons aussi s'aider des équations physiques qui régissent le système pour acquérir ces connaissances

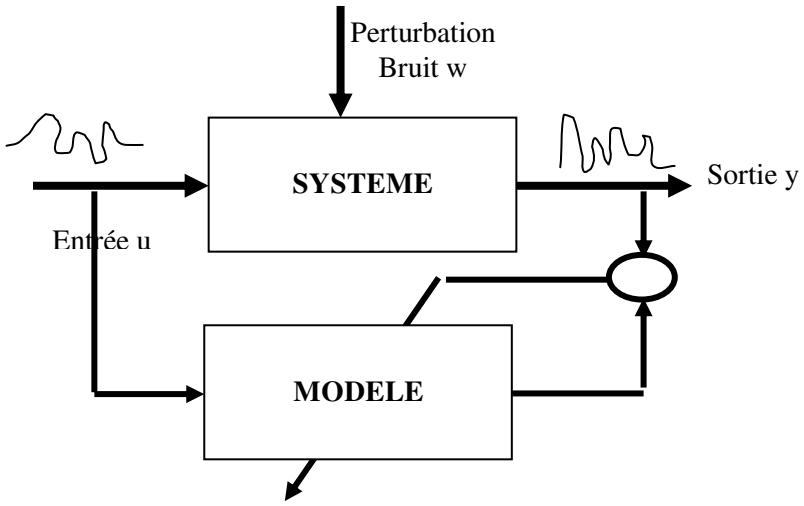


Figure A.3. Principe de l'identification d'un système

C.2. Démarche d'identification

L'approche itérative adoptée est en concordance avec les approches de l'identification des systèmes linéaires que nous trouvons dans [Ljung, 1987].

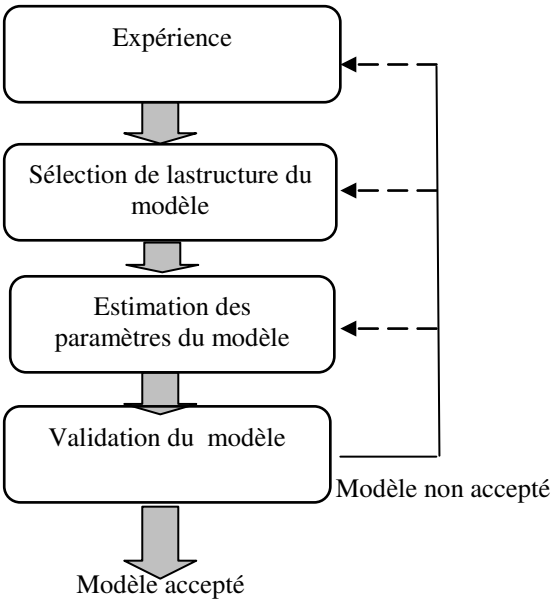


Figure A.4. Démarche d'identification

C.3. Expérience et collecte de données

Consiste à rassembler les connaissances dont nous disposons sur le comportement dynamique du processus, en particulier à collecter un jeu de données représentatif au fonctionnement du système.

L'idée est de faire varier l'entrée du processus et d'observer l'impact sur sa sortie. Le jeu de données recueilli peut être écrit de la façon suivante:

$$Z^N = \{[u(k), y(k)], k = 1,..., N\} \quad \text{(A.8)}$$

Si le processus est instable, il est nécessaire de faire l'expérience en boucle fermée en en introduisant une boucle de commande stabilisante.

La question qui peut se poser est comment choisir les séquences d'apprentissage. La détermination des contraintes sur l'entrée u (la commande) peut être parmi les réponses. Ces contraintes peuvent porter sur l'amplitude et le type de signaux de commandes que le processus est susceptible de recevoir pendant son fonctionnement. Les amplitudes maximales sont en général, faciles à déterminer, car leur ordre de grandeur correspond aux valeurs de saturations des actionneurs, qui peuvent être estimées physiquement (puissance maximale que peut recevoir une machine, tension et intensité maximales d'alimentation, etc.). En ce qui concerne le type de signaux à utiliser, un principe général est que les signaux utilisés pour l'identification doivent être de même nature que ceux qui seront calculés par l'organe de commande pendant l'utilisation du processus. Une démarche, couramment utilisée, consiste à

explorer le mieux possible le domaine de fonctionnement, par exemple avec des créneaux de commande (riche en fréquence), d'amplitudes et de durées diverses [Norgaard, 1996].

