REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE SUPERIEUREEN SCIENCES APPLIQUEES
--T L E M C E N--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

# École Supérieure En Sciences Appliquées de Tlemcen

Département de la formation du Second Cycle

Filière : Automatique

Polycopié des Travaux Pratiques

# LOGIQUE FLOUE ET RÉSEAUX DE NEURONES

Élaboré par
Mme Fatima Zohra Bekaddour
Mr Ahmed Tahour

# Table des matières

# Liste des figures

# Liste des tableaux

# Liste des abréviations

ANFIS : Adaptative Neuro-Fuzzy Inference System
FIS : Fuzzy Inference System
FLC : Fuzzy Logic Controller
GUI : Graphical User Interface
I.A : Intelligence Artificielle
L.F : Logique Floue
MCC : Machine à Courant Continu
P.I.D : Proportional Integral Derivative
P.M.C : Perceptron Multi-Couches
R.B.F : Radial Basis Network
R.N : Réseaux de Neurones
R.N.A : Réseaux de Neurones Artificiels
S.I.F : Système d'Inférence Flou
T.S.K : Takagi-Sugeno- Kang

*Nous remercions tous les collègues pour les remarques qui ont permis à corriger et améliorer les polycopies.*

# Préface

Ce polycopié est destiné aux étudiants ingénieurs de la quatrième année automatique.
L'objective pédagogique de ces travaux pratiques, sert à familiariser les étudiants aux fondements de l'I.A, modélisation des systèmes en utilisant les approches connexionnistes, optimisation et résolution des problèmes en utilisant la logique floue et les modèles neuro-flous tel que le modèle ANFIS (Adaptative Neuro-Fuzzy Inference System). Ceci permet aux étudiants de bien comprendre certaines notions sur les méthodes d'apprentissage, et d'acquérir une expérience sur la résolution des problèmes en appliquant la technique des réseaux de neurones et/ou logique floue.
En effet l'I.A est certainement intéressante par son but principal : réaliser des systèmes capables de trouver la solution de problème, qui était auparavant seulement résolus par les êtres vivants.
Ce polycopié est constitué de 5 travaux pratique qui se déroule au cours du deuxième semestre:

– TP N◦ 1 : Initiation à la logique floue.
– TP N◦ 2 : La commande par logique floue
– TP N◦ 3 : Initiation aux réseaux de neurones.
– TP N◦ 4 : Régression à base des RNAs.
– TP N◦ 5 : Initiation aux modèles neuro-flous.

Le TP N 1 a été réalisé d'une certaine façon que l'étudiant avant de manipuler les systèmes d'inférence flous, il aura une idée bien détaillé sur la logique floue qui se définit comme un outil puissant pour la modélisation des systèmes pour lesquels on dispose des données approximatives ou imprécises. Donc, la première séance des travaux pratiques est consacrée à la présentation générale du principe de fonctionnement d'un SIF, ainsi que l'utilisation du régulateur flou dans le Simulink.

L'objective du TP N 2 consiste à réaliser une machine à courant continu, en utilisant un PID (Proportional Integral Derivative) et un FLC (Fuzzy Logic Controller). Ceci permet aux étudiants d'étudier d'un côté la simulation d'un moteur à courant continu associé à un régulateur PI et de déterminer un correcteur à base de logique floue équivalent au correcteur continu et à simuler le système de l'autre côté. Notre but dans ce TP est bien de montrer aux étudiants comment sélectionner le régulateur convenable qui donne les meilleures performances du système de point de vu stabilité, précision et rapidité.

Nous présentons dans le 3eme TP, une approche assez célèbre de l'apprentissage artificielle qui est fondé sur le système nerveux biologique. Ce thème est considéré comme faisant parti des approches d'apprentissage supervisé, utilisé dans le domaine de classification ou régression. Le TP N 3 est consacré à la présentation générale du PMC et constitue un premier pas vers la manipulation des RNAs à base de l'outil 'nftool' de matlab, ainsi que la réalisation d'un classifieur neuronale à base des commandes.

Le TP N 4 permet aux étudiants de découvrir un autre modèle des RNAs : RBF (Radial Basis Function). Un exemple classique de l'utilité de cette approche est la régression qui décrit l'approximation d'une fonction dont la variable de sortie prend des valeurs continues. Un exemple d'application assez courant est la modélisation d'un bras robotique dont le but d'adapter le RBF et le PMC pour le control de mouvement d'un simple bras de robot à liaison unique. Un autre exemple d'application traiter dans le TP N 4 est l'analyse et la correction neuronale d'un système de régulation automatique.

La logique floue et les réseaux de neurones forment aujourd'hui la base de la majorité des systèmes intelligents. Il serait intéressant de fusionner les deux pour exploiter la richesse des deux approches. Dans le TP N 5, nous traitons le modèle ANFIS (Adaptative Neuro-Fuzzy Inference System) en expliquant son architecture et son principe de fonctionnement ainsi que la réalisation d'un classifieur neuro-flou à base de la ligne des commandes.

Les travaux pratiques ont été réaliser de tel façons que les étudiants peuvent effectuer leurs manipulations eux-mêmes. A l'aide du support, ils seront dirigés du début jusqu'à l'obtention de tous les résultats. Nous avons pensé aussi que dans chaque séance des travaux pratiques, la majorité du temps est dédié non pas seulement à l'acquisition des résultats mais à l'analyse de ces derniers qui est à notre avis la phase la plus essentiel.

TP N° 1

# Initiation à la Logique floue

## 1.1 Objectifs

- Se familiariser avec la création d'un système flou (SIF) à l'aide des commandes
- Utiliser le Toolbox fuzzy control de Matlab pour la manipulation des SIFs.

## 1.2 Introduction

Bien que les ordinateurs ne traitent que des nombres, les individus utilisent essentiellement des concepts liés entre eux par des règles logiques. Ces concepts qui possèdent un fort contenu sémantique, sont matérialisés par des mots, plus ou moins vagues. La logique floue se propose de formaliser l'usage des termes vagues, dans le but de les rendre manipulables par les ordinateurs.

La théorie des ensembles flous repose sur la notion d'appartenance partielle : chaque élément appartient partiellement ou graduellement aux ensembles flous qui ont été définis.

### 1.2.1 Variables linguistiques et sous ensemble flous :

La description imprécise d'une certaine situation, d'un phénomène ou d'une grandeur physique ne peut se faire que par des expressions relatives ou floues à savoir :

-**Q**uelque **Q**, **B**eaucoup **B**, **S**ouvent **S**,

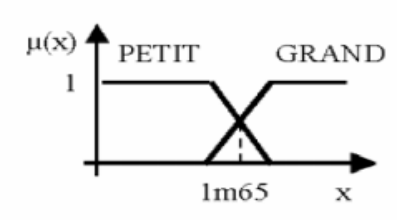
-**C**haud **C**, **F**roid **F**, **R**apide **R**, **L**ent **L**,

-**G**rand **G, P**etit **P**, etc.

Ces différentes classes d'expressions floues dites *ensembles flous* forment ce qu'on appelle des *variables linguistiques*. Afin de pouvoir traiter numériquement ces variables linguistiques (normalisées sur un intervalle appelé *univers de discours*), il faut les soumettre à une

définition mathématique à base de *fonctions d'appartenance* qui montre le degré de vérification (appelé *degré d'appartenance*) de ces variables linguistiques relativement aux différents sous-ensembles flous de la même classe.

**Exemple**




On parle de variable linguistique taille et de valeurs linguistique petit et grand avec $\mu(x) \in [0,1]$

**Figure 1.1** : Variable et Valeurs Linguistiques

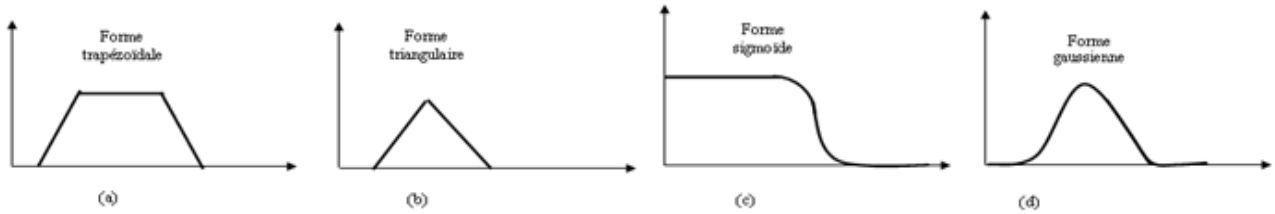



**Figure 1.2** : Formes usuelles des fonctions d'appartenance

## 1.2.2 Principe de fonctionnement d'un système d'inférence floue

Un système d'inférence flou, aussi appelé : modèle floue, système à base de règles floues, ou tout court : le système flou est construit à partir de trois composants conceptuels :

- ✓ Base de règle : Une base des règles floues est composée de règles qui sont généralement déclenchées en parallèle.
- ✓ Base de donnés : définie les fonctions d'appartenance utilisées pour les règles floues.
- ✓ Le mécanisme de raisonnement : représente la procédure d'inférence appelée aussi la généralisation du *modus-ponens* ou raisonnement approximative.

Généralement, un système d'inférence flou comprend trois étapes, décrit dans la **figure 1.3** suivante :

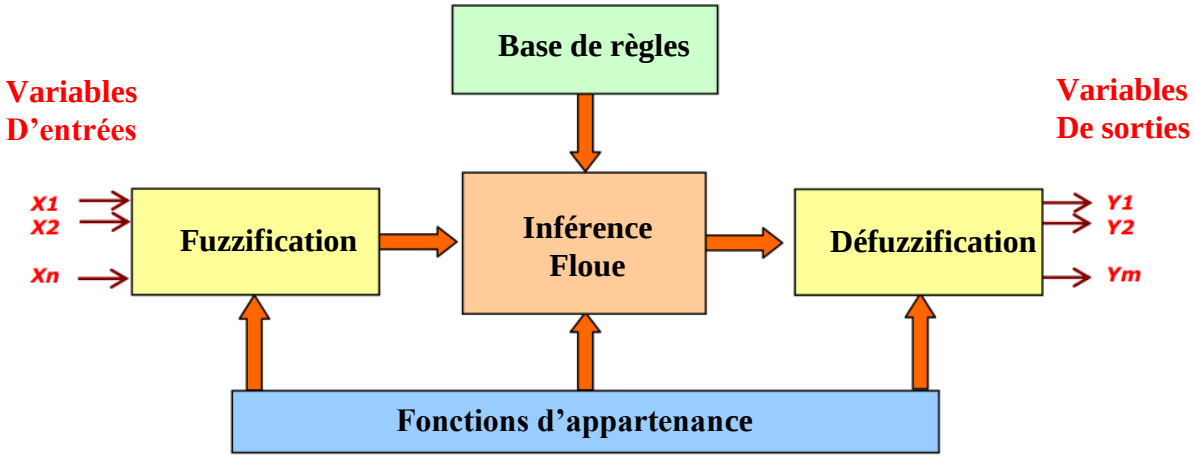



**Figure 1.3** : Système d'inférence flou

La ***fuzzification*** consiste à transformer les entrées numériques en parties floues. Ceci alimente alors le mécanisme **d'*inférence*** qui à partir des valeurs d'entrées et selon la base de connaissance, détermine la valeur correspondante de la sortie. Enfin, la ***défuzzification*** joue le rôle inverse de la fuzzification, en convertissant les parties floues relatives aux sorties du mécanisme d'inférence en sorties numériques.

Il existe principalement deux types de systèmes flous :

- ✓ **Système floue de Mandani** : Dans ce type de systèmes flous, la prémisse et la conclusion sont floues. Après la réalisation de l'inférence floue. Une étape de défuzzification est obligatoire pour le passage du symbolique au numérique.
- ✓ **Système floue de Takagi-Sugeno (TSK)** : Dans ce type de SIF, la conclusion correspond à une constante ou une expression polynomiale. Il permet d'obtenir directement la sortie défuzzifiée à partir des règles linguistiques.

## 1.3. Création d'un système flou à l'aide des commandes de la boîte à outils

### 1.3.1 Création d'un système flou

La création d'un système flou se fait à l'aide de la commande newfis qui accepte jusqu'à 7 arguments.Ses différentes syntaxes sont :

sys_flou = newfis('nom_syst');

sys_flou : matrice caractérisant le système flou,

nom_syst : nom du système flou.

sys_flou = newfis('nom_syst', 'type')

nom_syst : nom du système flou,

type : type Mamdani ou Sugeno.

La syntaxe générale avec les 7 arguments est :

sys_flou = newfis('nom_syst',’type’,’ET_methode’, ‘OU_method’, ’imp_method’, ‘agg_method’,’deffuz_method’);

nom_syst : nom du système flou,

type : type Mamdani ou Sugeno,

ET_methode : méthode utilisée pour l’opérateur ET,

OU_method : méthode utilisée pour l’opérateur OU,

imp_method : méthode d’implication (min ou prod),

agg_method : méthode d’agrégation des règles (max, OU probabiliste ou somme),

deffuz_method : méthode de défuzzification.

## 1.3.2. Fuzzification des variables d’entrée et de sortie

Pour ajouter des variables d’entées la commande suivante est utilisée :

sys_fuz = addvar(’nom’, 'type', ’sys_fuz’,’intervalle’);

sys_fuz : nom du système flou,

type : variable d’entrée 'input' ou de sortie 'output',

nom : nom de la variable auquel feront référence les règles floues,

intervalle : intervalle de valeurs que prend la variable.

Après la définition des différentes variables d’entrée et de sortie par la commande addvar, les différentes fonctions d’appartenance sont spécifiées par la commande addmf.

sys_fuz = addmf(’nom’, 'type', ‘num’, ‘nom_mf’, ’type_mf’,’params’)

nom : nom de la fonction d’appartenance,

type : variable d’entrée 'input' ou de sortie 'output',

num : numéro de la variable (la variable n° 1 est la 1ère créée),

nom_mf : nom de la fonction d’appartenance,

type_mf : type de la fonction d’appartenance, exemple « gaussmf »

params : paramètres de la fonction d’appartenance.

Le tracé des fonctions d’appartenance de la première valeur d’entrée du système sys_flou se fait par la commande plotmf.

**plotmf(sys_flou,'input',1)**

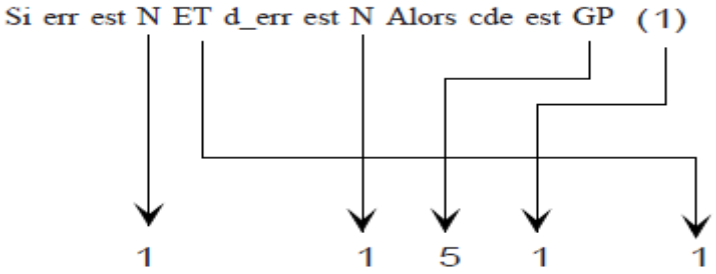
### 1.3.3. Edition des règles floues

Pour un système flou possédant m entrées et n sorties, l'ensemble des règles floues est défini par une matrice de règles possédant autant de lignes que d'ensembles flous de chacune des entrées et (m+n+2) colonnes.

La première règle floue, rappelée ci-après, constitue la première ligne de cette matrice. Si l'erreur est négative (1er ensemble nommé N) ET la dérivée de l'erreur négative (1$^{er}$ ensemble nommé N) alors la commande est grande positive (5ème ensemble nommé GP).

Cette règle est pondérée par 1.

Le dernier chiffre symbolise l'opérateur liant les clauses des 2 prémisses (1 pour ET, 2 pour OU).




```
regles = [ 1     1     5     1          1          (Règle 1)
           1     2     2     1          1];        (Règle 2)
           m     m     n     m+n+1      m+n+2
```

La commande suivante nous permet d'ajouter ses règles à la logique floue.

sys_flou = addrule(nom,regles)

### 1.3.4. Génération de la surface de la variable de sortie en fonction des entrées

La commande surfview(sys_flou) ouvre la fenêtre Surface Viewer dans laquelle on a la possibilité de modifier l'angle de vue à l'aide de la souris en déplaçant un point quelconque de la fenêtre.

La commande gensurf(sys_flou) trace cette surface dans une fenêtre graphique.

Une représentation graphique du système, qui rappelle succinctement les paramètres essentiels (nombre d'entrées et de sorties, nombre et types de fonction d'appartenance, etc.), est obtenue par la commande **plotfis**.

### 1.3.5. Défuzzification

La commande ruleview affiche la fenêtre Rule Viewer dans laquelle on peut observer la défuzzification par la méthode max-min. A l'aide de la souris, on peut choisir des valeurs quelconques pour chacune des entrées et observer la fonction d'appartenance de la variable de sortie obtenue par la méthode max-min.

Par défaut, la défuzzification est réalisée par la méthode du centre de gravité.

### 1.3.6. Sauvegarde sous forme FIS (Fuzzy Inference System)

La commande **writefis** permet de sauvegardée un programme qui est créé dans la structure espace de travail matlab dans un fichier qui a l'extension *.fis

**writefis (sys_flou, nom du système flou)**,

pour appeler se *.fis on utilise la commande

**sys_flou =readfis(nom du système flou)** .

La commande ruleview affiche la fenêtre **Rule Viewer** dans laquelle on peut observer la défuzzification par la méthode choisie.

**ruleview(sys_flou)**

## 1.4. Utilisation du régulateur flou dans SIMULINK

La boite à outils Fuzzy Logic Toolbox est un outil additionnel à Simulink.

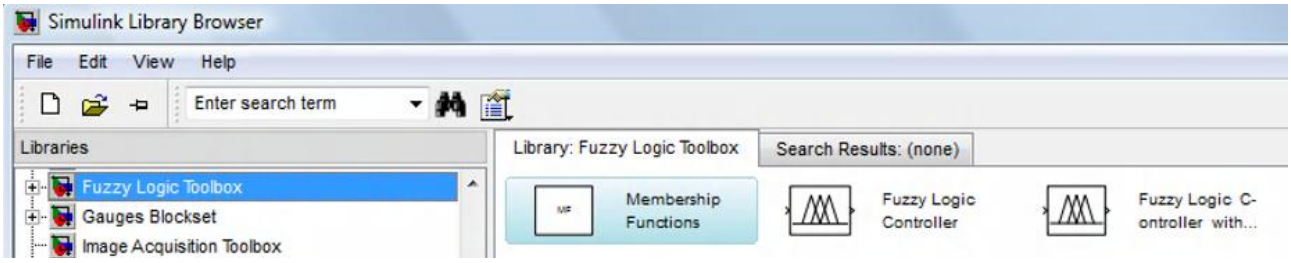



**Figure 1.4** : Bibliothèque de Simulink

La librairie comporte 3 blocs dont le premier comporte différentes formes de fonctions d'appartenance (ensembles flous), les 2 autres étant des régulateurs.

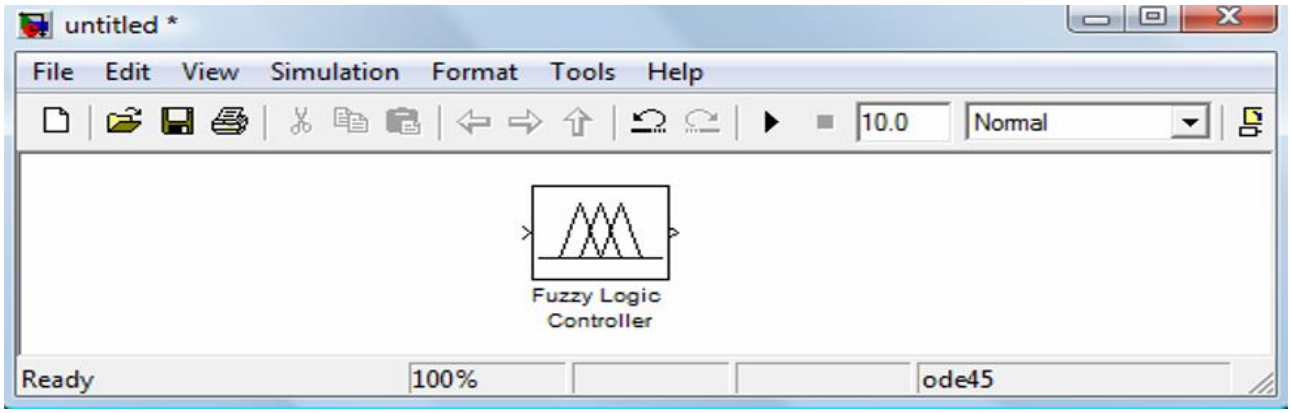



**Figure 1.5** : Le composant Fuzzy Logic Controller (FLC)

A partir de l'interface graphique, nous avons sauvegardé ce système sous le nom regul_flou.fis et dans l'espace de travail sous forme d'une matrice regul_flou.

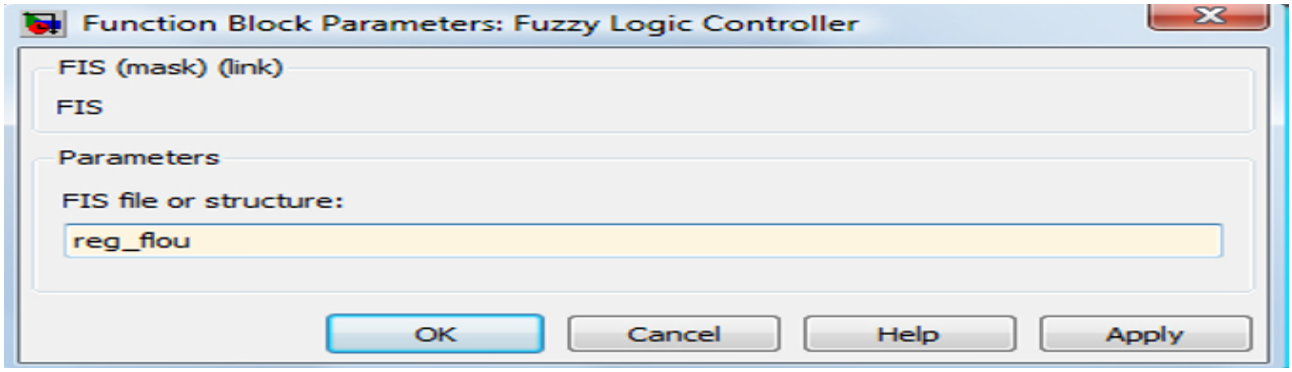



**Figure 1.6** : Paramètres du FLC

Comme le régulateur possède 2 entrées, l'erreur et sa variation, ces dernières doivent être multiplexées.