C.4. Sélection de la structure du modèle

Consiste à déterminer un ensemble de modèles candidats. Le problème de la sélection d'une structure d'un modèle est double parce qu'il faut:

- Sélectionner une famille de structures de modèles appropriée pour la description du système étudié: structures de modèles linéaires, de réseaux de neurones multicouches ou de modèles d'Hammerstein, Weiner, etc.

- Sélectionner un sous-ensemble de la famille choisie. Dans la famille des structures linéaires, cela peut être, par exemple, une structure ARX.

Le problème d'identification d'un processus peut être entreposé comme suit: nous disposons des entrées observées u(k) et des sorties correspondantes y(k) d'un système dynamique:

$$u^k = \left[u(1), u(2), ..., u(k)\right] \tag{A.8}$$

$$y^k = \left[y(1), y(2), ..., y(k)\right] \tag{A.9}$$

Nous cherchons une relation entre les observations passées $\left[u^{k-1}, y^{k-1}\right]$ et la sortie y(k):

$$y(k) = g(u^{k-1}, y^{k-1}) + e(k) \tag{A.10}$$

Le terme additif e(k) exprime que la sortie y(k) ne sera pas une fonction exacte des observations passées. Cependant, le but doit être que e(k) soit le plus petit possible pour que $g(u^{k-1}, y^{k-1})$ soit une bonne prédiction de y(k).

L'équation (A.10) modélise d'une façon générale les systèmes dynamiques discrets. Les systèmes statiques ne sont que des cas particuliers des systèmes dynamiques. Le plus souvent,la fonction g est recherchée parmi une famille de fonctions de la forme:

$$g(u^{k-1}, y^{k-1}) = g(\varphi(k), \theta) \tag{A.11}$$

Où θ est un vecteur de paramètres de dimension finie et $\varphi(k)$ un vecteur de régression permettant de sélectionner dans u^{k-1} et y^{k-1} les observations passées utiles à la description du modèle.

Le choix de la forme non-linéaire (A.11) est décomposé en deux sous-problèmes: comment choisir le vecteur de régression $\varphi(k)$ et comment choisir la forme de la fonction non-linéaire g. L'analogie avec les cas linéaires fournit toute une série de structures associées au choix des régresseurs. Les structures linéaires ''boîte noire'' utilisées en pratique peuvent être décrites en utilisant le modèle général [Ljung, 1987]:

$$A(q^{-1})y(k) = \frac{B(q^{-1})}{F(q^{-1})}u(k) + \frac{C(q^{-1})}{D(q^{-1})}e(k) \tag{A.12}$$

Où q^{-1} est l'opérateur retard et:

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + a_2 q^{-2} + ... + a_n q^{-n} \\ B(q^{-1}) &= b_0 + b_1 q^{-1} + b_2 q^{-2} + ... + b_m q^{-m} \\ C(q^{-1}) &= 1 + c_1 q^{-1} + c_2 q^{-2} + ... + c_i q^{-i} \\ D(q^{-1}) &= 1 + d_1 q^{-1} + d_2 q^{-2} + ... + d_p q^{-p} \\ F(q^{-1}) &= 1 + f_1 q^{-1} + f_2 q^{-2} + ... + f_r q^{-r} \end{aligned} \tag{A.13}$$

Les composantes du vecteur de régression $\varphi(k)$ sont les entrées retardées u(k-t), associées au polynôme B, les sorties retardées y(k-t), associées au polynôme A, les sorties simulées uniquement à partir des entrées retardées $\tilde{y}(k-t \mid \theta)$, associées au polynôme F, les erreurs de prédiction, $\varepsilon(k-t) = y(k-t) - \tilde{y}(k-t)$, associées au polynôme C et les erreurs de simulation $\varepsilon_n(k-t) = y(k-t) - \tilde{y}_n(k-t \mid \theta)$, associées au polynôme D.

C.5. Estimation des paramètres du modèle

Elle consiste à déterminer les valeurs numériques des coefficients de la structure choisie à partir d'algorithmes d'identification et des données provenant du processus à étudier. L'estimation des paramètres du modèle est effectuée en minimisant une fonction coût, définie à partir de l'écart entre les sorties mesurées du processus, et les valeurs prédites par le modèle. La qualité de cette estimation dépend du modèle choisi, de la richesse des séquences d'apprentissage et de l'efficacité de l'algorithme utilisé.

C.6. La validation du modèle

Le modèle doit être évalué pour tester s'il répond bien aux exigences demandées. Le modèle obtenu n'est pas valable en toute rigueur que pour l'expérience réalisée ; il faut donc qu'il soit compatible avec l'utilisation que nous en ferons.Si le résultat des tests de validation n'est pas satisfaisant, il est nécessaire de reprendre tout ou une partie de la démarche d'identification. Plusieurs retours en arrière peuvent être envisagés.

- Retour à l'estimation des paramètres du modèle si la technique utilisée n'est pas adaptée.

- Retour à la sélection de structures de modèles si la classe de modèles n'est pas apportée.

-Retour à l'expérience si le jeu de données n'est pas suffisamment représentatif du système à étudier. Dans ce cas, il est nécessaire de refaire l'expérience afin d'acquérir plus d'information à propos des régimes manquants.

BIBLIOGRAPHIE

[Ajith, 2001] A. Ajith,'Neuro Fuzzy Systems: State-of-the-art Modeling Techniques', School of Computing & Information Technology, Université Monash, Australie, 2001.

[Antsaklis, 1990] P.J. Antsaklis, ''Neural networks in control systems'', IEEE Control Systems Magazine, vol.12 n°10, pp: 3-5, 1990.

[Aussem, 1995] A. Aussem, ''Théorie et applications des réseaux de Neurones Récurrents et Dynamiques à la Prédiction, à la Modélisation et au Contrôle Adaptatif des Processus dynamiques'', thèse de doctorat1995.

[Babuška, 2003] B. Babuška, H. Verbruggen, "Neuro-fuzzy methods for nonlinear system identification," A Journal of Annual Reviews in Control, 2003, vol. 27 pp 73–85.

[Barron, 1993] A. R. Barron, Universal Approximation Bounds for Superpositions of a Sigmoidal Function, IEEE Transactions on Information Theory, vol. 39, no. 3, pp. 930-945, 1993.

[Bartlett,1994] E. B. Bartlett, 'Dynamic node architecture learning: an information theoretic approach', Neural Networks, vol.7, pp: 129-140, 1994.

[Bartolini, 1998] G. Bartolini, A Ferrara et E. Usai, 'Chattering Avoidance by Second Order Sliding Mode control', IEEE transactions on Automatic Control, vol. 43, no. 2, pp. 241-246, 1998.

[Belhachat, 2007] F. Belhachat, C. Larbes, L. Barazane et S. Kharzi, 'Commande Neuro-Floue d'un Hacheur MPPT', 4th International Conference on Computer Integrated Manufacturing CIP, Novembre 2007.

[Berghuis, 1993] H. Berghuis, et H. Nijmeijer, 'Global regulation of robots using only position measurements', Systems & Control Letters, 1993, vol. 21, no 4, p. 289-293.

[Boukezzoula, 2000] R. Boukezzoula, 'Commande floue d'une classe de systèmes non linéaires : application au problème de suivi de trajectoire', Thèse de Doctorat, Université de Savoie, 2000.

[Canudas, 1996] C. Canudas, B. Siciliano et G. Bastin, 'Theory of robot control', Springer-Verlag, Berlin, 1996.

[Chaouch, 2011] D. E. Chaouch, H. MAAREF, 'Intelligent Control of an Inverted Pendulum by Self Tunable Fuzzy PI-type Controller', In Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-2011), pp. 728–733, July 2011 Aix-les-Bains, France © 2011. Published by Atlantis Press.

[Chaouch, 2012.a] D. E. Chaouch, Z. Ahmed-Foitih et M. F. Khelfi, 'A sliding mode based control of 2dof robot manipulator using neural network', In : Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2012 6th International Conference on. IEEE, 2012. p. 906-911.