La commande appliquée au processus est la somme de la sortie du régulateur et celle appliquée à l'instant d'échantillonnage précédent.

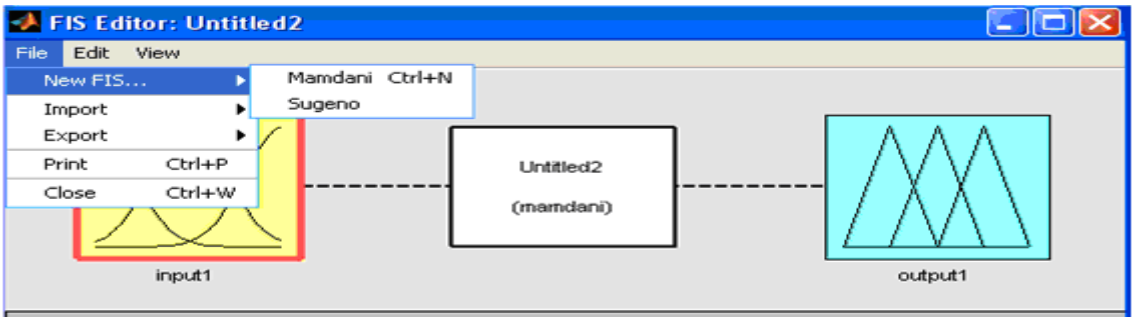



**Figure 1.7** : Fenêtre principale du SIF

On peut choisir une fonction d'appartenance (menu Edit membership function)

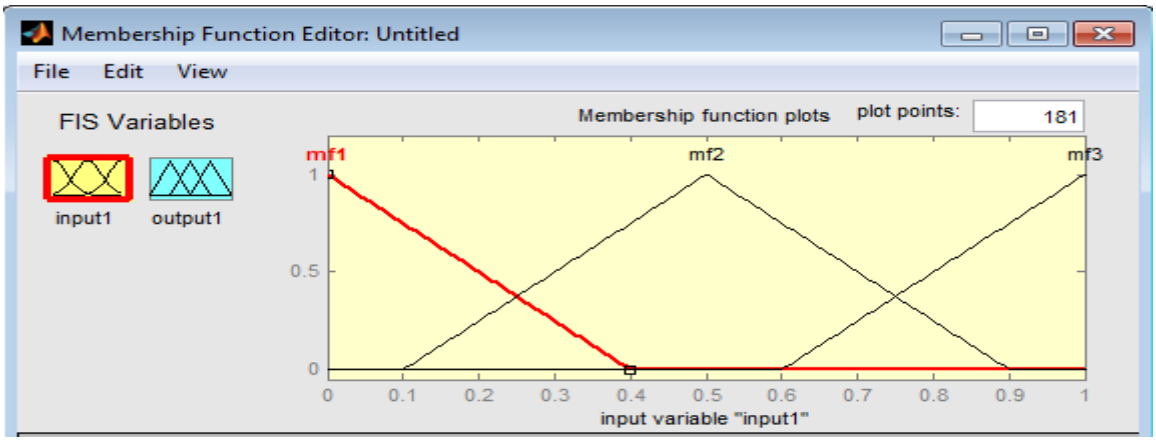



**Figure 1.8** : Edition des fonctions d'appartenances

Editer les règles d'inférences : a une combinaison des entrées correspondra une valeur floue de sortie en utilisant **Add rule.**

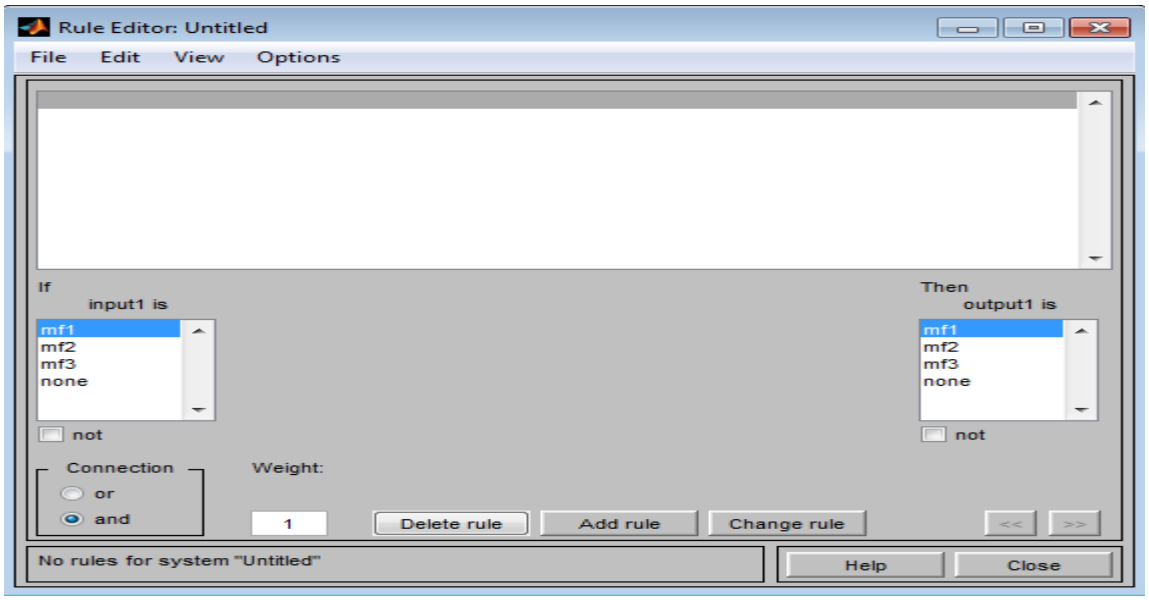



**Figure 1.9** : Edition des règles floues

A partir de l'interface Fuzzy, il faut exporter le fichier vers: Workspace pour pouvoir l'utiliser sous Simulink dans le schéma de système.

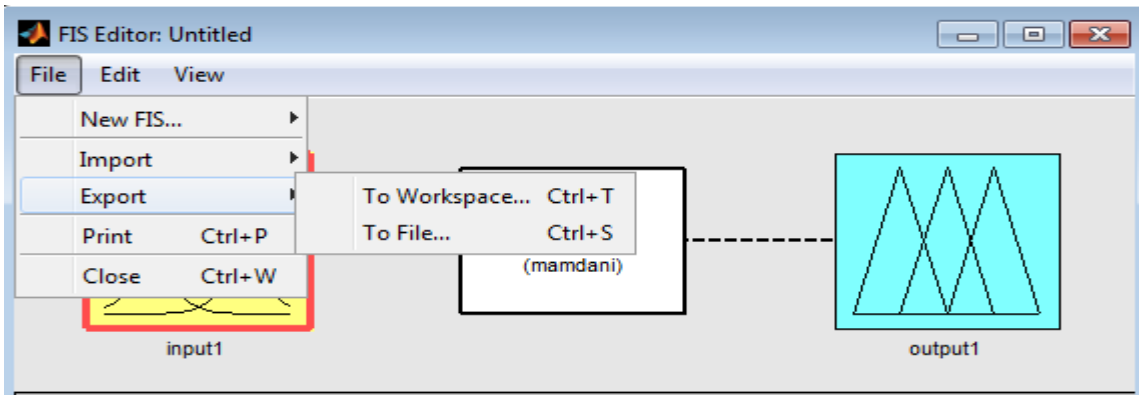



**Figure 1.10** : Sauvegarde du SIF

## 1.5. Travail demandé

On se propose de réaliser à l'aide des commandes de la boîte à outils, un système flou avec deux entrées : l'erreur (**e**) et sa variation (**de**) et une sortie (**cde**).

Par défaut, l'interface propose une entrée et une sortie avec la méthode de mamdani. Les operateurs **ET** et **OU** sont réalisés respectivement par le min et le max, l'implication se fait par le min, l'agrégation des règles par le max et la **défuzzification** par la méthode du centre de gravité (centroid). La fonction d'appartenance choisie en premier lieur est de type gaussienne et en deuxième lieu est de type triangulaire.

**Définition des variables d'entrées : l'erreur et sa variation**

interv_err=[-10 10] ;

interv_derr=[-10 10] ;

**Définition de la variable de sortie du régulateur : la commande**

interv_cde=[-15 15] ;

**Les règles d'inférences**

Négative Grand (NG), Négative(N) , Zero (Z), Positive (P) et Positive Grand (PG).

| **e** / **de** | **N** | **Z** | **P** |
|---|---|---|---|
| **N** | **PG** | **P** | **Z** |
| **Z** | **P** | **Z** | **N** |
| **P** | **Z** | **N** | **NG** |

**Tab 1.1** : Règles floues - Exemple 1-

1- Tracer les ensembles flous de l'erreur (e).
2- Tracer les ensembles flous de la dérivé de l'erreur (de).
3- Tracer les ensembles flous de la variable de sortie (cde).
4- Tracer la surface.
5- Représentation graphique du système flou.
6- Tracer la fenêtre Rule Viewer.

**Même travail pour :**

NB est négative big, NM est négative medium, NS est négative small, ZE est zéro, PB est positive big, PM est positive medium et PS est positive small.

interv_e=[-1 1] ;

interv_de=[-1 1] ;

interv_U=[-1 1] ;

| U | | De | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **NB** | **NM** | **NS** | **ZE** | **PS** | **PM** | **PB** |
| *e* | **NB** | **NB** | **NB** | **NB** | **NB** | **NM** | **NS** | **ZE** |
| | **NM** | **NB** | **NB** | **NB** | **NM** | **NS** | **ZE** | **PS** |
| | **NS** | **NB** | **NB** | **NM** | **NS** | **ZE** | **PS** | **PM** |
| | **ZE** | **NB** | **NM** | **NS** | **ZE** | **PS** | **PM** | **PB** |
| | **PS** | **NM** | **NS** | **ZE** | **PS** | **PM** | **PB** | **PB** |
| | **PM** | **NS** | **ZE** | **PS** | **PM** | **PB** | **PB** | **PB** |
| | **PB** | **ZE** | **PS** | **PM** | **PB** | **PB** | **PB** | **PB** |

**Tab 1.2** : Règles floues - Exemple 2-

TP N° 2

# La commande par logique floue

## 2.1 Objectifs

L'étudiant doit être capable de :

- Visualiser le comportement du système en utilisant les différents types de régulateurs : PI et à base de logique floue.
- Sélectionner le régulateur convenable qui donne les meilleurs performances du système de point de vu stabilité, précision et rapidité.

**N.B :** *P* : action proportionnelle ; *I* : action intégrale

## 2.2 Commande par logique floue

Contrairement aux techniques de réglage classique, le réglage par la logique floue n'utilise pas des formules ou des relations mathématiques bien déterminées ou précises. Mais, il manipule des inférences avec plusieurs règles floues à base des opérateurs floues ET, OU, ALORS,…etc, appliquées à des variables linguistiques.

Un régulateur flou peut être présenté de différentes façons, mais en générale la présentation adoptée se scinde en quatre parties : la fuzzification qui permet de passer de variables réelles à des variables floues, le cœur du régulateur représenté par les règles reliant les entrées et sorties la défuzzification qui permettent à partir des ensembles flous d'entrée de déterminer la valeur réelle de sortie et en fin le bloc de règles qui fournit les différentes paramètres de trois bloc cités précédemment.

- Bases des règles ;
- Fuzzification ;

- un mécanisme d'inférence ;
- Défuzzification.

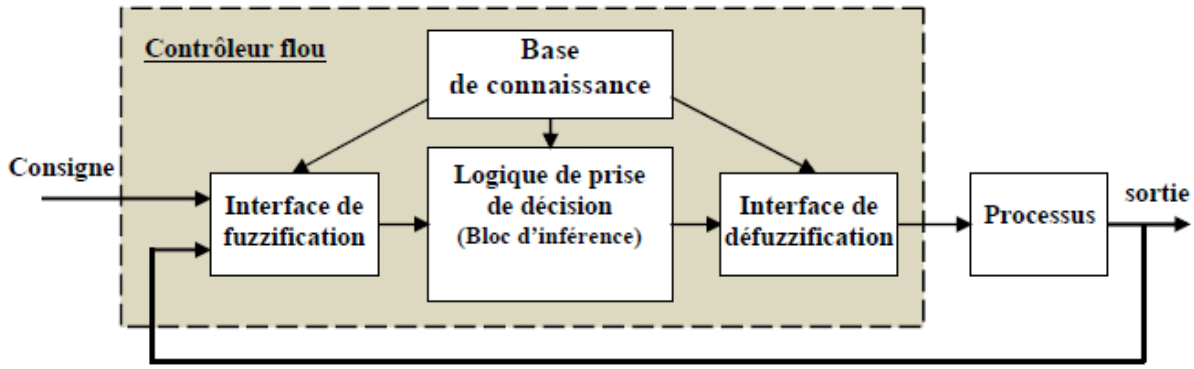



***Figure 2.1.*** Schéma général d'un contrôleur flou.

### 2.3 Le toolbox Fuzzy control

Fuzzy contol toolbox –sous matlab- permet de présenter des informations de haut niveau concernant les SIF (système d'inférence flou), en mettant à la disposition de l'utilisateur trois catégories d'outils fondamentaux:

1. Bloc graphique (GUI interface).
2. Ligne de commande
3. Bloc Simulink.

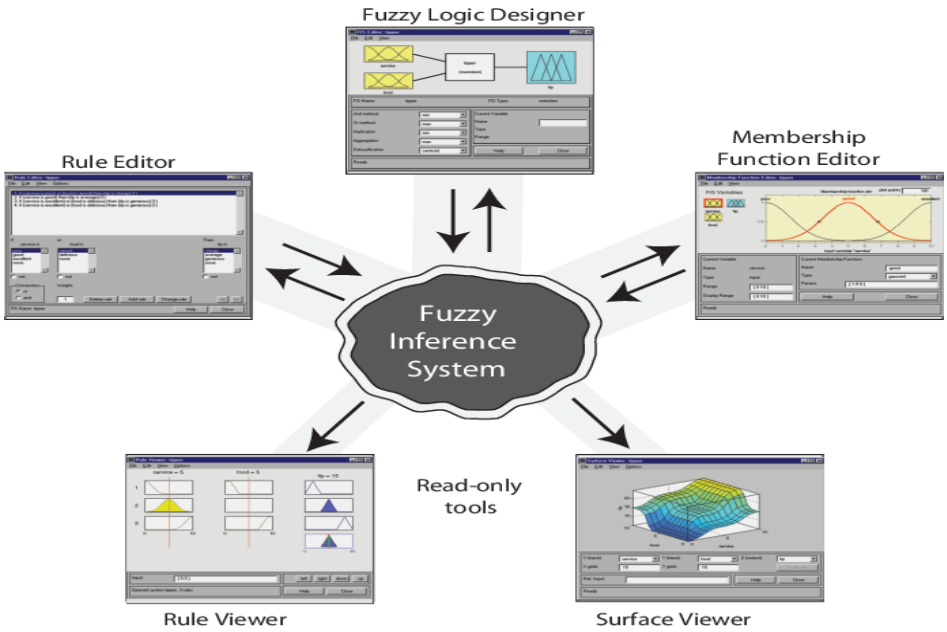



***Figure 2.2*** Présentation du toolbox fuzzy logic

Cette boite à outils possède 3 éditeurs (FIS, Règles, Fonctions d'Appartenance), ainsi que 2 interfaces graphiques (visualisation de la base de règles, surfaces de contrôle). Les principales opérations de SIFs incluent : Fuzzification, Inférence floue, Défuzzification.

Bref aperçu sur les principales commandes matlab correspondantes sont décrites dans le Tableau 1.

| Commande | Action |
|---|---|
| **Fuzzy** | Lancer l'éditeur FIS |
| **Mfedit** | Lancer l'éditeur des fonctions d'appartenance |
| **Ruleedit** | Lancer l'éditeur de regles |
| **Ruleview** | Lancer l'interface graphique d'inférences |
| **Surfview** | Lancer l'interface graphique de visualisation des surfaces de contrôle |

***Table 2.1*** Principales commandes des SIFs

## 2.4 Exemple d'application : Étude comparative entre deux régulateurs PID et FLC appliqués à la Machine à Courant Continu

### 2.4.1. Mise en équation de la machine

La modélisation de l'ensemble moteur + charge est réalisable à partir des équations de base de la machine à courant continu et de la relation fondamentale de la dynamique.

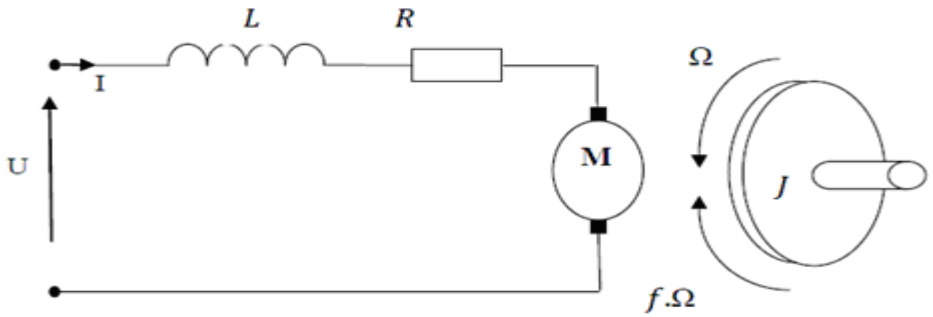



***Figure 2.3.*** Schéma électrique d'une MCC à excitation indépendante

Après calcul, la fonction de transfert d'un moteur à courant continu est :

$$\Omega(p) = \frac{K}{J.L.p^2 + (F.L + R.J).p + F.R + K^2}.U(p) - \frac{L.p + R}{J.L.p^2 + (F.L + R.J).p + F.R + K^2}.C_r$$

## 2.5 Travail demandé

On considère un moteur à courant continu de résistance R = 1,4Ω, d'inductance L= 4,055mH, de constante de fem et de couple K = 0,4095Nm/A. L'inertie de l'ensemble moteur et charge

est J=0,02 kg.m$^2$. Un couple de frottement, proportionnel a la vitesse et de coefficient F=0,0002Nms, s'applique au système. L'entrée du système est la tension u(t); la sortie est la vitesse Ω(t). $C_r$ est le couple de la charge.

- Réaliser à l'aide des blocs Simulink le schéma du moteur a courant continu.
- Réaliser le schéma de commande de vitesse d'un moteur CC associé à un régulateur PI.
- En utilisant la méthode de votre choix, calculer les paramètres du régulateur PI.

On considère que le système est commandé par un système a base de logique floue. La technique retenue consiste à déterminer un correcteur à base de logique floue équivalent au correcteur continu et à simuler le système.

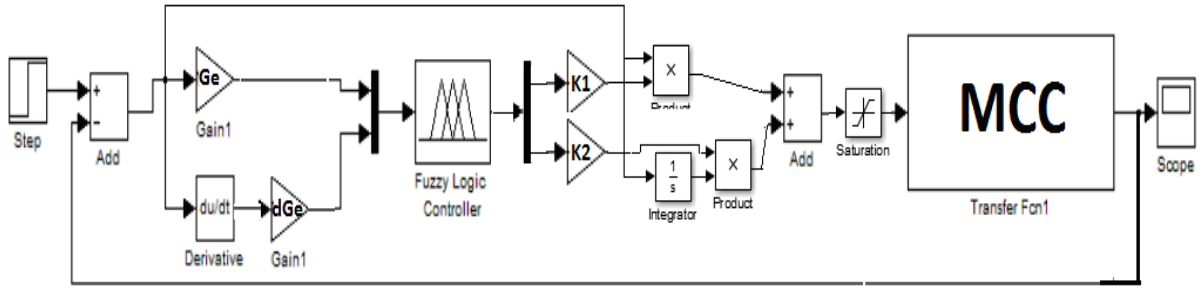



***Figure 2.4.*** Schéma de la commande floue d'une machine a courant continu

On se propose de réaliser **à l'aide des commandes de la boîte à outils** et Simulink, un régulateur a base de flou avec deux entrées : l'erreur (**e**) et sa variation (**de**) et une sortie (**cde**). Par défaut, l'interface propose une entrée et une sortie avec la méthode de mamdani.

**Définition des variables d'entrées : l'erreur et sa variation**

interv_err=[-1 1] ;

interv_derr=[-1 1] ;

**Définition des variables de sorties du régulateur :**

interv_cde1=[0 1] ;

interv_cde2=[0 1] ;

Les coefficients de normalisation sont : Ge=0,001, dGe=7.10$^{-5}$, K1=0,15 et K2=1,9.

**Les règles d'inférences**

Négative Grand (NG), Négative moyen (NM) , Zero (EZ), Positive moyen (PM) et Positive Grand (PG).