[Chaouch, 2012.b] D. E. Chaouch, Z. Ahmed-Foitih et M. F. Khelfi, 'A self-tuning fuzzy inference sliding mode control scheme for a class of nonlinear systems', Journal of Vibration and Control, 2012, p. 1077546311419177

[Chekroun, 2009] S. Chekroun, 'Commande Neuro-Floue sans Capteur de Vitesse D'une machine Asynchrone Triphasée', Mémoire de Magister, Ecole Normale Supérieure d'Enseignement Technologique d'Oran,25 Octobre 2009.

[Chen, 1997] C. T. Chen, W.D. Chang et J. Hwu, 'Direct control of nonlinear dynamical systems using an adaptive single neuron', IEEE Trans on Neural Networks, vol.2 n° 10, pp: 33-40, 1997.

[Cheng, 1996] C.S Chen et W.L. Chen, 'Robust model reference adaptive control of nonlinear systems using fuzzy systems', Int. J. Syst. Sci. vol. 27, no. 12, pp. 1435–1442, 1996.

[Cybenko, 1989] G. Cybenko, 'Approximation by superposition of sigmoidale functions', Mathematics of of Control Signal and Systems, vol.2, pp: 303-314, 1989.

[Delyon, 1995] B. Delyon, A. Juditsky et A. Benveniste, 'Accuracy analysis for wavelet approximations', IEEE Trans. Neural Networks, vol. 6, no. 2, pp. 332–348, 1995.

[Dreyfus, 2000] G. Dreyfus, 'Réseaux de neurones: méthodologies et applications', Ed Eyrolles, 2002.

[Dubois, 1980] D. Dubois et H. Prade, 'Fuzzy Sets and Systems: Theory and Applications', Academic Press, 1980.

[Eliasmith, 2002] C. Eliasmith et C. H. Anderson, 'Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems', Ed The MIT Press, 2002.

[Funahashi, 1989] K. Funahashi, 'On the approximate realization of continuous mappings by neural networks', Neural Networks, vol. 2, pp: 183-192, 1989.

[Gomi, 1993] H. Gomi et M. Kawato, 'Neural network control for a closed loop system using feedback error learning', Neural Networks, vol.5, pp: 933-946, 1993.

[Grondin, 1994] B. Grondin, 'Les réseaux de neurones pour la modélisation et la conduite des réacteurs chimiques: simulations et expérimentations', Thèse de doctorat de l'université de Bordeaux I, 1994.

[Guerra, 2001] T.M. Guerra et L. Vermeiren, 'Control laws for Takagi-Sugeno fuzzy models, Fuzzy Setsand Systems', vol. 120, no.1, 2001.

[Guez, 1990] A. Guez et I. Bar-Kana, 'Two degree of freedom robot adaptive controller', In American Conference on Control, San Diego, pp: 3001-3006, 1990.

[Ha, 2001] Q. P . Ha, Q. H. Nguyen, D. C. Rye et H. F. Durrant-Whyte, 'Fuzzy Sliding-Mode Controllers with Applications', IEEE transactions on industrial electronics, vol. 48-1, 2001.

[Hagan, 1995] M. T. Hagan, H. Demuth et M. Beale, 'Neural network design', Ed PWS publishing company. Boston, U.S.A, 1995.

[Harris, 1994] C. J. Harris, 'Advances in intelligent control', Ed Taylor and Francis Inc, 1994.

[Hazzab, 2006] A. Hazzab 'Commande des Systèmes par logique floue, Réseaux Neurones et Algorithmes Génétiques' Thèse de Doctorat Es-Sciences, Option commande électrique, Université des Sciences et de Technologie d'Oran Mohamed Boudiaf, février 2006.

[Hsu, 2004] C.F. Hsu, T.T. Lee, C.M. Lin et L.Y. Chen, 'Robust neuro-fuzzy controller design via sliding-mode approach', IEEE International Conference on Fuzzy Systems, Budapest (Hongrie), vol.2, pp. 917- 922, 2004.

[Hung, 2007] L. C. Hung, et H. Y. Chung, 'Decoupled sliding-mode with fuzzy-neural network controller for nonlinear systems', International journal of approximate reasoning, 2007, vol. 46, no 1, p. 74-97.

[Hunt, 1991] K. Hunt et D. Sbarnaro, 'Neural networks for non-linear internal model control', IEEE Proceedings-D, vol.138 n°5, pp: 431-438, 1991.