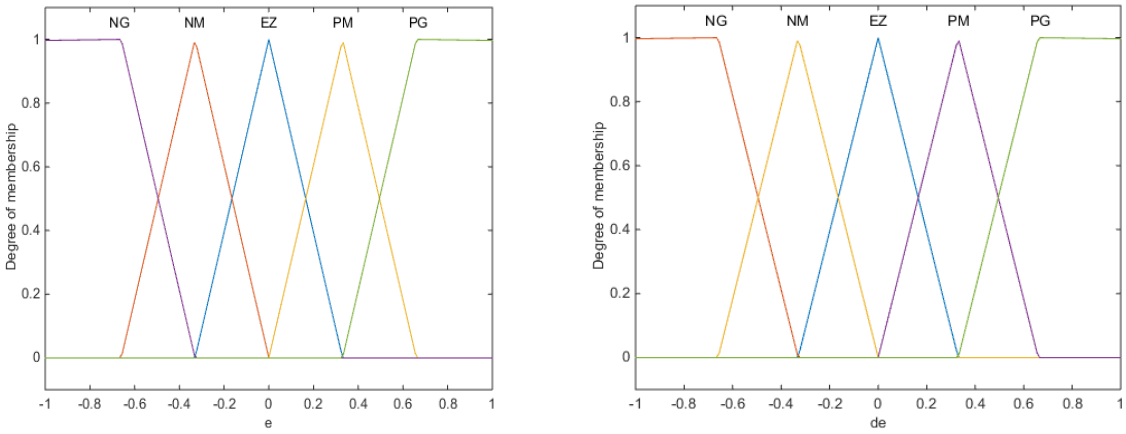



***Figure 2.5.*** Fonctions d'appartenance pour les entrées de régulateur e et de.

Pour la sortie de régulateur :

P : petit, G : grand

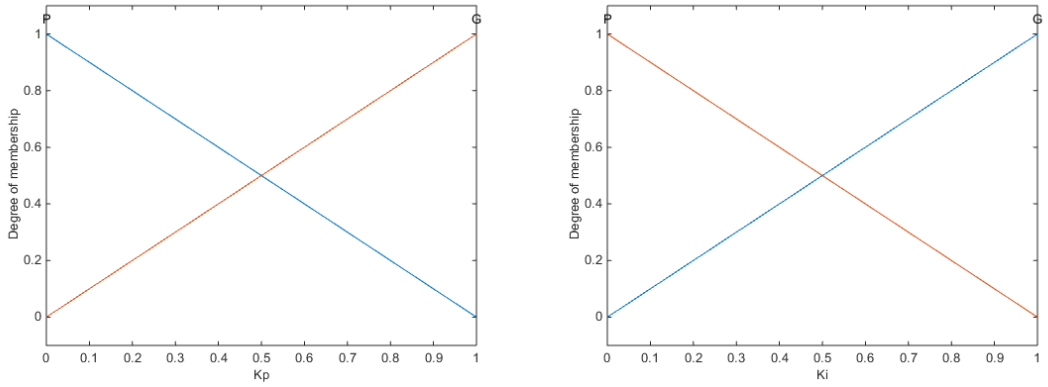



***Figure 2.6.*** Fonctions d'appartenance pour la sortie de régulateur**.**

- Lors de la construction du modèle du système sur SIMULINK, utilisez les données décrites dans les tableaux suivants :

| **Kp** | | de | | | | |
|---|---|---|---|---|---|---|
| | | **NG** | **NM** | **ZE** | **PM** | **PG** |
| *e* | **NG** | **G** | **G** | **G** | **G** | **G** |
| | **NM** | **P** | **G** | **G** | **G** | **P** |
| | **ZE** | **G** | **G** | **G** | **G** | **G** |
| | **PM** | **P** | **G** | **G** | **G** | **P** |
| | **PG** | **G** | **G** | **G** | **G** | **G** |

| **Ki** | | de | | | | |
|---|---|---|---|---|---|---|
| | | **NG** | **NM** | **ZE** | **PM** | **PG** |
| *e* | **NG** | **G** | **P** | **P** | **P** | **G** |
| | **NM** | **G** | **G** | **P** | **G** | **G** |
| | **ZE** | **G** | **G** | **G** | **G** | **G** |
| | **PM** | **G** | **G** | **P** | **G** | **G** |
| | **PG** | **G** | **P** | **P** | **P** | **G** |

***Table 2.2.*** Base de règles utilisée

A partir de l'interface Fuzzy, il faut exporter le fichier vers :

Workspace pour pouvoir l'utiliser sous Simulink dans le schéma de système.

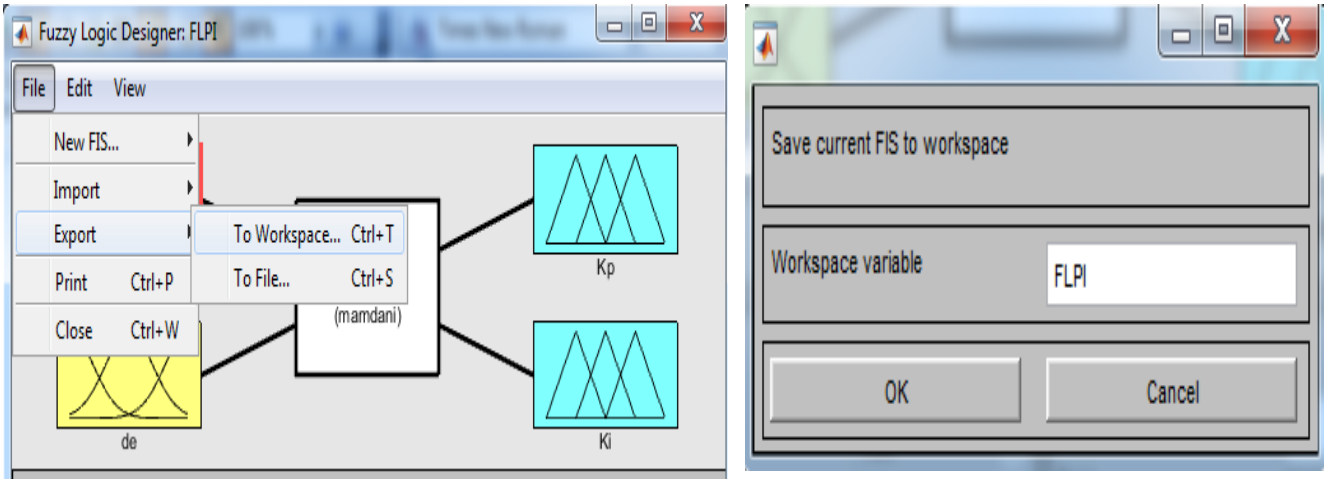



***Figure 2.7.*** Sauvegarde du SIF

On effectue des réglages sur le schéma en réduisant le temps de simulation à 3 secondes, en imposant la vitesse de 150rd/s et en réglant le couple résistant de 5N.m à l'instant 1,5 seconde.

- Visualiser les courbes de la sortie du commande et de la vitesse $\Omega_m(t)$.
- Caractérisez la réponse en terme de temps de réponse, précision et dépassement pour les deux régulateurs.
- Commentez.

TP N° 3

# Initiation aux réseaux de neurones

## 3.1 Objectifs

L'étudiant doit être capable de :

- Se familiariser avec la création d'un réseau de neurones à l'aide des commandes
- Utiliser le Toolbox Neural Network de Matlab pour la manipulation des RNAs.

## 3.2 Introduction

L'architecture des réseaux de neurones est directement inspirée des structures et du fonctionnement du cerveau, d'où le nom de réseaux neuronaux est habituellement utilisé. Les réseaux de neurones se trouve à l'intersection de plusieurs disciplines (informatique, science cognitive, neurobiologie…) et possède des applications dans des nombreux domaines (industrie, télécommunication, informatique, médecine…). Comme dans la nature, le fonctionnement du réseau de neurones est fortement influencé par la connections des éléments entre eux. On peut entraîner un réseau de neurone pour une tâche spécifique (reconnaissance de caractères par exemple) en ajustant les valeurs des connections (ou poids) entre les éléments (neurone).

## 3.3 Présentation des RNAs

Un réseau de neurones artificiel (RNA) est un système de traitement de l'information qui a certaines caractéristiques en commun avec les réseaux de neurones biologiques. En général, un neurone envoie son activation comme un signal à plusieurs autres neurones. Un neurone peut envoyer un seul signal à la fois, bien que ce signal peut être connecté à plusieurs autres neurones. L'apprentissage est la caractéristique principale des RNAs. Il peut être

considéré comme un problème de mise à jour des poids de connexions au sein du réseau. Il existe essentiellement deux types d'apprentissage (supervisé, non supervisé). La *figure 1* montre quelques types des réseaux de neurones artificiels.

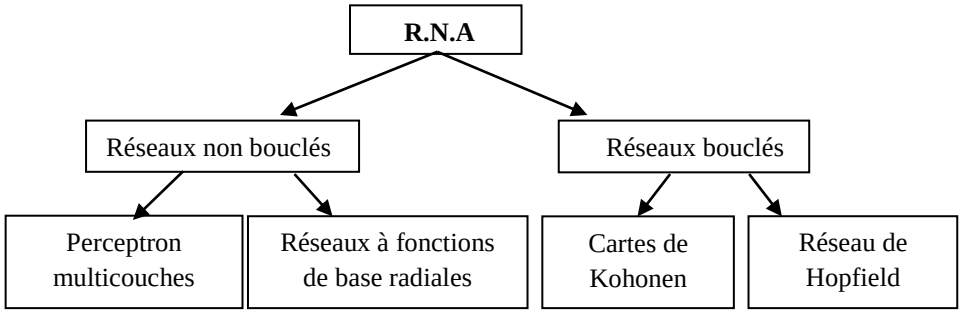



**Figure 3.1** : Principales types de réseaux de neurones

## 3.4 Perceptron multicouches (PMC)

Le perceptron multicouche avec rétro- propagation est l'un des modèles des RNAs les plus largement utilisés. Un exemple du perceptron multicouche est illustré dans la **figure II** au-dessous. La couche la plus à gauche représente la couche d'entrée, elle n'effectue aucun calcul, contrairement aux éléments des autres couches. La couche centrale est la couche cachée (il peut y avoir plus d'une couche cachée dans des réseaux complexes). La couche la plus à droite de neurones est la couche de sortie, qui produit des résultats. Il n'y a pas d'interconnexions entre les neurones dans la même couche, mais tous les neurones dans une couche donnée sont entièrement connectés aux neurones dans les couches adjacentes. Ces interconnexions sont associées à des valeurs numériques (poids) qui sont ajustées au cours de la phase d'apprentissage. La valeur détenue par chaque neurone est appelée activité.

Les étapes d'apprentissage du perceptron multicouches se déroulent comme suite :

1. Présentation d'un exemple parmi l'ensemble d'apprentissage.
2. Calcul de l'état du réseau.
3. Calcul de l'erreur = fct (sortie réelle - sortie désirée)
4. Calcul des gradients par l'algorithme de rétro-propagation de gradient
5. Modification des poids synaptiques
6. Critère d'arrêt : Seuil d'erreur. Nombre de présentation d'exemples, ….
7. Retour en 1.

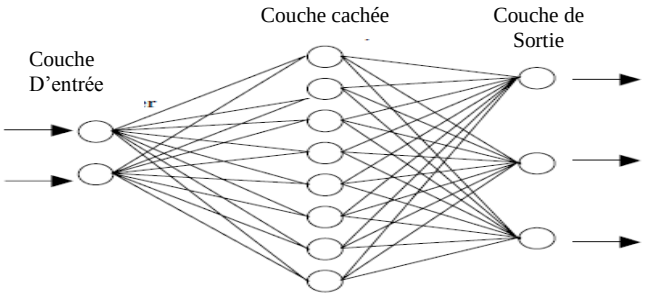



**Figure 3.2** : Exemple de PMC à trois couches

Une des principales capacités d'un réseau de neurones est d'apprendre. Des modèles de lois d'apprentissage ont été réalisé dont la rétro-propagation est la plus connue. Cette technique est basée sur la propagation de l'erreur. La puissance de cet algorithme vient de sa capacité d'ajuster les poids de connexion pour effectuer une tâche particulière. Pour ce faire, cet algorithme utilise une série d'exemples pour l'apprentissage ou chaque vecteur d'entrée est associé à une réponse désirée ou un vecteur de sortie. Ces exemples sont traités un par un par le réseau et les poids sont ajustés en conséquence.

## 3.5 Création d'un perceptron à l'aide des commandes

### 3.5.1 Conception du réseau multi couches :

La création d'un PMC se fait à l'aide de la commande 'newff' qui définit la structure du réseau, le type des fonctions d'activation, le type d'algorithme d'apprentissage et d'autres paramètres.

net=newff (P,T,S,TF,BTF) ;

P : matrice d'entrées,

T : éléments de sortie

S : taille de la $i^{eme}$ couche cachée.

TF : fonction d'activation de la $i^{eme}$ couche cachée.

BTF : fonction d'apprentissage du réseau (trainlm).

La commande 'net.view' permet de visualiser le diagramme du réseau créé.

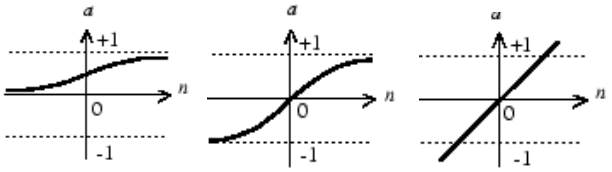
La commande 'net.IW{i,j} permet de récupérer la matrice des poids d'entrées.

La commande 'net.LW{i,j} permet de récupérer la matrice des poids des couches cachées.

La commande 'net.b{i,j} permet de récupérer les valeurs des biais.

- **Fonction d'activation**

  Une fonction d'activation (transfert) permet d'introduire un seuil (saturation). L'algorithme de rétro-propagation fait intervenir la dérivé d'une fonction d'activation.




***Logsig*** : Log-sigmoid transfer function

***Tansig*** : Hyperbolic sigmoid transfer function

***Purelin*** : linear transfer function

**Figure 3.3 :** Fonction Logsig Fonction Tansig Fonction Purelin

### 3.5.2 Apprentissage du réseau :

La deuxième étape consiste à former le réseau. Ceci se fait à l'aide de la commande train.

[net tr]=train (net,P,T)

Les différents algorithmes d'apprentissage du perceptron sous matlab sont :

trainlm : Levenberg-Marquardt backprobagation.

traingd : Gradient descent backpropagation.

traingda : Gradient descent with adaptive learning rate backpropagation.

trainbfg : Quasi-Newton backpropagation

- **Paramètres d'apprentissage :**

Il est utile de définir les paramètres associés à l'algorithme d'apprentissage

net.trainParam. epochs : fixer le nombre maximum d'itérations.

net.trainParam.goal: fixer le seuil d'erreur.

net.trainParam.time: fixer le temps d'apprentissage (en seconde).

net.trainParam.lr: fixer le pas d'apprentissage.

### 3.5.3 Tester le réseau :

La commande 'sim' permet de simuler le RNA.

Sortie_reseau=sim (net,P1)

P1 : Nouvelle ensemble de données.

La commande 'save' permet de sauvegarder un réseau qui est déjà créé.

save nom_reseau net

Pour appeler ce réseau, on utilise la commande suivante :

net=load nom_reseau

## 3.6 Utilisation de l'outil réseau de neurone

Taper la commande 'nftool' pour ouvrir l'outil d'ajustement des RNAs puis cliquer 'Next' pour procéder.

A partir de l'interface graphique mentionnée ci-dessous, cliquer sur 'Load Example Data Set'.

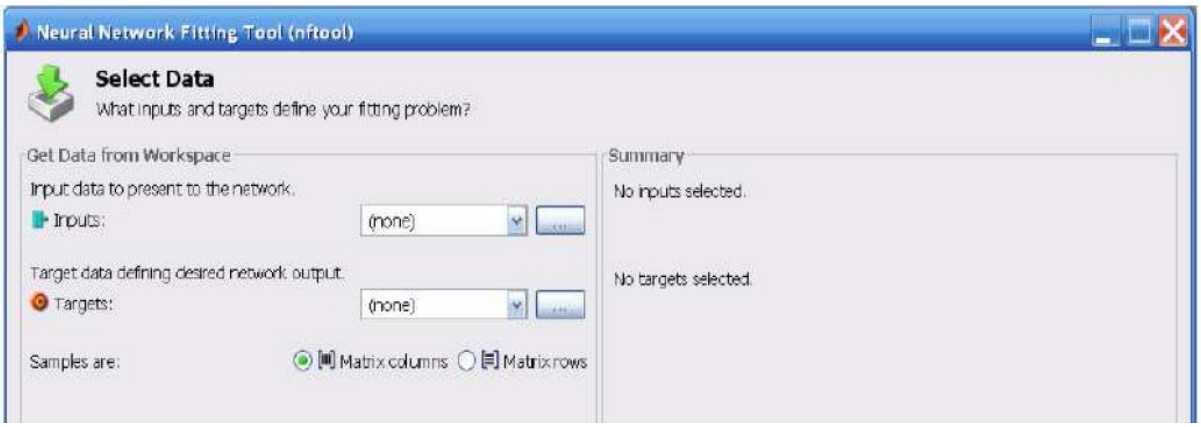



**Figure 3.4** : Sélection des données

La fenêtre 'Fitting Dataset Chooser' s'ouvre. On peut aussi utiliser les options 'inputs' et 'outputs' de cette figure à fin d'importer les données à partir de Matlab workspace. Comme exemple sélectionner la base 'Body Fat' puis cliquer 'importer' pour retourner à la fenêtre 'Select Data'.

Cliquer 'Next'. La fenêtre 'Validation and Test Data' apparait comme illustré ci-dessous. Prenons comme exemple 10% pour les ensembles validation et 20% pour les données de test.

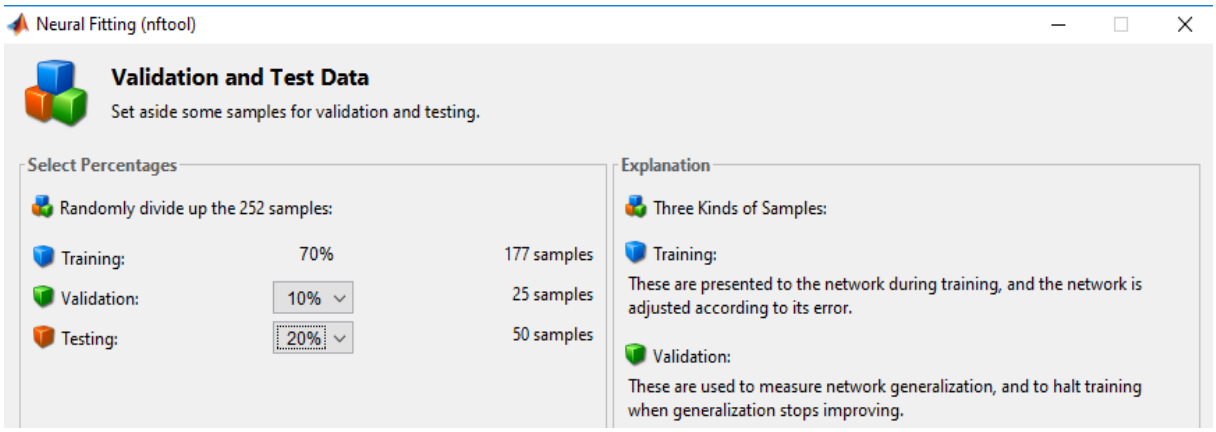



**Figure 3.5** : Partition des données

Ceci permet aux données d'entrées d'être subdivisé aléatoirement en trois parties :

70% : sera utilisé pour l'apprentissage.

10% : sera utilisé pour valider le réseau.

20% : sera utilisé pour un test indépendant de la généralisation du réseau.

Cliquer sur ‘Next’ pour procéder.

Le réseau standard utilisé est un réseau feedforward a deux couches (une couche cachée, une couche de sortie), dont la fonction d’activation ‘sigmoid’ est utilisé dans la couche cachée et

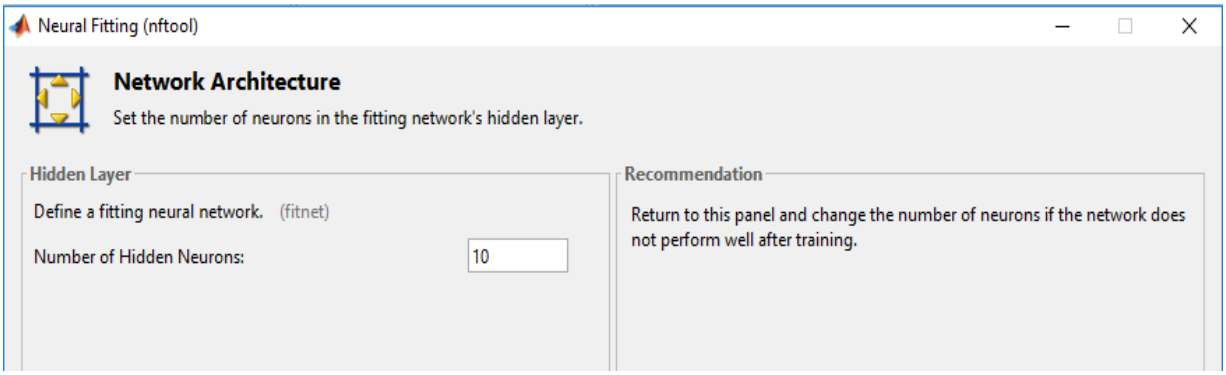



**Figure 3.6** : Choix des paramètres du réseau

la fonction de transfert ‘linéaire’ dans la couche de sortie. Le nombre par défaut des neurones cachés est 10.

Vous pourriez ajuster ce nombre, si le réseau créé possède des performances insatisfaisantes.

Appuyer sur ‘Next’ pour créer le réseau

Cliquer sur ‘train’ pour former le réseau créé.

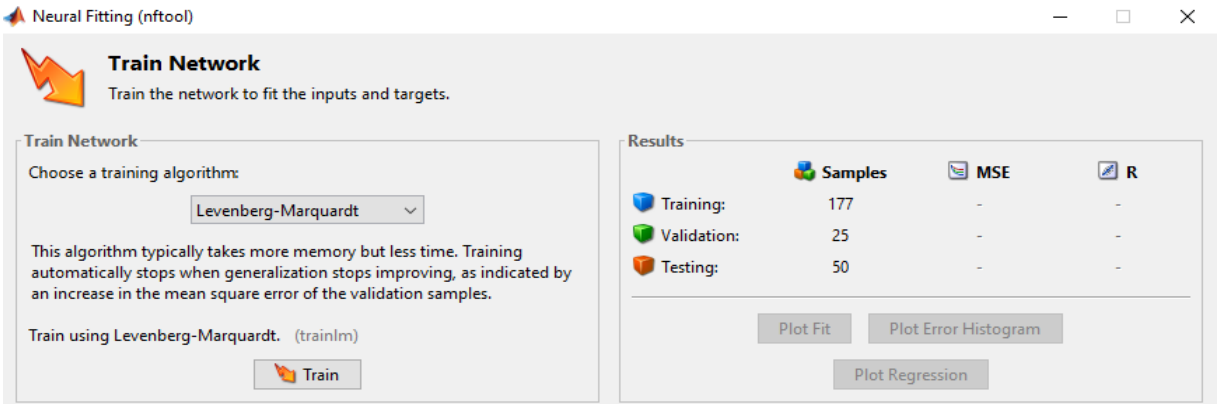



**Figure 3.7** : Apprentissage du réseau

La fenêtre Neural Network Training s’affiche. Cliquer sur ‘Regression’ pour valider la performance du réseau.

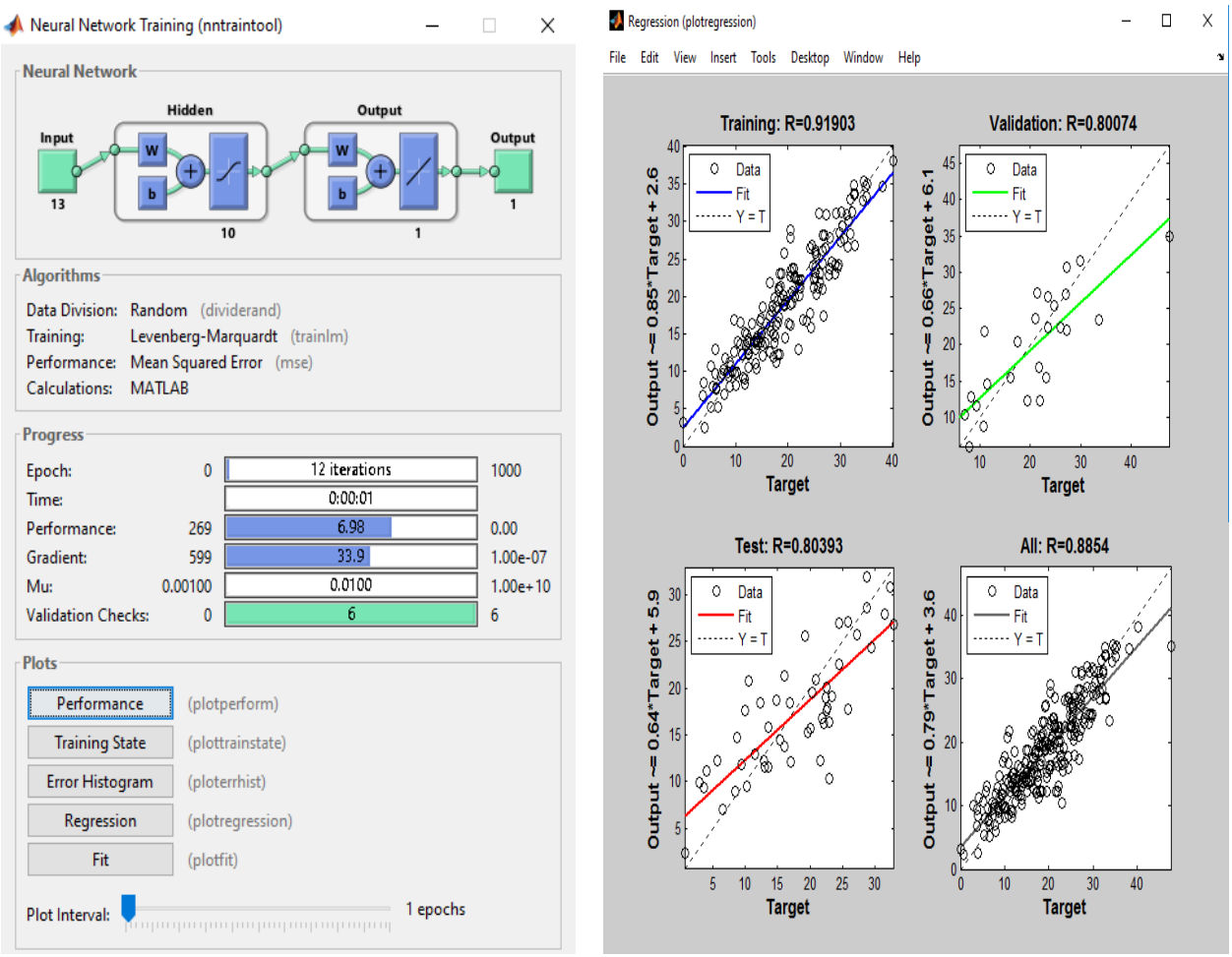



**Figure 3.8** : Résultats du réseau neuronale

Vous pourriez considérer l'histogramme d'erreur pour obtenir plus d'informations sur la performance du réseau en cliquant sur 'Error Historgram'.

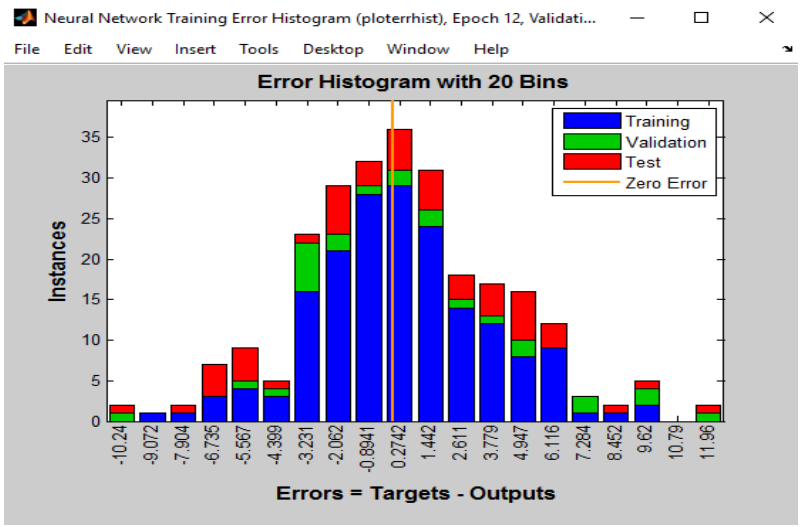



**Figure 3.9** : Histogramme des erreurs

A ce stade, vous pouvez tester votre réseau sur des nouvelles données. Vous pouvez augmenter le nombre des neurones cachés et former le réseau à nouveau si le réseau formant la performance est pauvre, en cliquant sur 'Adjust Network size' et 'Train Again' respectivement. Cliquer 'Next' si vous êtes satisfaits de la performance de réseau afin de sauvegarder vos résultats.

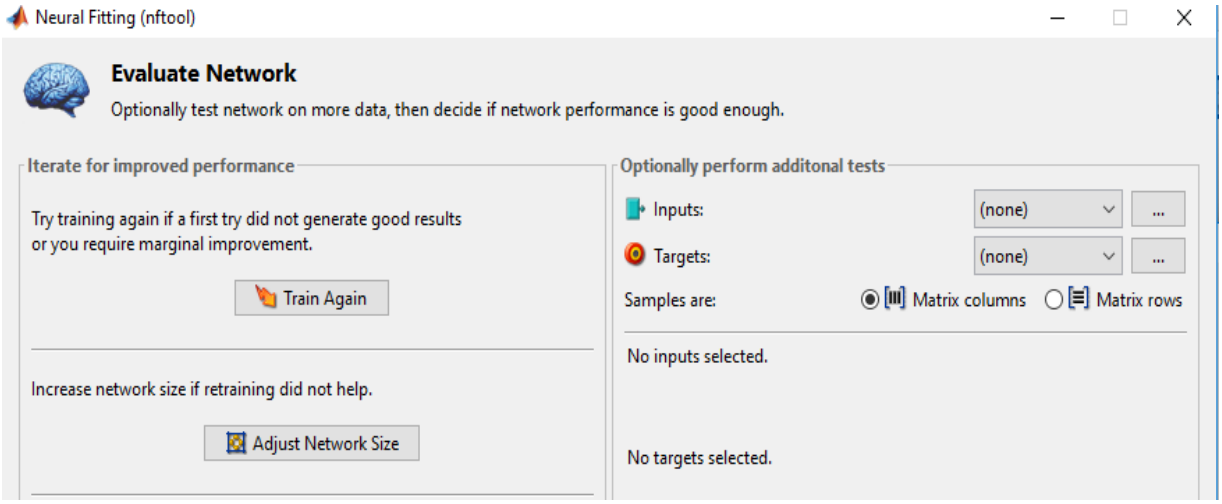



**Figure 3.10** : Evaluation du réseau

Vous pouvez créer automatiquement le code Matlab pour reproduire les étapes précédentes, sauver votre réseau dans Matlab workspace, ou même exécuter d'autres tests sur de nouveaux entrés.

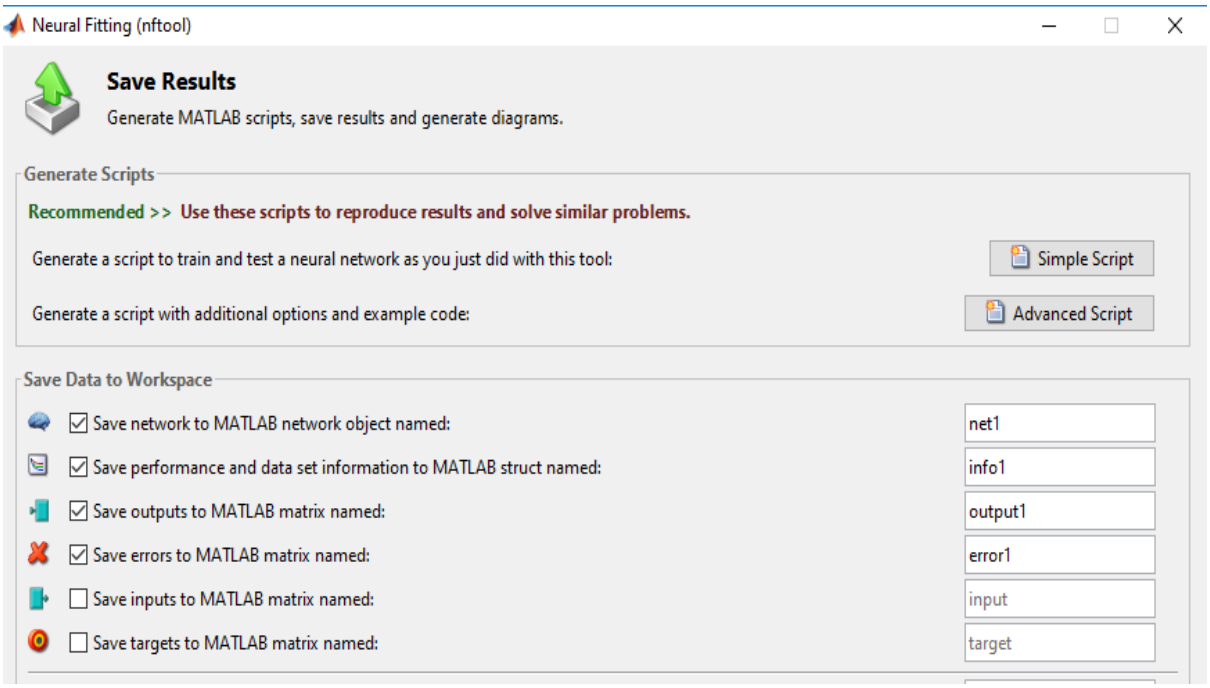



**Figure 3.11** : Sauvegarde du réseau

## 3.7 Travail demandé :

**1.** Etant donnée un RNA créé comme suite :

net=newff([-6 6 ; -5 2], [0 1],1)

a. Tracer l'architecture du réseau.
b. Afficher les valeurs de poids et biais.
c. Affecter les valeurs 2, 3, 5 au poids et 5,2 au biais.
d. Afficher à nouveau les valeurs de poids et biais.

**II**. On se propose de réaliser un classifieur neuronale **(à l'aide des commandes matlab)** à fin de prédire un signal.

Soit : P=0 :0.02 :3 ; T=0.5*abs(sim(5*P)*exp(-.5*P)+cos(2*P))

a. Tracer la courbe du signal en utilisant les commandes :
   Plot (P,T,'x') ; xlabel ('Entrées') ; ylabel('Sortie') ; title('Apprentissage PMC') ;
b. Créer un PMC avec deux couches cachées contenant 5 neurones cachés dans la 1ere et 2 neurones dans la 2 eme.
c. Afficher la structure du réseau.
d. Faite l'entrainement du PMC et tracer la courbe d'évolution d'erreur.
e. Tester votre réseau sur les mêmes données et tracer la courbe d'évolution d'erreur.
f. Tester votre réseau sur des nouvelles données :
   P1= 0 :0.01 :3 ; T1 = 0.5*abs (sim(5*P1)*exp(-.5*P1)+cos(2*P1))
g. Tracer la courbe d'évolution d'erreur (Apprentissage et Test).
h. Comparer les résultats d'apprentissage et de test.
i. Améliorer la performance de votre PMC :
   Augmenter le nombre des neurones cachés.
   Essayer d'autre fonction d'apprentissage
   Jouer sur le seuil d'erreur
   …
j. Que remarquez-vous ???

**III. Exercices supplémentaires :**

1. Trouver l'approximation du signal sinusoïdal suivant à l'aide du RNA.
   X=(0 :0.0001 :0.05)
   Y= sin (100*pi*X-2*pi*0.75)

2. Même travail pour

   X=(0 :0.1 :5)

   Y=erf(X)

TP N° 4

# Regression à base des RNAs

## 4.1 Objectifs

- Etudier les modèles connexionnistes PMC (Perceptron Multi-Couches) et RBF (Réseaux à Fonctions de Base Radiales) en problème de régression.
- Analyse et correction neuronale d'un Système de Régulation Automatique.

## 4.2 Présentation du RBF

Les réseaux à fonctions de base radiales (RBF) sont des modèles connexionnistes très utilisés pour la régression et la discrimination. Il constitue avec le Perceptron multicouche, un modèle connexionniste le mieux connu. Ces réseaux avec une seule couche cachée peuvent approximer n'importe quelle fonction continue ayant un nombre fini de discontinuités sur tout compact. Les réseaux à fonction radiale RBF sont très semblables à celle des PMC mais leur particularité est qu'ils sont caractérisés par 1'utilisation des fonctions Gaussiennes comme fonctions de base. Les règles d'apprentissage les plus utilisées pour les RBF sont la règle de correction de l'erreur ou la règle d'apprentissage par compétition.

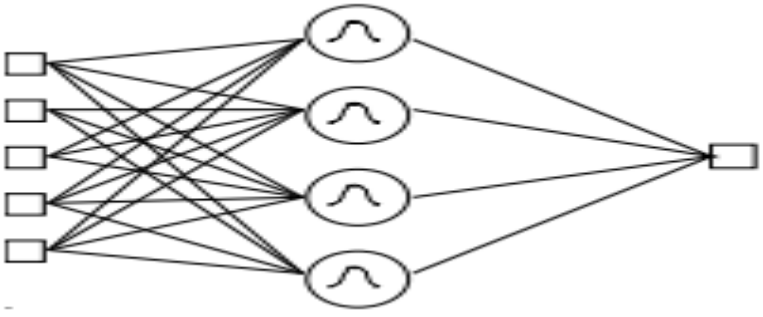


**Figure 4.1** : Représentation d'un réseau RBF

## 4.3. Création d'un réseau RBF à l'aide des commandes

### 4.3.1 Conception et apprentissage du réseau RBF :

Un exemple d'un RBF est illustré dans la figure 4.1. La conception d'un RBF se fait à l'aide de la commande 'newrb' qui définit les paramètres du réseau.

net= newrb(P,T,goal,spread,MN,DF);

P : matrice RxQ de Q vecteurs d'entrée.

T : matrice SxQ de Q vecteurs cibles de classes.

goal : mse (sum-squared error goal) objective. Par défaut = 0.0

spread : propagation des fonctions RB (spread constant) . Par défaut = 1.0

MN : Nombre maximal de neurones.

DF : Nombre de neurones à rajouter entre chaque évaluation . Par défaut 25.

La commande 'net.view' permet de visualiser le diagramme du réseau créé.

La commande 'net.IW permet de récupérer la matrice des poids d'entrées.

La commande 'net.LW permet de récupérer la matrice des poids de la couche cachée.

La commande 'net.b permet de récupérer les valeurs des biais.

### 4.3.2 tester le réseau :

La commande 'sim' permet de simuler le réseau RBF.

Sortie_reseau=sim (net,P1)

P1 : Nouvelle ensemble de données.

La commande suivante peut être aussi utilisée :

Sortie_reseau= net(P1)

La commande 'save' permet de sauvegarder un réseau qui est déjà créé.

save nom_reseau net

Pour appeler ce réseau, on utilise la commande suivante :

net=load nom_reseau

### 4.3.3 Evaluation du réseau :

La commande 'perform' permet d'évaluer le réseau RBF.

perform(net,T,Sortie_reseau)

## 4.4 Utilisation de l'outil réseau de neurone

Pour commencer, taper la commande **'nntool'** dans la ligne de commande de MATLAB. La fenêtre suivante constituée de plusieurs champs (Input Data, Target Data, Networks, Output Data..) apparaît.

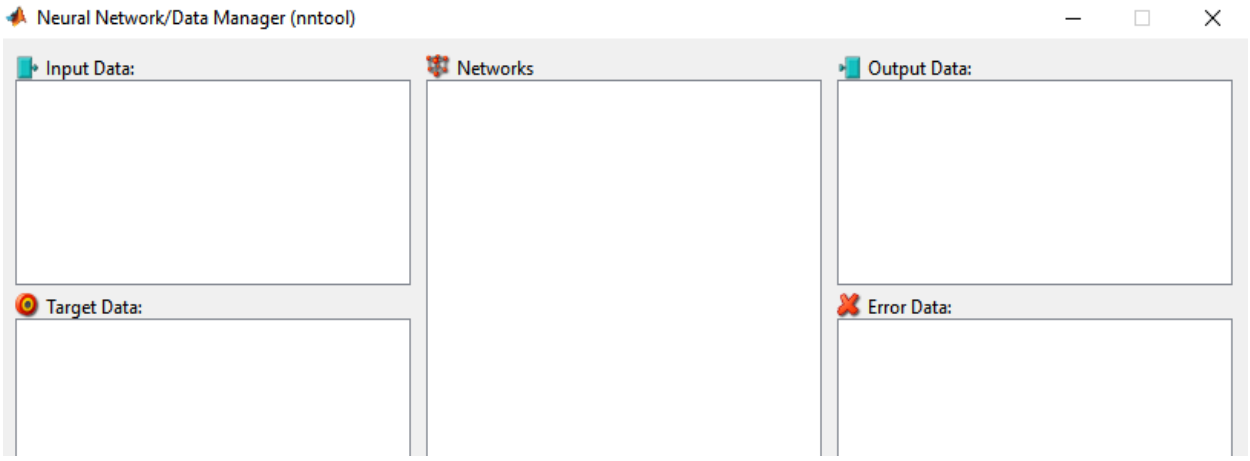



**Figure 4.2** : La fenêtre principale nntool

A partir de l'interface graphique mentionnée ci-dessus, cliquer sur '**Import**'. La fenêtre **'Import to Network/Data Manager'** s'ouvre.

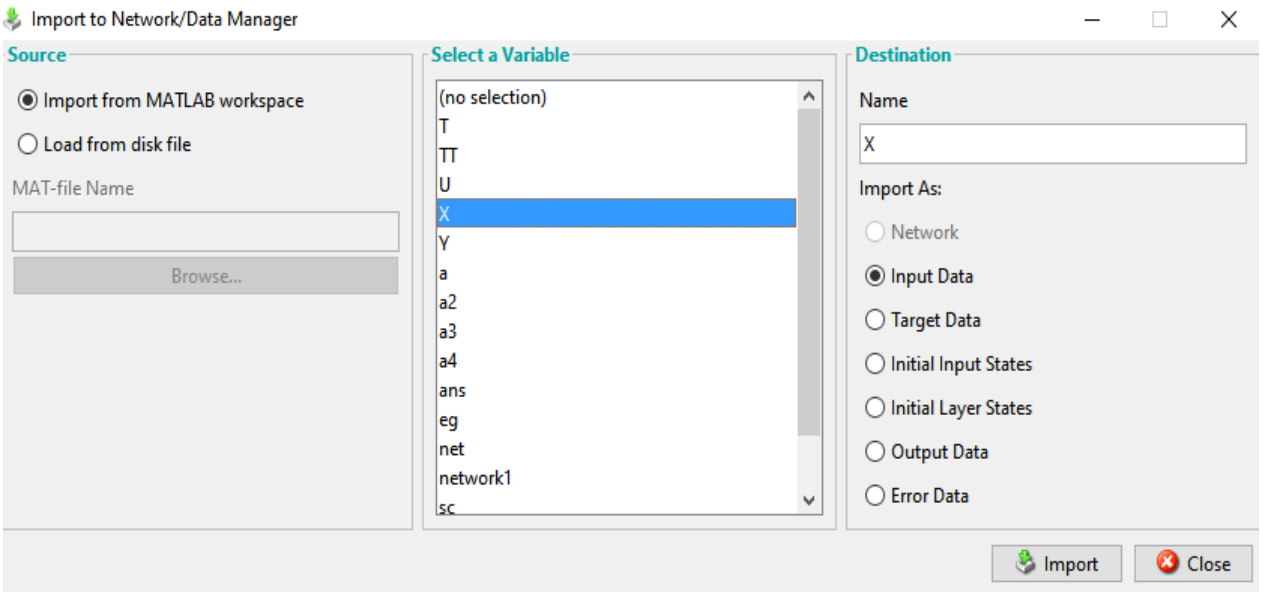



**Figure 4.3** : Importation des données

Avant de créer un réseau il faut d'abord introduire la base de donnée en cliquent sur **'load from disk file'**. On peut aussi utiliser les options 'Input Data' et 'Target Data' de cette figure a fin d'importer des données à partir de Matlab workspace. Comme exemple sélectionner la variable 'X' (données d'entées) et 'Y' (classe cible) puis cliquer 'importer'.

Avec X = -0.5:.1:0.5 ; Y= [ .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960]

Cliquer '**New'**. La fenêtre '**Create Network or Data'** apparaît comme illustré ci-dessous .On donne un nom au réseau (exemple : net1) puis on choisit les paramètres du réseau (type :

RBF ; input data : X ; Target data :Y ; Goal : 0.0 ; Spread : 10). Cliquer '**View**' pour visualiser l'architecture du réseau créé.

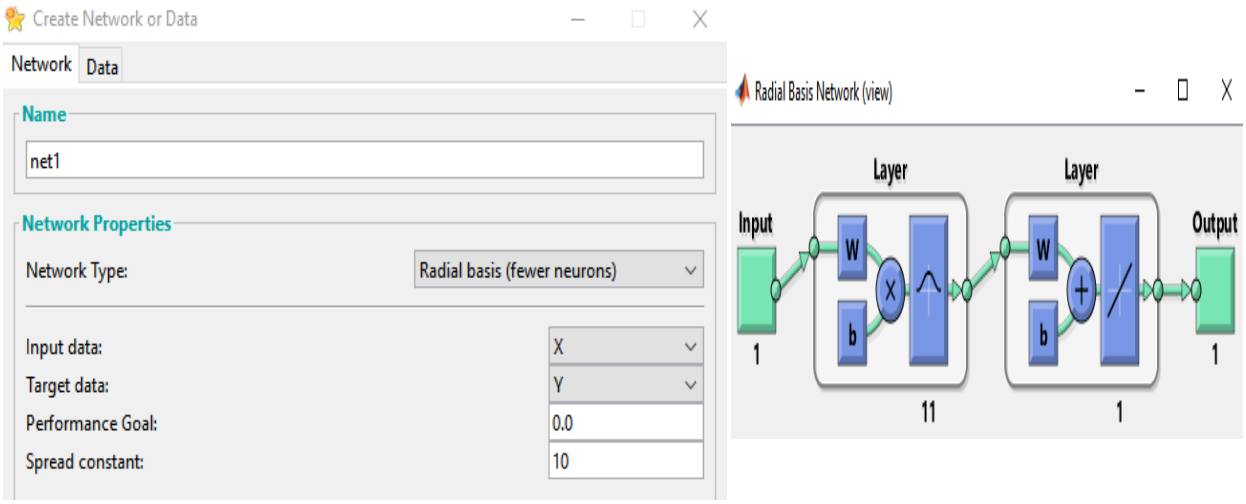



**Figure 4.4** : Création du réseau RBF

Cliquer sur '**Create**' pour procéder puis '**Close**' pour revenir à la fenêtre principale.