[Ichikawa, 1992] Y. Ichikawa et T. Sawa, 'Neural network application for direct feedback controllers. Neural Networks, IEEE Transactions on, 1992, vol. 3, no 2, p. 224-231.

[Jang, 1993] J.S.R. Jang, 'ANFIS: adaptive-network-based fuzzy inference system', Systems, Man and Cybernetics, IEEE Transactions on, 23(3), 665-685, 1993.

[Jorgensen, 1990] C. C. Jorgensen et C. Schley, 'A neural network baseline problem for control of aircraft flare and touchdown', Neural networks for Control, pp: 423-425, 1990.

[Kaiyu, 2000] Kaiyu, Z., Hongye, S., Jian, C., et al. ''Globally stable robust tracking of uncertain systems via fuzzy integral sliding mode control''. In :Intelligent Control and Automation, 2000. Proceedings of the 3rd World Congress on. IEEE, 2000. p. 1827-1831.

[Kelley, 1999] KELLEY C. T, 'Iterative Methods for Optimization', Ed SIAM, 1999.

[Khalil, 1996] Khalil HK. ''Nonlinear systems''. Second ed. Englewood Cliffs, NJ: Prentice Hall; 1996.

[Khalil, 1999] Khalil W. et Dombre E., "Modélisation, identification et commande des robots", Hermès Science Publications, Paris, 1999

[Khelfi, 1996] M.F. Khelfi, M. Zasadzinski, H. rafaralahy, E. Richard, and M. Darouach, "Reduced-order observer-based point-to-point and trajectory controllers for robot manipulators," Control Engineering and Practice, A Journal of IFAC,1996, ,vol. 4, N° 7, pp. 991-1000, Elsevier Science ltd,

[Khelfi, 2013] Khelfi, M.F., Daikh, F.Z., et Chaouch, D. E., ''Sliding mode with neuro-fuzzy network controller for inverted pendulem''. In : Industrial Technology (ICIT), 2013 IEEE International Conference on. IEEE, 2013. p. 193-198.

[Labiod, 2005] S. Labiod, M.S. Boucherit et T.M. Guerra, Adaptive fuzzy control of a class of MIMO nonlinear systems, Fuzzy Sets and Systems, vol. 151, no. 1, pp. 59-77, 2005.

[Larsen, 1980] P. M. Larsen, Industrial applications of fuzzy logic control, Int. J. Man. Mach. Studies, vol. 12, no. 1, pp. 3-10, 1980.

[Leu, 2013] Leu V.Q., Mwasilu F., Choi H.H, Lee J. et. Jung J.W, "Robust fuzzy neural network sliding mode control scheme for IPMSM drives", International Journal of Electronics, 2013.

[Lewis, 1993] Lewis F.L., Abdallah C.T. et Dawson D.M., "Control of robot manipulators", Macmillan, New York, 1993.

[Lin, 2002] W.S. Lin et C.S. Chen, 'Robust adaptive sliding mode control using fuzzy modeling for aclass of uncertain MIMO nonlinear systems', IEE Proc. Control Theory and Applications, vol. 149, no. 3, pp. 193-201, 2002.

[Liu, 2006] J. Liu, et F. Sun, 'Chattering free adaptive fuzzy terminal sliding mode control for second order nonlinear system'. Journal of Control Theory and Applications, 2006, vol. 4, no 4, p. 385-391.

[Ljung, 1987] L. Ljung, 'System identification: theory for the user', Ed Prentice-Hall, Englewood Cliffs, Ni, 1987.

[Maaraf, 2002] H. Maaraf, 'Notion de base de la théorie de flou', Cour de la théorie de flou, Université d'Evry Val d'Essonne, 2002.

[Maaref, 2001] H. Maarafet C. Barret, 'Progressive optimization of a fuzzy inference system', In: IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th. IEEE, 2001. p. 47-52.

[Mackay, 1992] D.J.C. Mackay, 'A practical Bayesian framework for backprop Networks', Neural Computation, vol. 4, pp: 448-472, 1992.

[Marquardt, 1963] D.W. Marquardt, 'An Algorithm for Least-Squares Estimation of Nonlinear Parameters', Proceedings of Society for Industrial and Applied Mathematics (SIAP), Vol.11 I.2, pp: 431-441, 1963.