Sélectionner le nom de réseau **(net1)** sur le champ **Networks** de l'interface graphique puis appuyer sur '**Open**'. Une nouvelle fenêtre apparaît (Network : net1).

A ce stade, vous pouvez tester votre réseau sur des nouvelles données. Sélectionner l'onglet **'Simulate'** en modifiant les paramètres des données comme illustré dans la figure suivante :

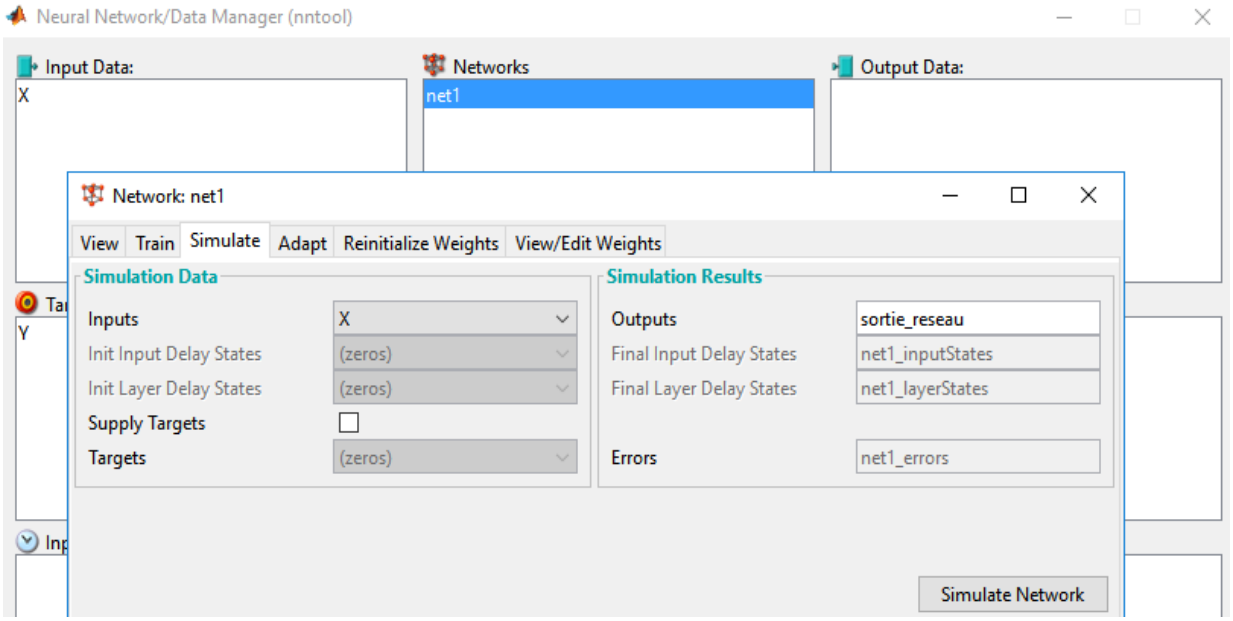



**Figure 4.5** : Teste du réseau

Appuyer sur **'Simulate Network'** pour tester le réseau créé.

Vous pouvez sauver vos résultats dans Matlab workspace, en cliquant sur **'Export'**. La fenêtre **'Export from Network'** s'affiche. Cliquer sur **'Select All'** puis **'Export'**.

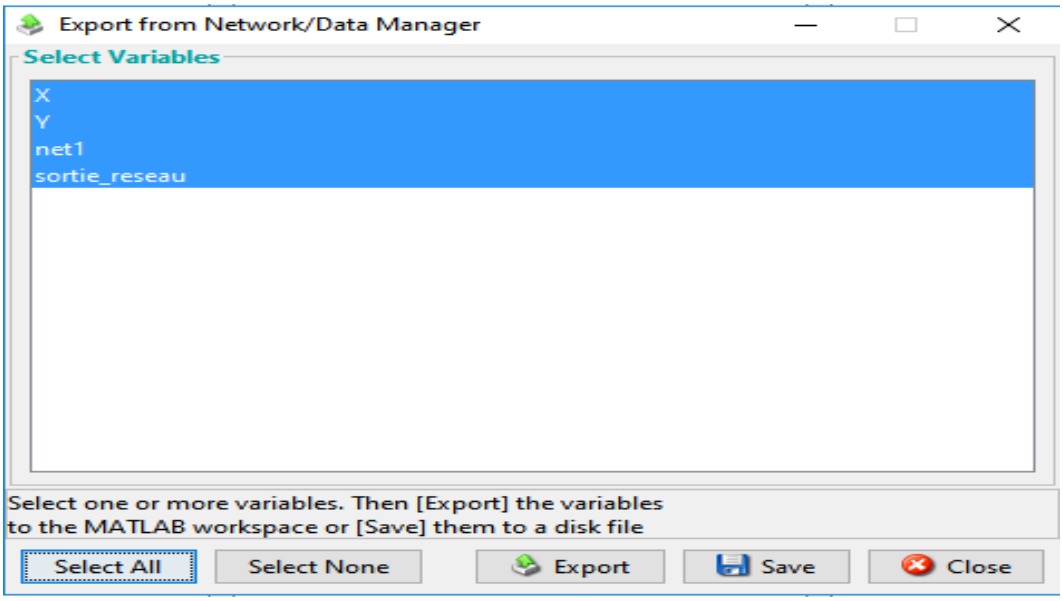



**Figure 4.6** : Sauvegarde du réseau

Cliquer sur le nom de la sortie **'sortie_reseau'** dans le champ '**Output Data'** de la fenêtre principale, la fenêtre des résultats apparaît .Ces valeurs représentent les résultats obtenus lors de la simulation.

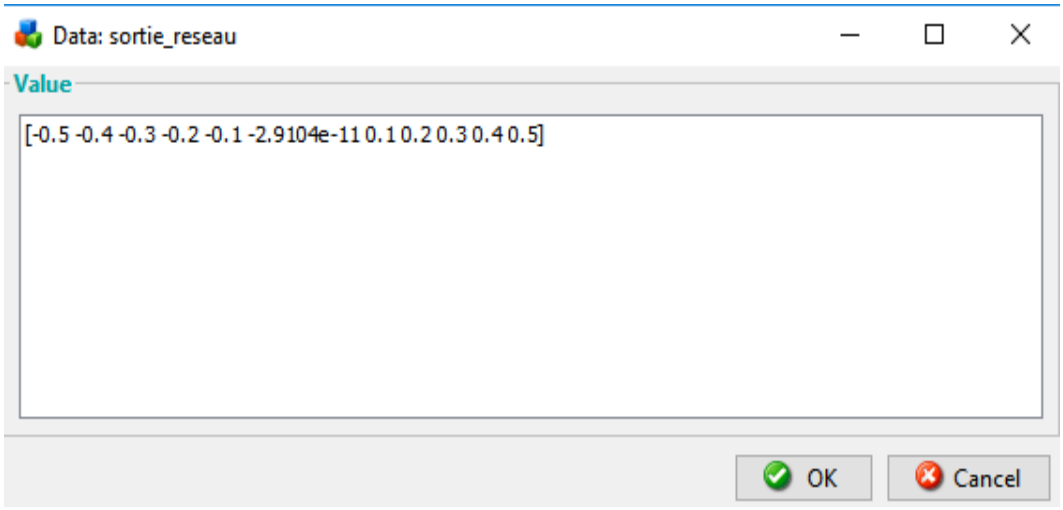



**Figure 4.7** : Résultats de la simulation

Vous pouvez copier ces valeurs dans le Workspace de Matlab, pour les utiliser lors du calcul de la performance du réseau RBF.

### 4.5. Travail demandé

**4.5.1** Trouver l'approximation du signal sinusoïdal suivant à l'aide du RBF.

X=(0 :0.0001 :0.05)

Y= sin (100*pi*X-2*pi*0.75)

**4.5.2** Dans cette partie, nous allons étudier la base de données RobotArm. C'est un data set constitué de 1463 exemples, et qui peut être utilisé pour former un réseau de neurones de type RBF, a fin de modéliser un bras robotique. Pour plus d'informations sur la base de donnée voir Annexe 1.

- Tout d'abords, il faut diviser la base de données en deux parties : apprentissage et test.
- La partie apprentissage sera utilisée pour entrainer le RBF.
- La partie test est utilisée pour évaluer le RBF ainsi appris.
- Réglages de l'apprentissage RBF : effectuer des tests avec différentes valeurs de 'spread', 'MN' et 'DF' pour l'apprentissage du RBF.
- Evaluer les performances du réseau RBF.
- En demandant de visualiser les erreurs, retrouver les instances sur lesquelles le modèle se trompe. Peut-on trouver les raisons des mauvaises prédictions.
- Refaites les expériences en modifiants 'newrb' par 'newrbe'. Obtenez-vous une amélioration des performances ?
- Utiliser le modèle PMC avec les mêmes expériences. Est-ce que les erreurs de prédiction commises sont les même que pour le RBF ? Expliquer…

**4.5.3** Dans cette partie, nous développons un système qui simule un modèle de contrôle d'un système de fonction du transfert :

$$G = \frac{5}{(s+1)(s^2+2s+3)} e^{-0.5s}$$

- Lancer le simulink par une instruction sur la ligne de commande MATLAB : Simulink.
- Réalisez le schéma de simulation de la figure 4.8 sous Simulink.

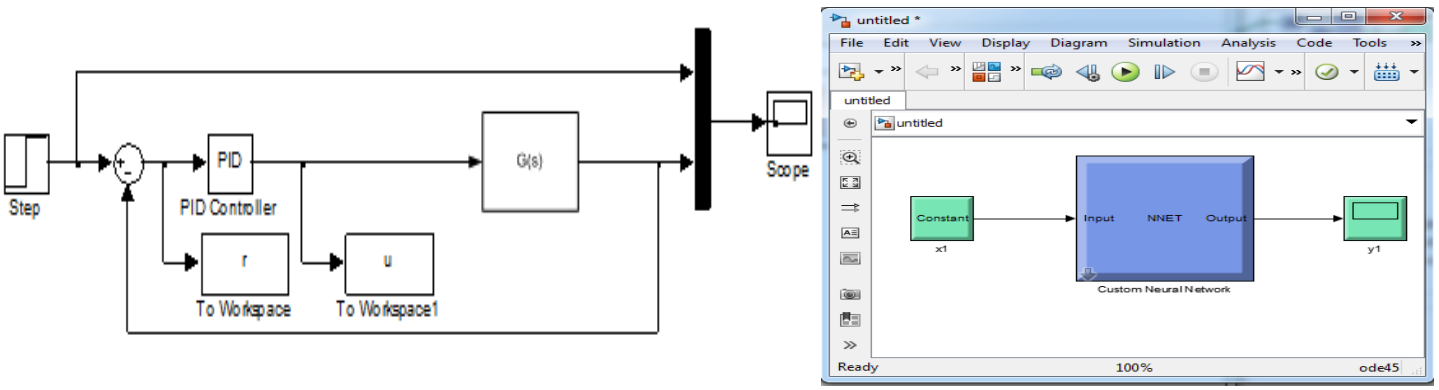



**Figure 4.8** Implémentation sur Simulink d'un modèle de contrôle

- Lors de la construction du modèle du système sur SIMULINK, utilisez les données décrites dans le tableau 4.1.
- Définissez les vecteurs de données d'entrée et de sortie pour les réseaux de neurones.

```
p=r';
t=u';
net=newff([-1 1],[20 10 1],{ 'tansig' 'tansig' 'purelin'});
net.trainparam.epochs=1000;
net.trainparam.goal=0.0001;
net=train(net,p,t);
```

- Pour passer du côté commande à SIMULINK, utilisez la commande gensim. Cela transférera les informations sur le réseau neuronal vers SIMULINK et en même temps, il générera automatiquement un fichier SIMULINK avec le bloc de réseau neuronal. Le deuxième argument est utilisé pour définir le temps d'échantillonnage. Pour un échantillonnage continu, la valeur est –1.

```
gensim(net,-1)
```

- Remplacer la commande PID par la commande neuronale créer par la figure précédente.
- Tester ces régulateurs sous Simulink.
- Faire varier le nombre de neurone ainsi que le nombre de couches et tester ce régulateur.

| Composant | Paramètre | Valeur |
|---|---|---|
| **Step** | | 1 |
| **Kp** | | 0,38 |
| **Ki** | | 0,285 |
| **Kd** | | 0,1 |

**Table 4.1.** Valeurs des paramètres du Modèle Simulink

**Annexe :**

**Robotarm_dataset (**Robot arm dataset)

Input-output time series problems consist of predicting the next value of one time-series given another time-series. Past values of both series (for the best accurary), or only one of the series (for a simpler system) may be used to predict the target series.

This dataset can be used to train a neural network to model a robot arm.

robotarmInputs – a 1*1463 cell array of scalar values representing 1463 timesteps of robot aram control signals.

TP N° 5

# Initiation aux modèles neuro-flous

## 5.1 Objectifs

Buts de ce TP est :

- Se familiariser avec la création d'un réseau neuro-flou à l'aide des commandes.
- Utiliser le Toolbox Fuzzy Logic de Matlab pour la manipulation des systèmes neuro-flou.

## 5.2 Introduction

Les réseaux de neurones et la logique floue sont deux approches qui sont très utilisées pour résoudre les problèmes de classification ou prédiction. L'avantage principal des réseaux de neurones réside dans leurs capacités d'apprentissage et leurs facilités d'implémentation, par contre la non interprétablilité de ces résultats constitue un inconvénient majeur (boîte noire). Les systèmes d'inférence flous permettent d'interpréter leurs résultats grâce à leur base de connaissances (base de règles). L'utilisation conjointe des réseaux de neurones et les systèmes d'inférence flous permettent d'exploiter les avantages des deux méthodes. Plusieurs types d'hybridation de la logique floue et les réseaux de neurones existent telle que : Nefcon, Falcon, Fun, et ANFIS.

## 5.3 Présentation du modèle ANFIS

Le modèle Anfis (**A**daptative **N**euro-**F**uzzy **I**nference **S**ystem) est un réseau de neurone flou proposé en 1993 par Jang**,** comporte cinq couches, représenté dans la **figure I**. Les nœuds sont de deux types différents selon leur fonctionnalité : des nœuds adaptatifs (carrés) et des nœuds fixes (circulaires**).** Pour simplifier la compréhension, nous considérons un modèle

composé de deux entrées (x et y), une seule sortie globale (f) et de deux règles. Tel qu'il est montré dans la **figure 5.1**

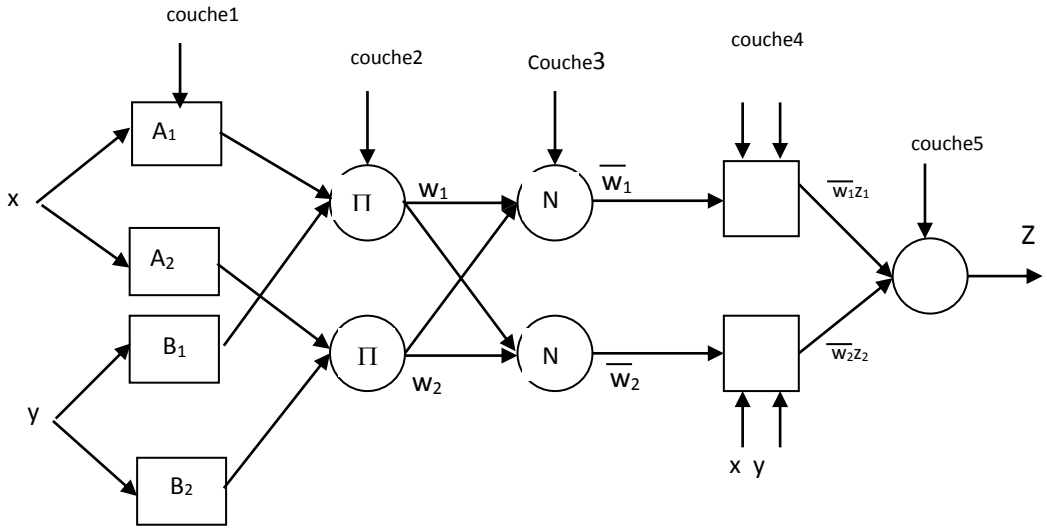



**Figure 5.1** Architecture de l'ANFIS

L'algorithme de l'ANFIS est résumé comme suite :

$$Rule\,1 : si\ x\ est\ A_1\ et\ y\ est\ B_1, alors\ z_1 = p_1 x + q_1 y + r_1$$
$$Rule\,2 : si\ x\ est\ A_2\ et\ y\ est\ B_2, alors\ z_2 = p_2 x + q_2 y + r_2$$

La sortie $O^k_i$ d'un nœud i de la couche k (appelée *noeud (i, k)*) dépend des signaux provenant de la couche k-1 et des paramètres du nœud (i, k) .

✓ *Couche 1* : Les nœuds de cette couche sont tous de types adaptatifs. Cette couche réalise la fuzzification des entrées c'est à dire qu'elle détermine les degrés d'appartenance de chaque entrée ($O_i^1$):

$$O_i^1 = \mu_{A_i}(x), \qquad i = 1,2$$
$$O_i^1 = \mu_{B_{i=2}}(y), \qquad i = 3,4$$

✓ *Couche 2* : Les nœuds de cette couche sont des nœuds fixes. Cette couche engendre le degré d'activation d'une règle. Ceci dépend des opérateurs présents dans les règles (ET ou OU).

$$O_i^2 = w_i = \mu_{A_i}(x)\mu_{B_i}(y) \qquad i = 1,2$$

✓ *Couche 3 :* Chaque neurone dans cette couche calcule le degré de vérité normalisé d'une règle floue donnée. Le résultat à la sortie de chaque nœud représente la contribution de cette règle au résultat final.

$$O_i^3 = \overline{w_i} = \frac{w_i}{w_1 + w_2} \quad i = 1,2$$

✓ *Couche 4 :* Les noeuds dans cette couche sont des noeuds adaptatifs. Chacun de ces noeuds est relié à un neurone de normalisation correspondant et aux entrées initiales du réseau. La sortie d'un noeud i est donnée par :

$$O_i^4 = \overline{w_i} z_i = \overline{w_i}(p_i x + q_i y + r_i) \qquad i = 1,2$$

Où $\overline{w_i}$ est la sortie de la troisième couche, et $\{p_i, q_i, r_i\}$ est l'ensemble des paramètres. Ces paramètres sont appelés *les paramètres conséquents*.

✓ *Couche 5 :* Cette couche comprend un seul neurone circulaire qui effectue la somme des signaux provenant de la couche précédente pour donner la sortie finale du réseau :

$$O_i^5 = \sum_{i=1}^{2} \overline{w_i} z_i = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}$$

La sortie globale peut être exprimée comme combinaison linéaire des paramètres conséquents:

$$z = (\overline{w_1} x) p_1 + (\overline{w_1} y) q_1 + (\overline{w_1}) r_1 + (\overline{w_2} x) p_2 + (\overline{w_2} y) q_2 + (\overline{w_2}) r_2$$

## 5.4 Apprentissage du modèle ANFIS

L'ajustement des paramètres de l'ANFIS se fait par une méthode hybride, qui représente une association de la méthode de descente de gradient et de la méthode d'estimation des moindres carrée. La méthode de descente de gradient permet d'ajuster les prémisses en fixant les paramètres conséquents alors que la méthode LSM (Least square Method) ajuste les paramètres conséquents en fixant les prémisses.

## 5.5 Création d'un ANFIS à l'aide des commandes

### 5.5.1 Conception du réseau ANFIS :

La création d'un système neuro-flou se fait à l'aide de la commande « genfis1» qui génère un système d'inférence floue à partir des données d'apprentissage et d'autres paramètres.

infis=genfis1 (data, numMF, inmftype, outmftype)

data : données d'apprentissage.

numMF: nombre des fonctions d'appartenance pour chaque entrée.

inmftype : type des fonctions d'appartenance pour chaque entrée.

outmftype : type des fonctions d'appartenance pour chaque sortie.

### 5.5.2 Apprentissage du réseau ANFIS:

La deuxième étape consiste à former le réseau. Ceci se fait à l'aide de la commande anfis.

fis=anfis (trnData, ,initFis)

trnData : Données d'apprentissage.

initFis: Structure initiale du SIF (créer à l'aide de la commande genfis1)

La syntaxe générale avec les 6 arguments est :

[fis, error, stepsize, chkFis, chkErr] = anfis(trnData, initFis, trnOpt, dispOpt, chkData, optMethod)

trnopt : Options d'apprentissage

T_OPT(1): nombre d'epochs (par défaut: 10)

T_OPT(2): seuil d'erreur (par défaut: 0)

T_OPT(3): pas d'apprentissage initiale (par défaut: 0.01)

T_OPT(4): taux de diminution du pas (par défaut: 0.9)

T_OPT(5): taux d'incrémentation du pas (par défaut: 1.1)

dispOpt : Options d'affichage

dispOpt(1): information sur l'ANFIS information (par exemple : nombre des fonctions d'appartenances des entrées et de sorties )

dispOpt(2): erreur (par défaut: 1)

dispOpt(3): résultat à chaque mise à jours des paramètres (par défaut: 1)

dispOpt(4): résultats finaux (par défaut: 1)

chkData : données de vérification

optMethod : méthode d'optimisation : '1' : méthode hybride ; '0' : méthode de rétro-probagation

### 5.5.3 Evaluation du réseau :

La commande 'evalfis' permet d'évaluer le réseau.

Sortie_reseau=evalfis (input, fis)

input : données de test.

## 5.6 Utilisation de l'outil neuro-flou

Taper la commande 'anfisedit' pour ouvrir l'outil d'ajustement du modèle anfis qui est composé de quatre parties : chargement des données, génération du SIF, apprentissage du SIF, test du SIF.

Taper les commandes ‘load  fuzex1trnData.dat’ et ‘load  fuzex1chkData.dat’pour charger les données d’apprentissage et de vérification respectivement.

A partir de l’interface graphique mentionnée ci-dessous, cliquer sur ‘Load Data’.

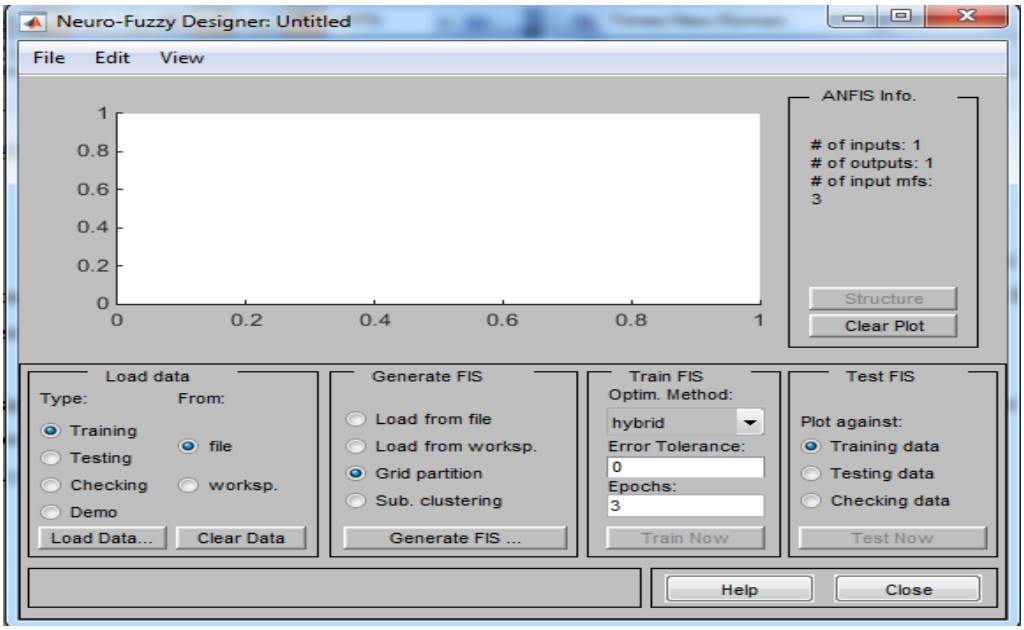



***Figure 5.2.*** Fenêtre principale d’ANFIS .

Vérifier que les options ‘Training’ et ‘Worksp’ sont sélectionnées. Taper ‘fuzex1trnData’ dans la fenêtre qui apparait puis ok pour procéder. On peut suivre les mêmes étapes pour charger les données de validation. Cliquer à nouveau ‘load data’ puis taper ‘fuzex1chkData’. Vérifier que les options ‘checking’ et ‘worksp’ sont sélectionnées.

La fenêtre suivante apparaît comme illustré ci-dessous.

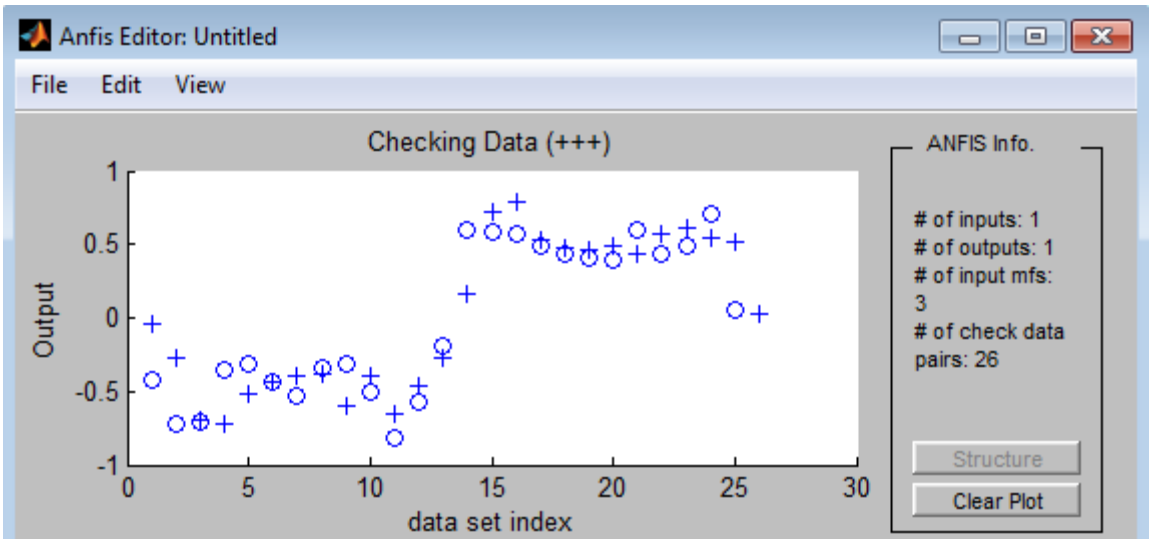



***Figure 5.3.*** Chargement des données

Cliquer sur ‘Generate Fis’ en s’assurant que l’option ‘Grid partition’ est bien choisie. Une nouvelle fenêtre apparait comme illustré ci-dessous. Prenons comme exemple ‘Number of MFs= 4’, ‘MF type = gbellmf ’ pour les entrées , et ‘MF type = linear’ pour la sortie.

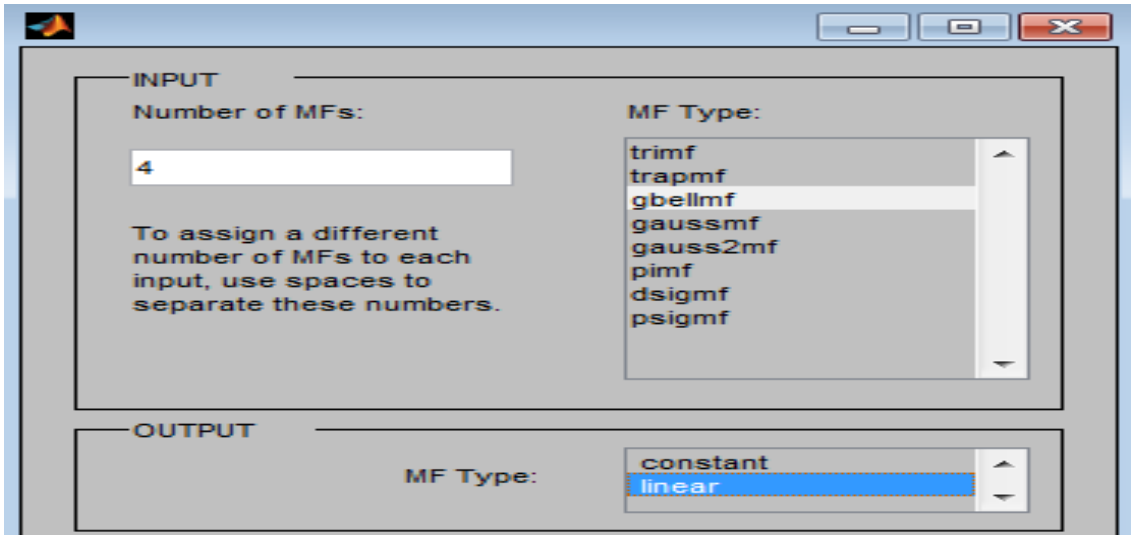



***Figure 5.4.*** Paramètres d’ANFIS

Augmenter le nombre d’epochs à 100 puis appuyer sur ‘Train now‘ pour former le réseau créé. La fenêtre mentionner ci-dessous s’ouvre en montrant l’erreur d’apprentissage (**) et de validation (♦♦).

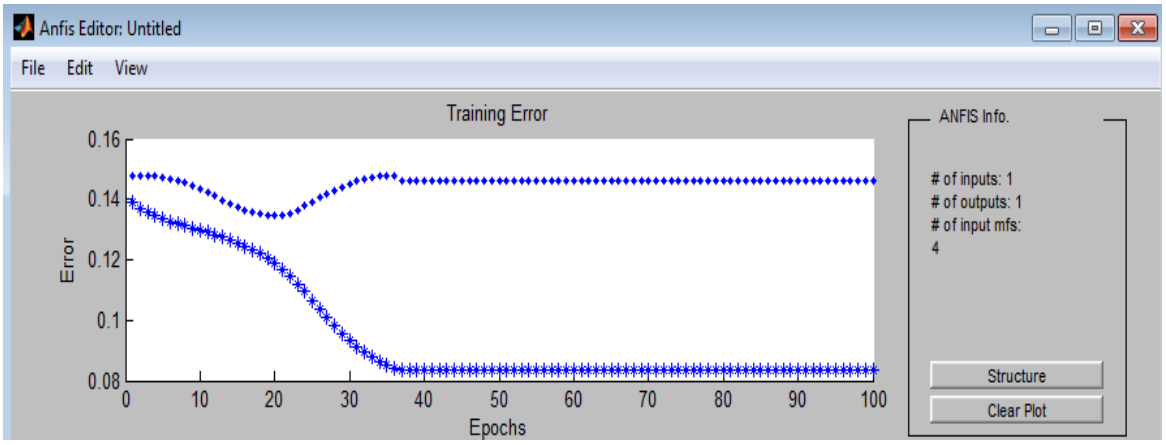



***Figure 5.5.*** Résultats finaux d’ANFIS

Vous pouvez visualiser la structure du FIS en cliquant sur le bouton structure.

Pour tester le réseau, sélectionner ‘checking Data’ dans l’onglet ‘test FIS’ puis appuyer sur le bouton ‘Test now’ .

Vous pouvez sélectionner File > export > toworkspace a fin d’exporter le système créer vers le workspace de matlab.

## 5.7 Travail demandé :

**I.** On se propose de réaliser un classifieur neuro-flou **(à l'aide des commandes matlab)** à fin de prédire un système non linéaire dans l'intervalle [-1 1], composé de la somme de trois fonctions sinusoïdales.

Soit : numPts=51; x=linspace (-1,1,numPts)';

$$y=0.6*\sin(\pi*x) + 0.3* \sin(3*\pi*x) + 0.1*\sin(5*\pi*x)$$

a. Subdiviser les données en deux parties (50% pour l'apprentissage, 50% pour la vérification).
b. Tracer les données d'apprentissage (o) et de vérification (x).
c. Générer les ensembles flous de l'entrée x et de la sortie y (utiliser cinq fonctions d'appartenance de type gbell).
d. Tracer les ensembles flous de l'entrée x (utiliser la commande plotmf).
e. Faite l'entrainement du réseau et tracer la courbe d'évolution d'erreur (utiliser lesinformations suivates : méthode d'optimisation= hybride, le seuil d'erreur =0.01 et le nombre d'epochs =200).
f. Afficher la structure du réseau neuro-flou.
g. Tester votre réseau sur les mêmes données et tracer la courbe d'évolution d'erreur.
h. Tester votre réseau sur des données de vérification et tracer la courbe d'évolution d'erreur (Apprentissage et vérification).
i. Comparer les résultats d'apprentissage et de test.
j. Améliorer la performance de votre réseau :
   Augmenter le nombre d'epochs
   Essayer d'autre fonction d'apprentissage
   Jouer sur le seuil d'erreur
   …
k. Que remarquez-vous ???

**II.** Trouver l'approximation du signal suivant à l'aide d'ANFIS.

```
X = (0:0.2:10)';
Y = sin(2*X)./exp(X/5) + randn(size(X))/30;
```

# Conclusion Générale

Les systèmes flous sont connus par leurs capacités de traitement qui se fait au niveau symbolique ainsi que la représentation de l'imprécision et l'incertitude d'un expert humain. Ceci a augmenté le nombre de domaine d'application qui utilisent la logique floue parmi lesquelles on trouve l'automatique.

Les réseaux de neurones sont composés d'éléments simples (neurones) inspirés par le système nerveux biologique. Comme dans la nature, le fonctionnement du réseau (de neurone) est fortement influencé par la connections des éléments entre eux. On peut entraîner un réseau de neurone pour une tâche spécifique (reconnaissance de caractères par exemple) en ajustant les valeurs des connections (ou poids) entre les éléments (neurone). En général, l'apprentissage des réseaux de neurones est effectué de sorte que pour une entrée particulière présentée au réseau corresponde une cible spécifique. La méthode d'apprentissage dite supervisé est souvent utilisée mais des techniques d'apprentissage non supervisé existent pour des réseaux de neurones spécifiques.

L'utilisation conjointe des systèmes d'inférence flous et les réseaux de neurones artificiels permettent d'exploiter les avantages des deux approches. Le tableau **6.1** suivant regroupe une vue comparative entre les deux méthodes :

| Les réseaux de neurones | La logique floue : |
|---|---|
| La base de règle ne peut être utilisée | La base de règle peut être utilisée |
| Basé sur l'apprentissage | Pas d'apprentissage (utilise la connaissance linguistique) |
| Boite noire | Interprétable (la règle Si…Alors) |
| Complexité des algorithmes d'apprentissage | Implémentation universel et simple |
| Difficulté pour extraire la connaissance | La connaissance doit être disponible |

**Tab 6.1** : Etude comparative entre la logique floue et les réseaux de neurones

Ainsi, on peut dire que les systèmes neuro-flous sont des modèles hybride qui fusionne les systèmes d'inférences flou et les réseaux de neurones artificiels pour exploiter la richesse des deux approches.

Pour cette raison, et durons tous ces travaux pratiques, nous avons étudier la logique floue et /ou les principaux modèles de base des réseaux connexionnistes pour le développement des systèmes intelligents, commençons par la présentation générale du principe de fonctionnement des SIFs et les utiliser pour l'optimisation des problèmes complexes, passons par l'étude de deux modèles célèbres des réseaux connexionnistes (PMC, RBF), jusqu'au la réalisation d'un classifieur neuro-flou. Tous ces travaux ont permis de faire progresser nos connaissances sur la modélisation et la manipulation des systèmes intelligents.

# Références


- Marquis Pierre, Papini Odile & Prade Henri, Panorama de l'intelligence artificielle, ses bases méthodologiques, ses développements Edition Cépaduès 2014.
- Howard Demuth, Mark Beale. Neural Network Toolbox For Use with MATLAB, User's Guide Version 3 by The MathWorks, January 1998.
- Karnopp D.C., Margolis D.L., Rosenberg R.C, System Dynamics. A unified approach, Wiley, 2nd edition, 1990;
- Shahbazova Shahnaz N, Sugeno Michio & Kacprzyk Janusz, Recent Developments in Fuzzy Logic and Fuzzy Sets, Springer International Publishing, 2020.
- Lotfi A. Zadeh, Fuzzy Logic = Computing with Words, IEEE Transactions On Fuzzy Systems, pp 103–111, Volume: 4, Issue: 2, May 1996.
- Dreyfus G, Martinez JM, Réseaux de neurones: Méthodologie et application, Collection Algorithmes, Eyrolles Edition, 2002.
- R.Jang, neuro-fuzzy modeling: architecture, analyses and applications, PhD thesis, Dep. Of electrical Engineering and computer Science, University of California, Berkeley, 1992.
- Paul Wilmott, Machine Learning: An Applied Mathematics Introduction Paperback, Panda Ohana Publishing 2019.
- Oussar Y, Dreyfus G, How to be a gray box: dynamic semi-physical modeling, Neural Networks Pages 1161-1172, Volume 14, Issue 9, November 2001.
- C. Naga Bhaskar, G. Vijay kumar : Neural Networks and Fuzzy Logic , BS Publications 2020.

# Annexes :

# Annexe I :

## Commandes Usuelles de Matlab

| | |
|---|---|
| help | donne toute la liste des aides |
| help sujet | affiche l'aide sur le sujet |
| quit, exit | Extinction de MATLAB |
| cd nom_répertoire | change de répertoire de travail |
| pwd | affiche le nom du répertoire courant |
| ls | affiche le contenu du répertoire courant |
| load nomfichier.dat | importe du fichier nomfichier.dat |
| save nomfichier | sauvegarde toutes les variables dans le fichier nomfichier.mat |
| save nomfichier X Y | sauvegarde uniquement les variables X et Y dans le fichier nomfichier.mat |

**Tab 7.1 :** Commandes de base

| | |
|---|---|
| plot(X,Y,'cs') | dessine le graphe de Y en fonction de X suivant la couleur c et en utilisant le symbole s. Exemple de valeurs que peut prendre c : y (jaune), r (rouge), g (vert), b (bleu), w (blanc)…. s peut être : . (point), - (tiret), + (plus), * (étoile), o (cercle), x (croix) |
| title('texte') | ajoute le texte comme titre de la figure |
| xlabel('texte') | ajoute le texte comme légende de l'axe des abscisses |
| ylabel('texte') | ajoute le texte comme légende de l'axe des ordonnées |
| legend('texte1', 'texte2'…) | place une légende sur la figure courante |
| hold on ou hold off | autorise ou interdit le dessin des prochaines fonctions sur la figure sans effacer les précédentes |
| close | ferme la figure courante |
| close(num) | ferme la figure portant le numéro num |
| close all | ferme toutes les figures |

**Tab 7.2 :** Fonctions Graphiques

# Annexe II:

## 1. Types des systèmes d'inférence flous :

Il existe principalement deux types de systèmes flous :

**a. Système floue de Mamdani** : Dans ce type de systèmes flous, la prémisse et la conclusion sont floues. Après la réalisation de l'inférence floue. Une étape de «défuzzification » est obligatoire pour le passage du symbolique au numérique. Un exemple de ce SIF est illustré dans la **figure 7.1**

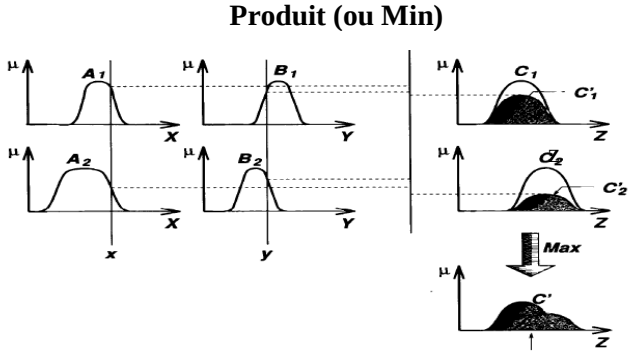



**Figure 7.1** : Exemple du SIFde Mamdani

**b. Système floue de Takagi-Sugeno** *:* Ce modèle est proposé par Takagi, Sugeno et Kang **.** il est composé d'une base de règles floues de la forme :

*Si x est Ai et y est Bi Alors z= f (x, y). avec f est une fonction nette.*

Dans ce type de SIF, la conclusion correspond a une constante ou une expression polynomiale. Il permet d'obtenir directement la sortie défuzzifiée à partir des règles linguistiques. Ainsi, un poids est associé à chaque règle, sa valeur dépend de la logique adoptée pour les opérateurs ET et OU contenus dans les prémisses des règles. La **figure 7.2** montre un exemple de SIF de Sugeno.

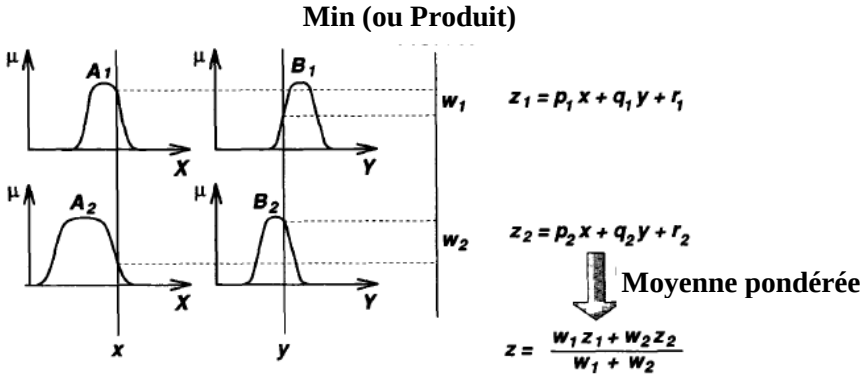



**Figure 7.2** :Exemple du Sif de Sugeno

# 2. Architecture des RNAs :

**a. Les réseaux « feed-forward » :** Les réseaux de neurones de type feed-forward avec rétro propagation constituent l'un des modèles les plus populaires. Leur procédure d'apprentissage supervisé est basé sur une idée simple: si le réseau donne une mauvaise réponse, les poids sur les connexions sont ajustées afin de réduire cette erreur et d'augmenter la probabilité d'une bonne réponse à la prochaine étape**.** Un exemple de réseau est illustré dans la **figure 7.3** suivante :

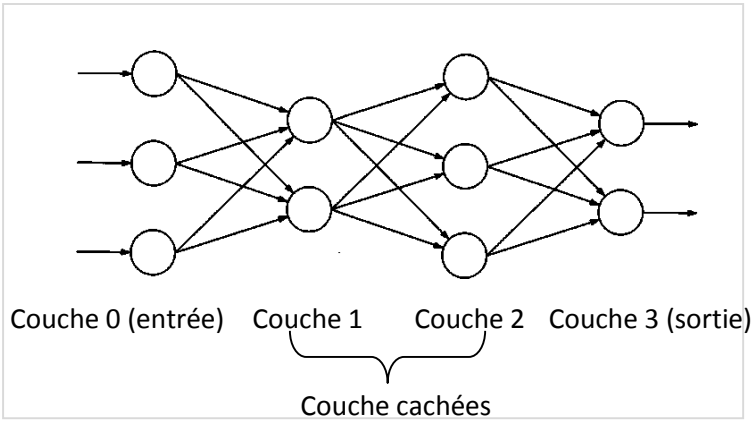



**Figure 7.3** : Réseau : feed-forward

Le modèle est composé de trois couches : la couche d'entrée, la couche cachée, et la couche de sortie. Chaque couche est bien connectée à la couche suivante par des connexions pondérées qui propagent le signal dans une direction en avant de l'entrée à la sortie. Aucun traitement aura lieu dans la couche d'entrée i.e le vecteur d'entrée fournie à la couche d'entrée est directement propagée à la couche cachée à travers des poids de connexion.