[Medsker, 2001] L. R. Medsker, L.C. Jain, 'Recurrent neural networks Design and applications', Ed CRC Press, 2001.

[Miller, 1991] W. T. Miller, R. P. Hewes, L. H. Glanz, L.G. Kraft, 'Real time dynamic control of an industrial manipulator using a neural network based learning controller', IEEE Trans on Robotics and Automation, vol.6 n°1, pp: 1-9, 1991.

[Narendra, 1990] K. S. Narendra et K. Parthasarathy, 'Identification and control of dynamical systems using neural networks', IEEE Transactions on Neural Networks, vol.1, pp: 4-27, 1990.

[Narendra, 1997] K. S. Narendra et S. Mukhopadhyay, 'Adaptive Control Using Neural Networks and Approximate Models', IEEE Trans. on Neural Networks, vol. 8, no. 3, 1997.

[Nauck,1997] D. Nauck et R. Kruse, 'What are Neuro-Fuzzy Classifiers?', Seventh International Fuzzy Systems Association World Congress IFSA'97, Vol. IV, pp. 228-233, Académie de Prague ,1997.

[Nerrand, 1993] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, 'Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms', Neural Computation. Vol.5 .pp: 165-199, 1993.

[Nguyen, 2005] NGUYEN, Hung T. et WALKER, Elbert A. A first course in fuzzy logic. CRC press, 2005.

[Norgaard, 1996] NORGAARD M, (1996), "System identification and control with neural networks", Institute of Automation, Technical University of Denmark, thesis.

[Noriega, 1998] J.R. Noriega et H. Wang, A direct adaptive neural network control for unknown nonlinear systems and its application, IEEE trans. Neural Networks, vol. 9, pp. 27-34, 1998.

[Noroozi, 2009] N. Noroozi, M. Roopaei et M.Z. Jahromi, 'Adaptive fuzzy sliding mode control scheme for uncertain systems'. Communications in Nonlinear Science and Numerical Simulation, 2009, vol. 14, no 11, p. 3978-3992.

[Otilia, 2008] E.V Otilia 'Contribution au Pronostic de Défaillances par Réseau Neuro-Flou :Maîtrise de L'erreur de Prédiction', Thèse présentée à L'UFL des Sciences et Techniques de l'Université de Franche-Comté, 28 Octobre 2008.

[Parizeau,2004] M. Parizeau, 'Réseaux de neurones', GIF-21140 et GIF-64326, 2004, vol. 124.

[Perruquetti, 2002] W. Perruquettiet J.P.Barbot, 'Sliding Mode Control in Engineering', Marcel Dekker, 2002.

[Psaltis, 1987] D. Psaltis, A. Sideris et A. Yamamura, 'Neural controllers', In International Neural Networks Conference, vol.4, pp: 551-558, 1987.

[Renders, 1995] J. M. Renders, 'Algorithmes génétiques et réseaux de neurones', Ed Hermes, 1995.

[Ronco, 1997] E. Ronc et P. J. Gawthrop, 'Neural networks for modeling and control', Technical Report CSC-97008, Center for Systems and Control, Glasgow, UK, November 1997.

[Ross, 2009] T.J. Ross, 'Fuzzy logic with engineering applications'. John Wiley& Sons, 2009.

[Saidi, 2007] K. Saidi,'Contrôle des Systèmes non Linéaires par la Logique Floue Optimisée par les Algorithmes Génétiques' Mémoire de Magister, Electronique, Université des Sciences et de Technologie d'Oran Mohamed Boudiaf, 2007.

[Salem, 2007] M. Salem, D.E. Chaouch et M.F. Khelfi, 'Commande neuronale inverse des systèmes non linéaires', International Conference on Computer Integrated Manufacturing CIP'2007.

[Saporta, 1990] G. Saporta, 'Probabilités, analyse des données et statistique', Ed Technip, 1990.

[Sauner, 1992] R. M. Sauner, J.J.E. Slotine, 'Gaussien networks for direct adaptive control', IEEE Trans on Neural Networks, vol.3 n°6, pp: 837-863, 1992.

[Sbarbaro, 1993] H. D. Sbarbaro, D. Neumerkel, K. Hunt, (1993), 'Neural control of a steel rolling mill', Proc of IEEE International Symposium on intelligent control, Glasgaw UK, pp: 122-127, 1993.