**b. Les réseaux «Feed-Back» :** Ces réseaux sont aussi appelés réseaux récurrents. Il s'agit de réseaux de neurones avec retour en arrière. Ce type de réseau est caractérisé par la Présence d'au moins une boucle de rétroaction au niveau des neurones ou entre les couches comme le montre la **figure 7.4** suivante :

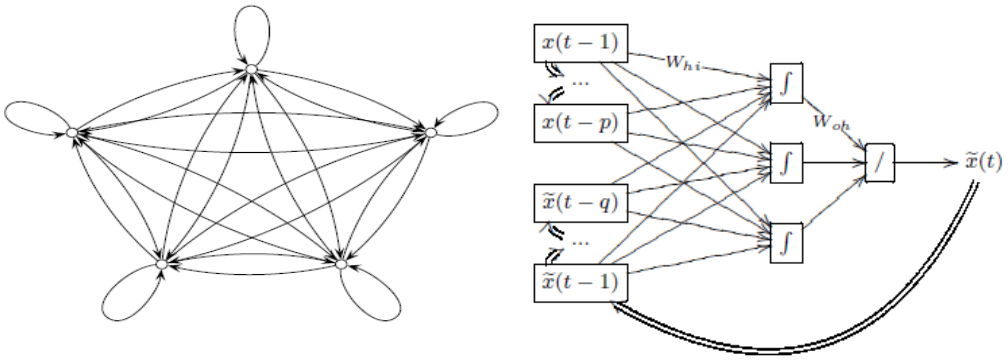



**Figure 7.4** : Réseau de neurone récurrent

# 3. Apprentissage

L'apprentissage est la caractéristique principale du réseau de neurone. Il peut être considéré comme un problème de mise a jour des poids de connexions au sein du réseau .Deux types d'apprentissage peut être envisagés :

**a. Apprentissage supervisé :** est caractérisé par la présence d'un «professeur» qui possède une connaissance approfondie de l'environnement. En pratique, les connaissances de ce professeur prennent la forme d'un ensemble de Q couples de vecteurs d'entrée et de sortie que nous noterons $\{(p_1, d_1), (p_2, d_2), \ldots, (p_Q, d_Q)\}$, ou $p_i$ désigne un stimulus (entrée) et $d_i$ la cible pour ce stimulus, c'est-`a-dire les sorties désirées du réseau. Chaque couple (pi, di) correspond donc a un cas de ce que le réseau devrait produire (la cible) pour un stimulus donné. Pour cette raison, l'apprentissage supervisé est aussi qualifié d'apprentissage par des exemples.

**b. Apprentissage non supervisé** : La deuxième forme d'apprentissage est dite « non-supervisée» ou encore «auto-organisée». Elle est caractérisée par l'absence complète de professeur. Nous ne disposons donc que d'un environnement qui fournit des stimulis, et d'un réseau qui doit apprendre sans intervention externe. L'apprentissage non-supervisé s'appuie généralement sur un processus compétitif.

# 4. Autres Modèles Neuro-Flous

## a. NEFCON (Neuro-Fuzzy Control)

NEFCON vise à mettre en œuvre le SIF de type Mamdani. Il est composé de trois couches : une couche d'entrée incluant les nœuds d'entrées et les sous ensemble flous d'antécédent, une couche cachée formé par des règles et un neurone de sortie pour les sous-ensemble flous des conséquences

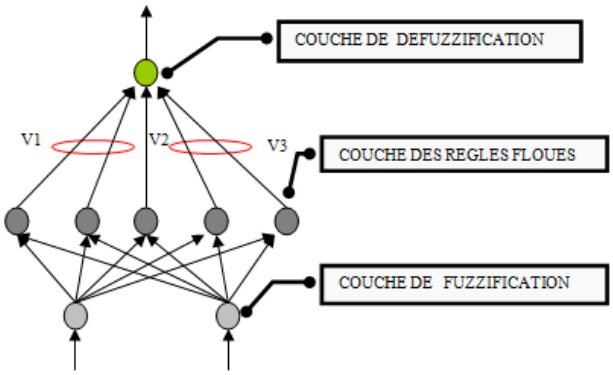



**Figure 7.5 :** Architecture de NEFCON

## b. FALCON (Fuzzy Adaptative Control Network)

Comme le montre la **figure 7.6** au-dessous, Falcon est un réseau comportant cinq couches. Il possède deux nœuds linguistiques, une pour la sortie désiré et l'autre pour la sortie du Falcon.

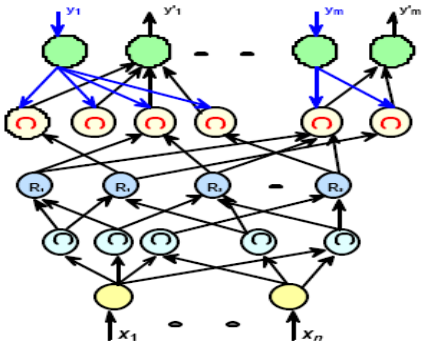


**Figure 7.6 :** Architecture de FALCON

FALCON utilise un algorithme d’apprentissage hybride :

- Apprentissage non supervisée : pour localiser les fonctions d’appartenance et la base des règles initiale
- Apprentissage supervisé : pour optimiser l’ajustement des paramètres des fonctions d’appartenances.

## c. FUN (Fuzzy Set)

Pour FUN , les neurones dans la première couche cachée contient les fonctions d'appartenance pour exercer une fuzzification des valeurs d'entrée. Dans la deuxième couche cachée, les conjonctions (fuzzy-AND) sont calculées. Les fonctions d’appartenances des variables de sortie sont stockées dans la troisième couche cachée. Leur fonction d'activation est un (fuzzy-OR). A la fin, le neurone de sortie effectue la défuzzification. Le réseau est initialisé avec une base de règles floues et les fonctions d'appartenance correspondant .il utilise une technique d'apprentissage qui change les paramètres des fonctions d’appartenance

# Annexe III:

## 1. Extrait du TP N1

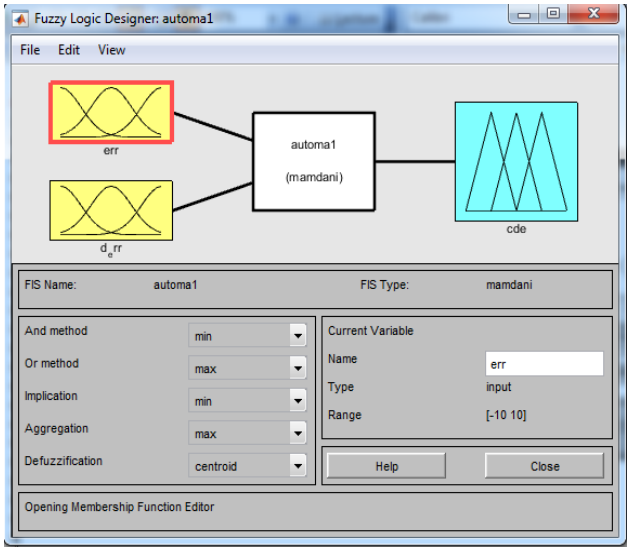



**Fig 7.7.** Fenêtre principale du SIF

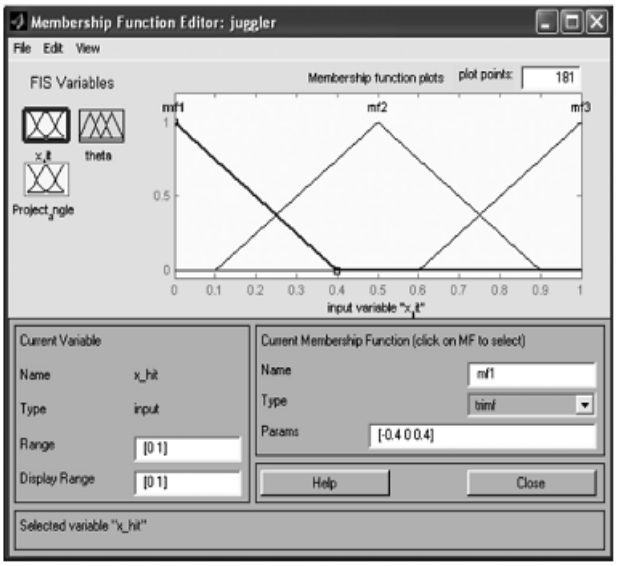



**Fig 7.8.** Editeur des fonctions d'appartenance

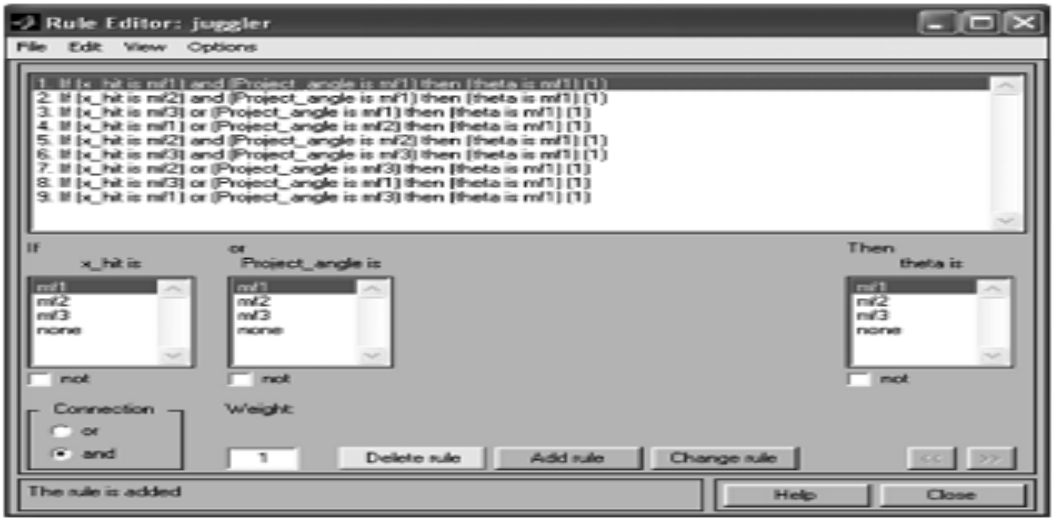



**Fig 7.9.** Editeur des règles

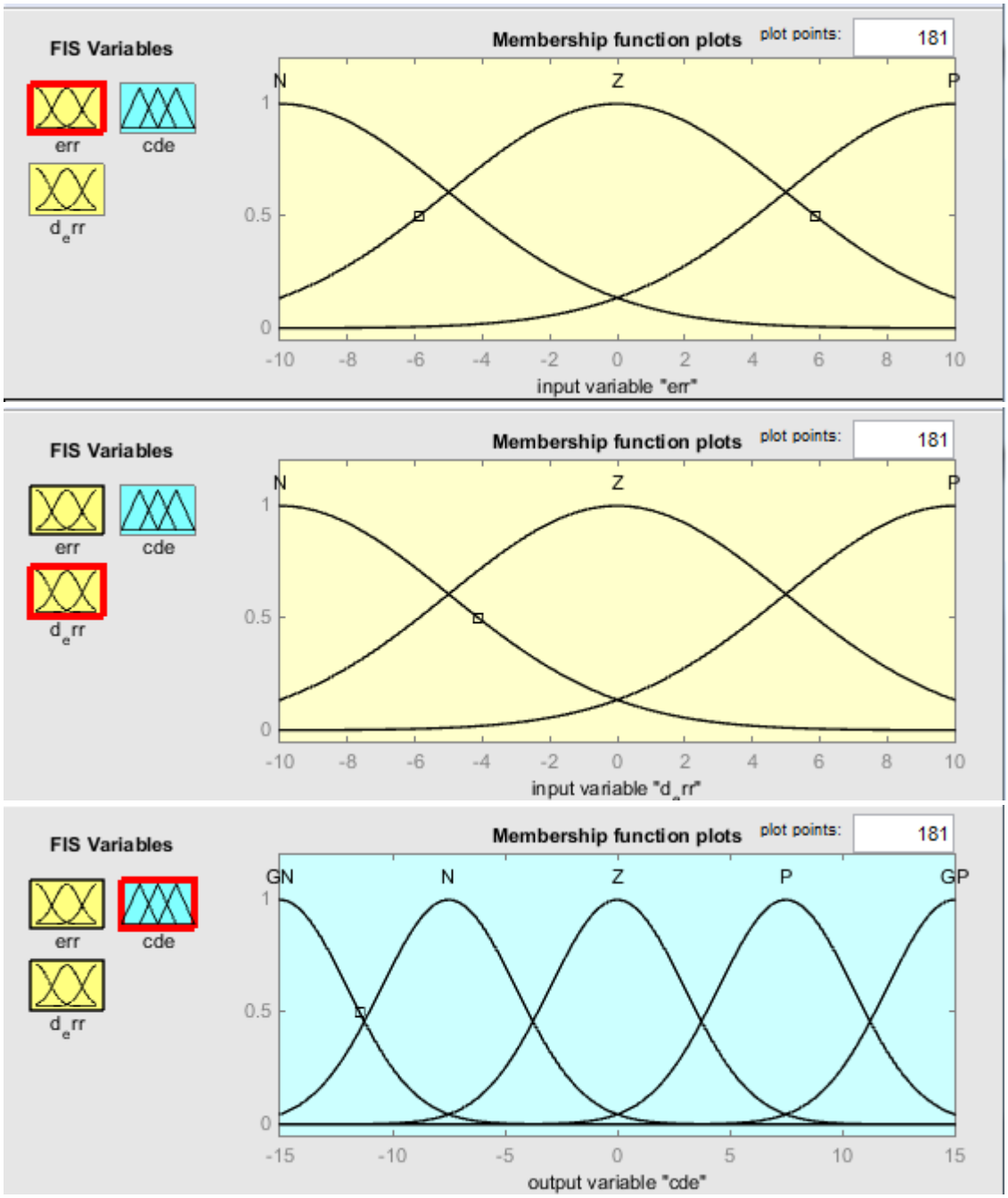



**Fig 7.10.** Fonctions d'appartenance

## 2. Extrait tp2 :

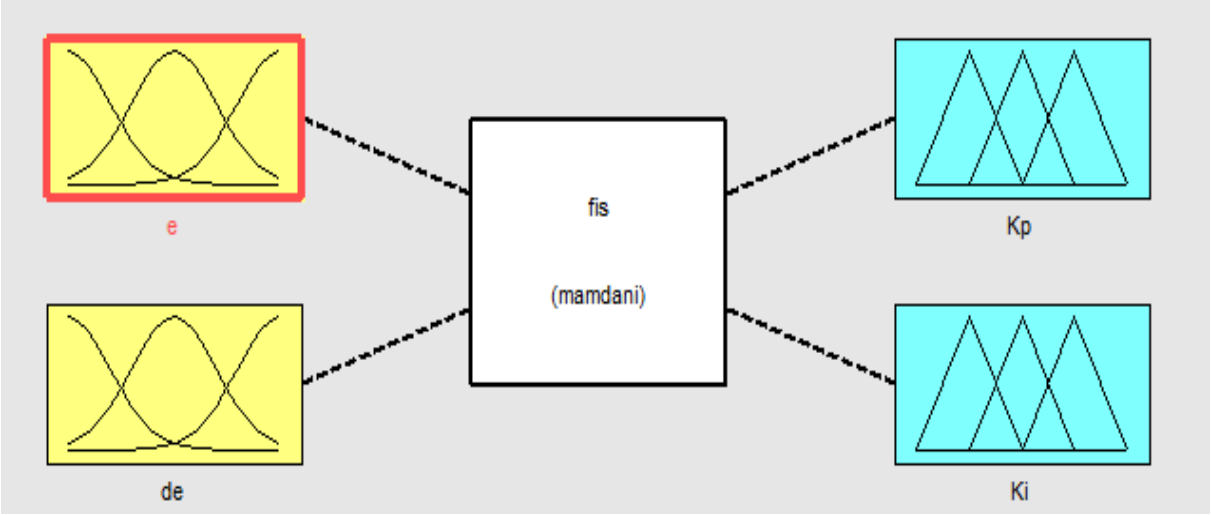



**Fig 7.11** architecture du système d'inférence flou implémenté

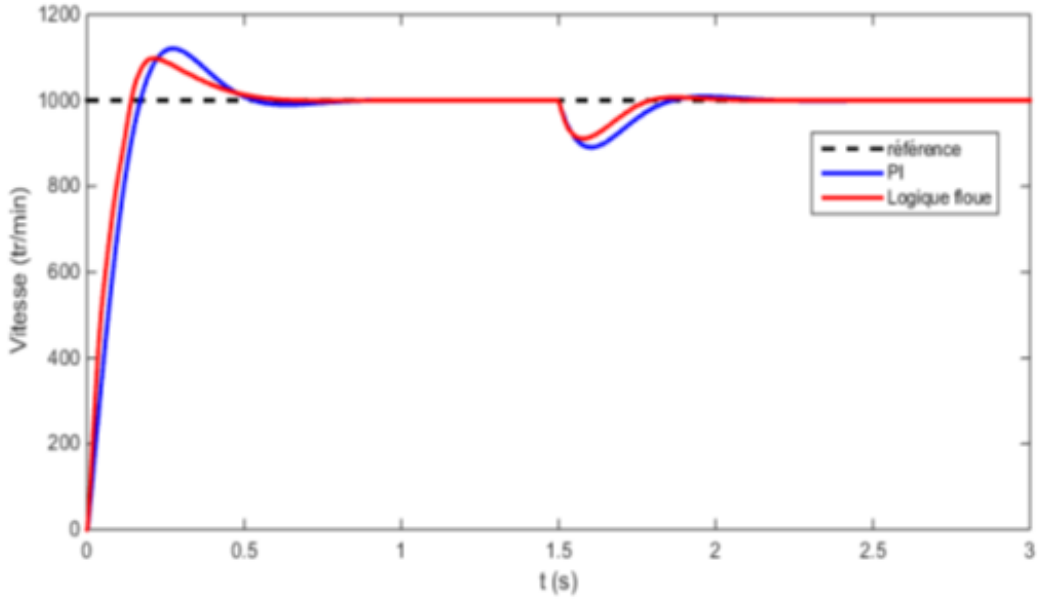



**Fig 7.12 C**omparaison entre la commande PI et logique floue

### 3. Extrait Tp3 :

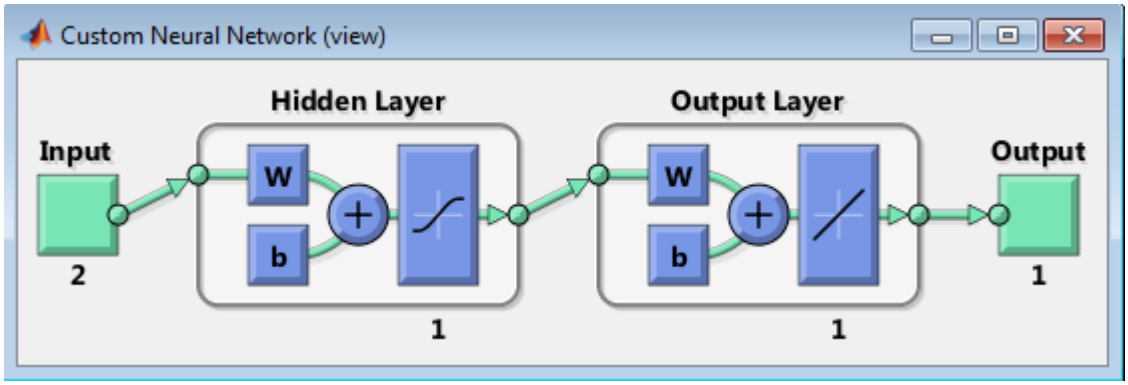



**Fig 7.13.** Architecture du réseau 1

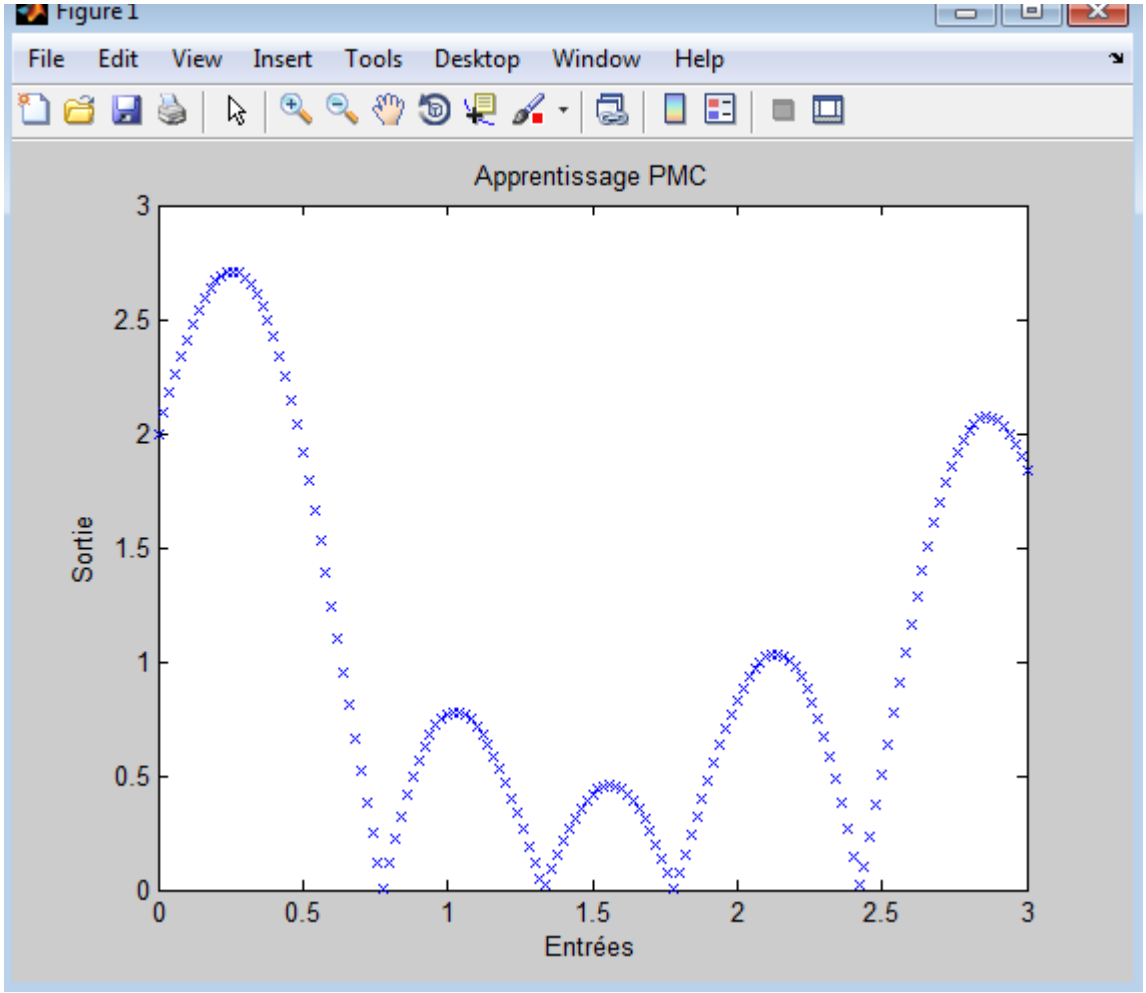



**Fig 7.14.** Courbe du signal (P=0 :0.02 :3 ; T=0.5*abs(sin(5*P)+ exp(-.5*P)+cos(2*P)))

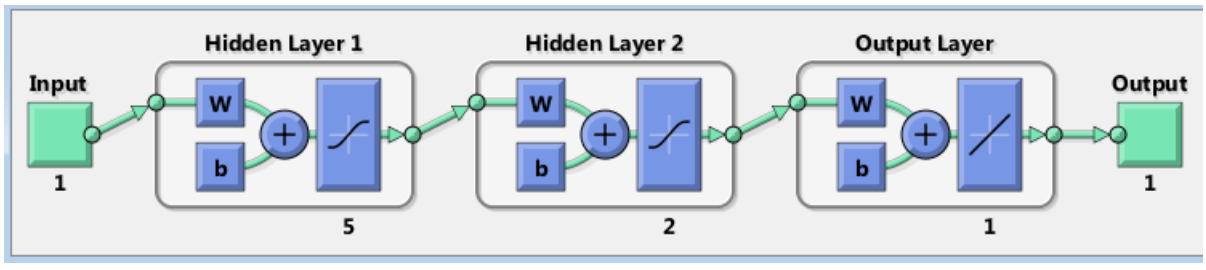



**Fig 7.15.** Architecture du réseau 2

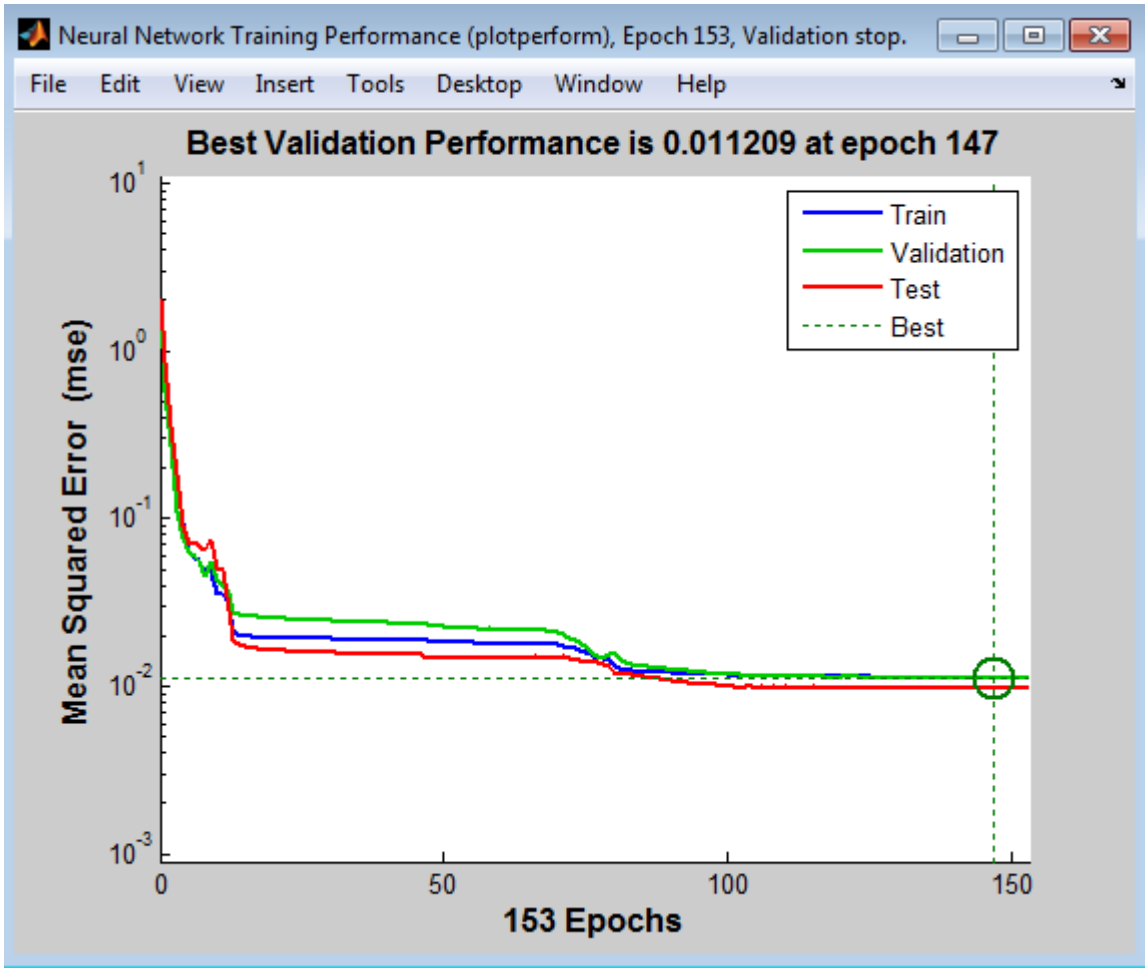



**Fig 7.16.** Evolution d'erreur

## 4. Extrait du TP N 4:

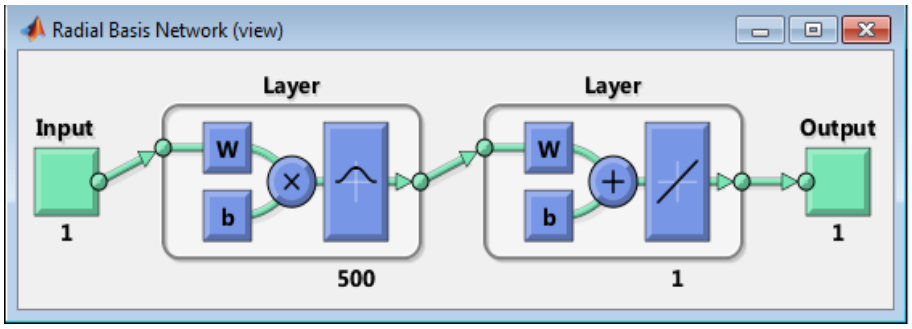



**Fig 7.17.** Architecture du réseau RBF implémenté

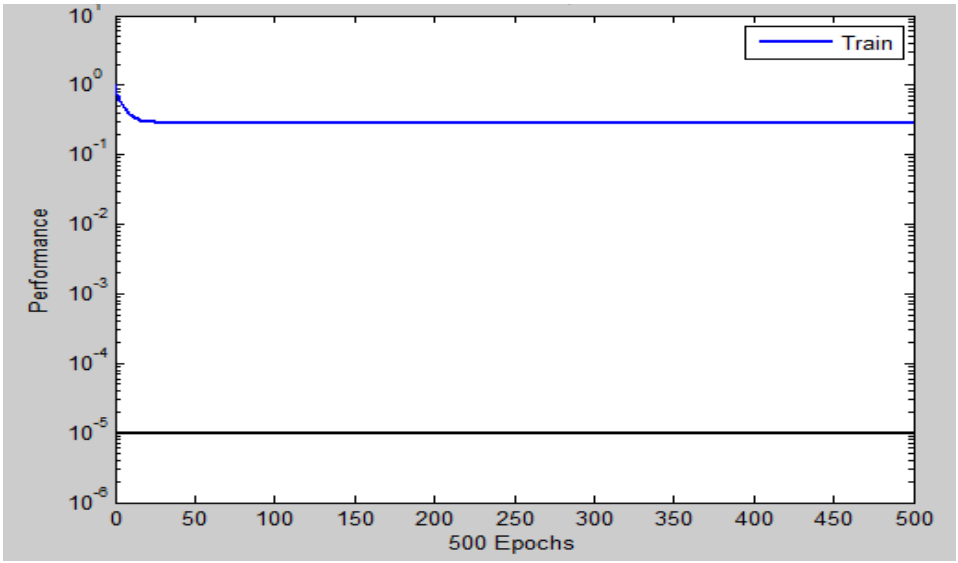



**Fig 7.18.** Evolution d'erreur d'apprentissage

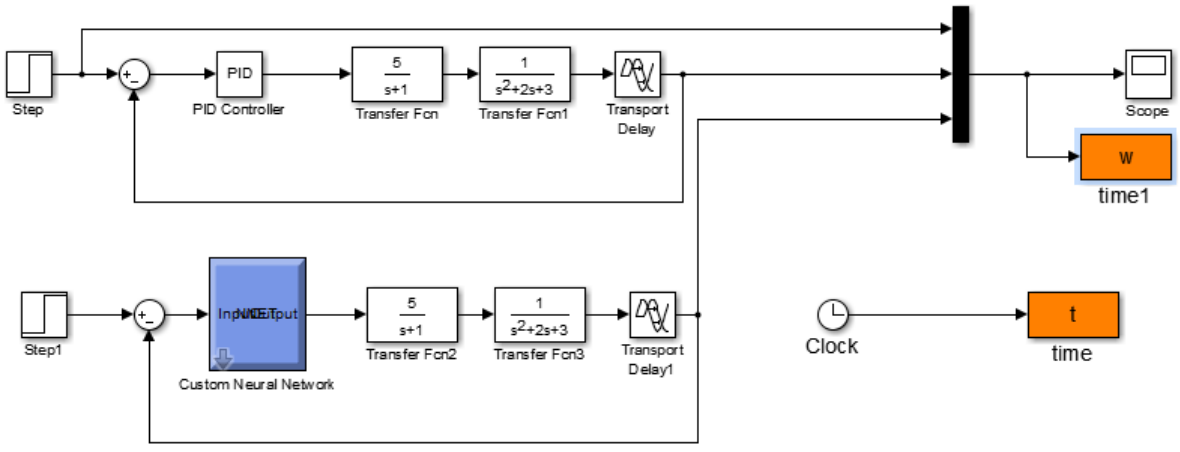



**Fig 7.19.** Implémentation sur Simulink d'un modèle de contrôle d'un système de fonction de transfert

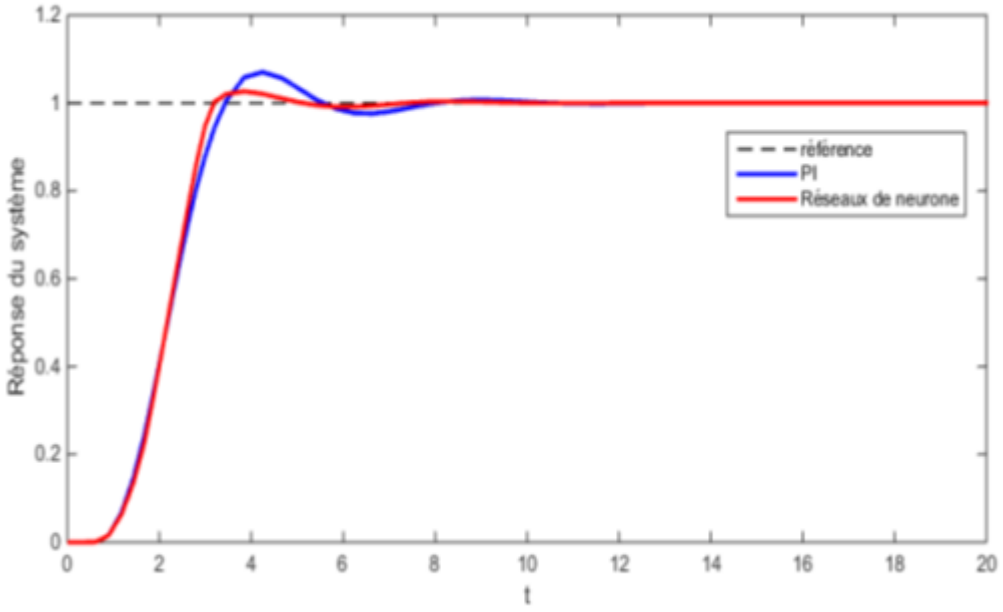



**Fig 7.20 C**omparaison entre la commande PID et réseaux de neurone

**5. Extrait tp5 :**

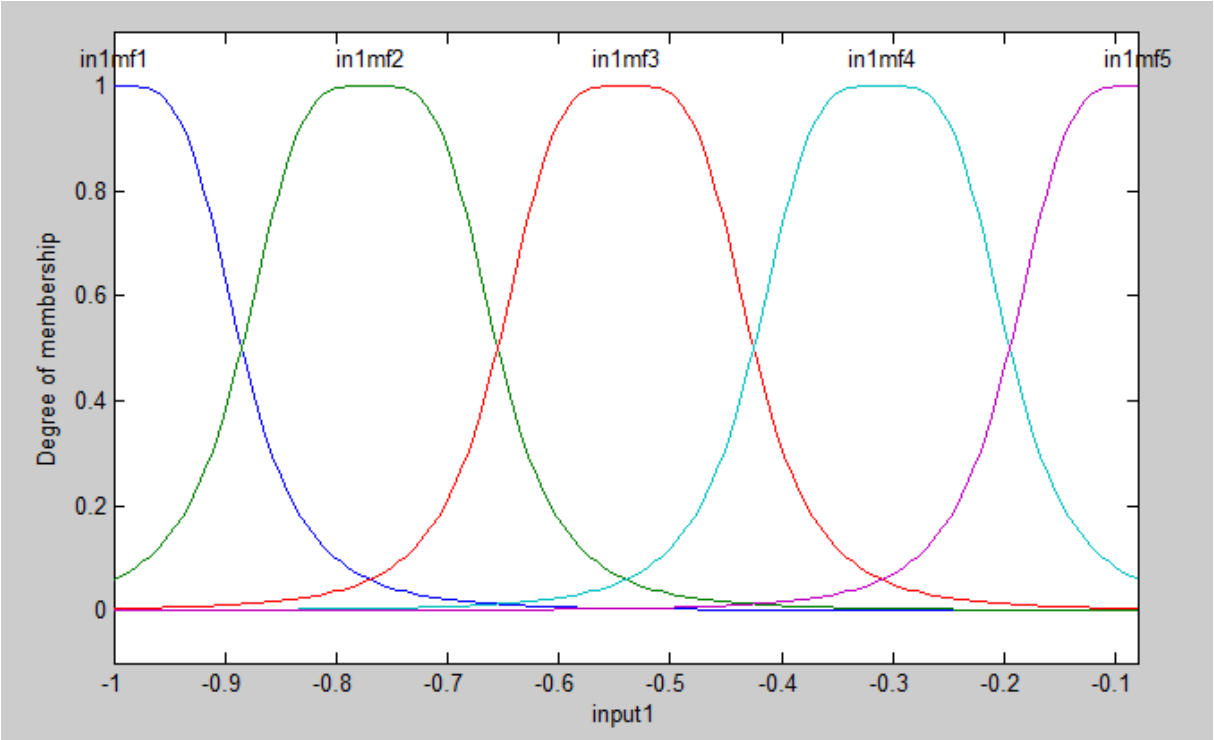



**Fig 7.21** Ensembles flous de l'entrée x

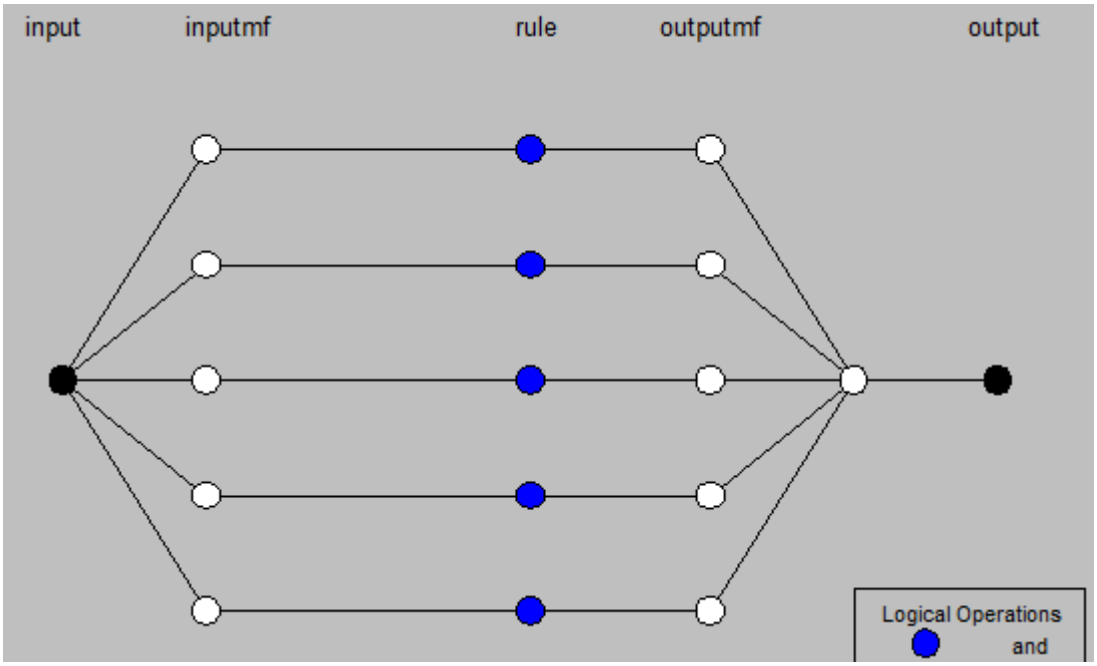



**Fig 7.22** Architecture du système neuro-flou implémenté

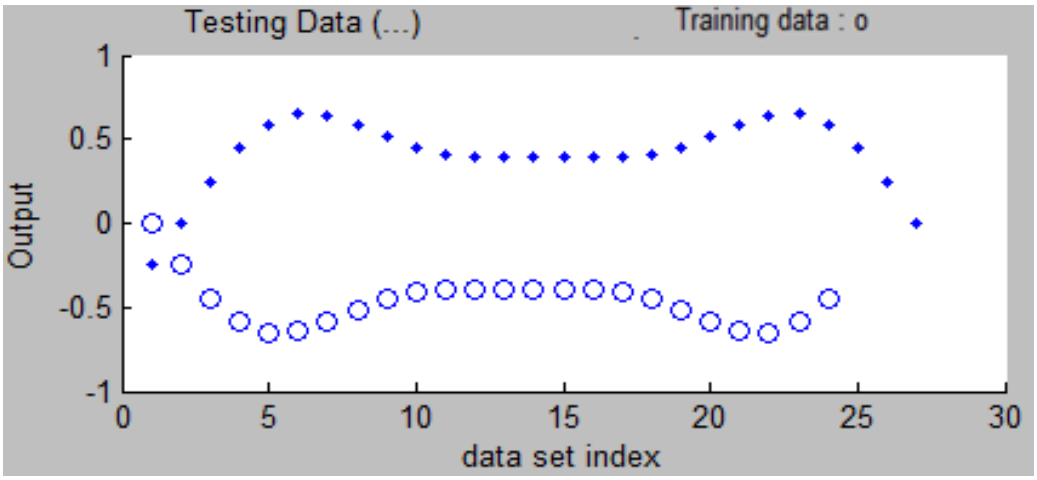



**Fig 7.23** Données d'apprentissage (o) et de vérification (♦)

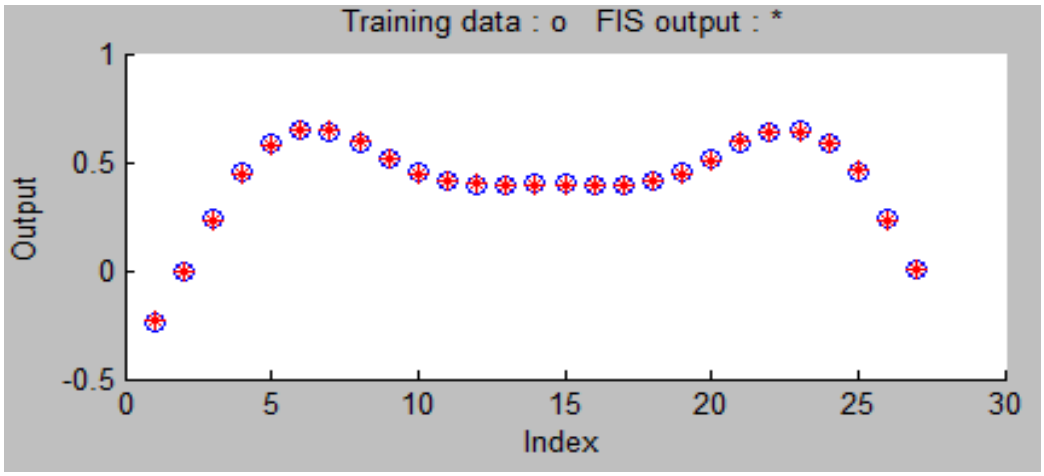



**Fig 7.24** Résultats d'apprentissage

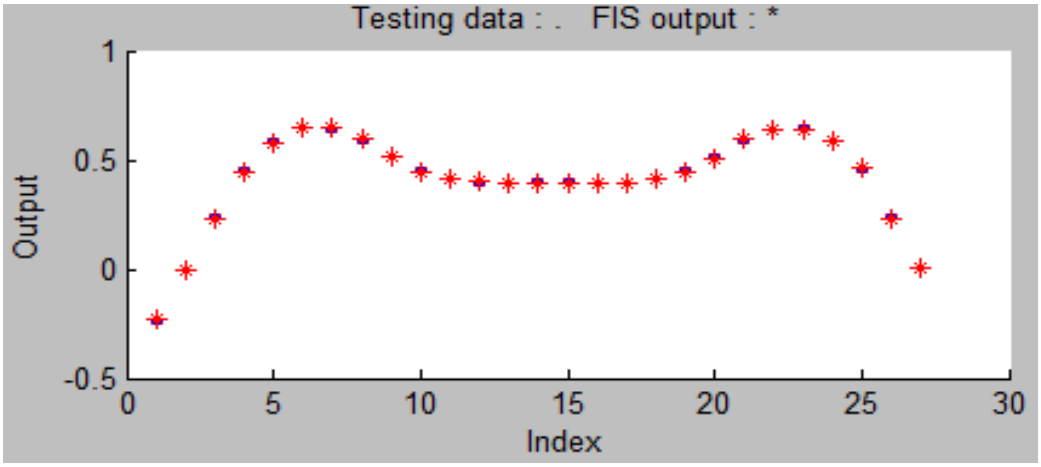



**Fig 7.25** Résultats de test