[Sciavicco, 2000] L. Sciavicco, et B. Siciliano, 'Modelling and control of robot manipulators', Springer Science & Business Media, 2000.

[Siciliano, 2009] B. Siciliano, L. Sciavicco et L. Villani, 'Robotics: modelling, planning and control', Springer Science & Business Media, 2009.

[Slotine, 1984] J.J.E. Slotine, 'Sliding controller design for nonlinear systems', Int. J. Control, vol. 40, no.2, pp. 421-434, 1984.

[Slotine, 1987] J. J. E. Slotine et W. Li, 'On the adaptive control of robot manipulators', The international journal of robotics research, 1987, vol. 6, no 3, p. 49-59.

[Slotine, 1991] J. J. E. Slotine et W. Li, 'Applied nonlinear control', Prentice Hall, N.J., 1991.

[Spong, 1989] M. W. Spong et M. Vidyasagar, 'Robot dynamics and control', John Wiley& Sons, New York, 1989.

[Spooner, 1996] J. T. Spooner et K. M. Passino, 'Adaptive control of a class of decentralized nonlinear systems', Automatic Control, IEEE Transactions on, 1996, vol. 41, no 2, p. 280-284.

[Steck, 1990] J. Steck, B. Krishnamurthy, B. McMillin et G. Leininger, (1990), 'Neural modeling and control of a distillation column', In International Joint Conference on Neural Networks Seattle, vol. II, pp: 771-773, 1990.

[Sugeno, 1988] M. Sugeno et G.T. Kang, 'Structure identification of fuzzy model', Fuzzy Sets and Syst., vol. 28, no. 1, pp. 15-33, 1988.

[Takagi, 1985] T.Takagi et M. Sugeno, Fuzzy identification of systems and its applications to modelling and control, IEEE Trans. on Man, and Cyber., pp. 116-132, 1985.

[Utkin, 1977] V.I. Utkin, Variable structure systems with sliding modes, IEEE Trans. Automatic Control, vol. 22, pp. 212-222, 1977.

[Wai, 2002] R.J. Wai, et K.Y. Hsieh. ''Tracking control design for robot manipulator via fuzzy neural network''. In : Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on. IEEE, 2002. p. 1422-1427.

[Wang, 1997] L. X. Wang, 'A Course in Fuzzy Systems and Control', Prentice Hall, USA, 1997.

[White, 1992] D. A. White et D. A. Sofge, 'Handbook of intelligent control Neural, fuzzy and adaptive approaches', Ed Van Nostrad, Reinhold, NY, 1992.

[Yoo, 1998] B. Yoo et W. Ham, 'Adaptive fuzzy sliding mode control of nonlinear system', IEEE Transactions on Fuzzy Systems, vol. 6, no. 2, pp. 315-321, 1998.

[Yoshimura, 2015] T. Yoshimura, 'Adaptive fuzzy sliding mode control for uncertain multi-input multi-output discrete-time systems using a set of noisy measurements'. International Journal of Systems Science, 2015, vol. 46, no 2, p. 255-270.

[Zadeh, 1965] L. A. Zadeh, 'Information and Control', *Fuzzy Sets*, vol 8, pp 338-353, 1965

[Zadeh, 1972] L. A. Zadeh, 'A fuzzy-set-theoretic interpretation of linguistic hedges'. 1972.

[Zemalache, 2006] K. Zemalache, 'Commande d'un système sous- actionne : Application a un drone a Quatre Helices', Thèse de doctorat en Génie Informatique, Automatique, Université d'Evry Val d'Essonne, Décembre 2006.

[Zemalache, 2008] K. Zemalache, L. Bejiet, H. Maaref, 'Control of drone: Study and analysis of the robustness', Journal of Automation, Mobile Robotics and Intelligent Systems 2(1): 33–42, 2008.

[Zemalache, 2009] K. Zemalacheet H. Maaref, 'Intelligent control for a drone by self-tunable fuzzy inference system', In Proceedings of the 6th International Multi-conference on Systems, Signals and Devices. pp. 23–26, Djerba, Tunisia, March 23–26, 2009.

[Zhang, 2006] H. Zhang, et D. Liu, 'Fuzzy modeling and fuzzy control', Springer Science & Business Media, 2006.