الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – Biskra
Faculté des Sciences et de la technologie
Département : **Génie Electrique**
Réf :……

جامعة محمد خيضر بسكرة
كلية العلوم والتكنولوجيا
قسم:الهندسة الكهربائية
المرجع :……

Mémoire présenté en vue de l'obtention
du diplôme de

Magister en : Automatique

Option : Commande et Identification des Systèmes Dynamiques

Commande Prédictive Non Linéaire en Utilisant Les Systèmes Neuro-Flous et les Algorithmes Génétiques

Présenté par :

Abdallah Bezzini

Soutenu publiquement le 06/06/2013

Devant le jury composé de :

Mr. Souri Mohamed Mimoune	**Professeur**	**Président**	**Université de Biskra**
Mr. Mohamed Boumehraz	**Maître de conférences**	**Rapporteur**	**Université de Biskra**
Mr. Abdelkrim Allag	**Professeur**	**Examinateur**	**Université de Biskra**
Mr. Achour Betka	**Professeur**	**Examinateur**	**Université de Biskra**

Remerciements

Au terme de ce travail, il m'est très agréable d'exprimer toutes mes reconnaissances à Mr Mohamed BOUMEHRAZ, mon encadreur pour ses précieux conseils, ses importantes directives ainsi que pour sa patience tout au long de l'élaboration de ce travail.

Mes profondes gratitudes et mes remerciements s'adressent également aux membres de jury:

-Souri Mohamed Mimoune.

- Abdelkarim Allag.

-Achour Betka.

Enfin, nous remercions toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Abdallah BEZZINI

Dédicace

Je dédie ce mémoire :

À ma mère et mon père qui n'ont pas cessé de m'encourager tout au long de mes études pour que j'arrive à ce jour-là. Ils sont les plus chers à mon cœur, avec toute ma gratitude,

À mes chères sœurs et frères,

À tous mes amis,

À toute ma famille.

Abdallah BEZZINI

Table des matières

Table des Figures

Résumé :

Grâce à ses termes de performances et de simplicité d'implémentation la commande prédictive, a connu un succès considérable dans le milieu industriel, malheureusement elle est parfaitement adaptée aux systèmes linéaires, et souvent il n'existe pas de modèle parfaitement linéaire. S'appuyant sur les avancées de la théorie de la modélisation des systèmes non linéaires, le choix du système est orienté vers l'utilisation des modèles Neuro-flous qui ont montré ses efficacités dans plusieurs applications. La loi de commande est obtenue, en général, par l'optimisation d'un critère qui pénalise l'effort de commande et l'état du système. On applique au système à commander la première valeur de la séquence de commande obtenue par la solution du problème, et le même processus de calculs est répété pour l'instant suivant à partir de la nouvelle mesure. Le problème majeur qui se pose lorsqu'on utilise des modèles non linéaires est que le problème d'optimisation à résoudre en ligne est généralement non convexe et sous contraintes dont le temps de calcul peut être prohibitif et la convergence vers un minimum global n'est pas assurée, malheureusement les méthodes numériques classiques utilisées nécessitent un temps de calcul très important sans garantie de convergence vers un minimum global. Les algorithmes génétiques qui sont des méthodes de résolution puissantes, peuvent apporter une solution efficace à ce problème.

Dans ce travail, on va étudier l'utilisation des modèles Neuro-flous pour la construction du prédicteur du système à commander, en utilisant les algorithmes génétiques pour la phase d'optimisation dans une structure de commande prédictive des systèmes non linéaires.

Mots-clés : Modèles Neuro-flous, système non linéaire, commande prédictive, optimisation, algorithme génétique.

Abstract:

With its performance and ease of implementation, the predictive control, has a big success in industry, unfortunately it is well suited for linear systems, and often there are no perfectly linear model. With refer to advances in modeling theory of nonlinear systems; the choice of system is oriented to the use of fuzzy-neural models that have shown its efficiencies in many applications. In general, The NMPC controller minimizes a loss function at every sampling instant. A number of future control moves (the control horizon) is calculated each time, and the first control move of this control horizon is implemented. The calculations are then repeated at the next sampling instant from the new measure of output. The major problem that arises when using nonlinear models is that the optimization problem to be solved online is generally no convex, and the computation time can be very big and convergence to a global minimum is not sure, unfortunately the classical numerical methods used requires a considerable computation time with no guarantee of convergence to a global minimum. Genetic algorithms are powerful methods of resolution, can provide an effective solution to this problem.

In this work, we will study the use of models for the construction of fuzzy-neural predictor of the controlled system, using genetic algorithms for the optimization phase, all in a structure of a predictive control of nonlinear systems.

Keywords: fuzzy-neural models, non-linear system, predictive control, optimization, genetic algorithm.

Résumé

ملخص:

نظرا لأدائها الناجح وسهولة تنفيذها حققت المتحكمة التنبؤية، نجاحا كبيرا في الصناعة، لسوء الحظ أنها مناسبة تماما للأنظمة الخطية، وغالبا لا يوجد نموذج خطي تماما.و بالاعتماد على التقدم في نظرية النمذجة للأنظمة غير الخطية، فان الاختيار موجه نحو استخدام الأنظمة العصبية الضبابية التي أظهرت كفاءة في العديد من التطبيقات.يتم الحصول على القانون التحكم، بشكل عام، عن طريق تحقيق أمثلية المعيار الذي هو عبارة عن مجموع الفارق بين المخرجة المراد الحصول عليها مع المخرجة الحقيقية بالإضافة إلى الجهد المبذول للتحكم في النظام. نطبق القيمة الأولى فقط الناتجة من خلال حل المشكل، ويتم تكرار نفس العملية خلال اللحظة المقبلة من الإجراء الجديد. المشكلة الكبرى التي تنشأ عند استخدام النماذج غير الخطية هو أن المشكلة التي يتعين حلها تكون متعددة الذروات، و قد يكون الوقت اللازم للحساب كبيرا والتقارب إلى الحد الأدنى غير مضمون، للأسف فإن الطرق الرقمية التقليدية المستخدمة تتطلب وقتا حساب كبير مع عدم وجود ضمان على تقاربها إلى الحد الأدنى. الخوارزميات الجينية هي وطرق قوية لحل هذا النوع من المسائل، وهو ما يمكنها من توفير حل فعال لهذه المشكلة.

في هذه المذكرة، وسوف نقوم بدراسة استخدام نماذج عصبية ضبابية لتوقع سلوك النظام الذي نريد التحكم فيه ، و باستخدام الخوارزميات الجينية لتحقيق أمثلية المشكلة و ذلك كله ضمن المتحكمة التنبؤية للأنظمة غير الخطية.

الكلمات المفتاحية: الأنظمة العصبية الضبابية،الأنظمة غير الخطية، المتحكمة التنبؤية،الأمثلة، الخوارزميات الجينية.

Introduction générale

Introduction générale :

La commande prédictive à base de modèle (MPC, sigle Anglais correspondant à Model Prédictive Control) connue aussi sous l'appellation de commande à horizon fuyant ou glissant (Receding Horizon Control or Moving Horizon Control) est apparu au début de la décennie 60, connue plus simplement comme commande prédictive, se situe parmi les commandes avancées les plus utilisées dans le milieu industriel ces dernières décennies [15], grâce à ses performances , sa facilité de mise en œuvre et sa capacité d'inclure de manière explicite des contraintes imposées dans l'étape de calcul de la loi commande.

Les premières formulations concernant cette stratégie étaient basées sur des modèles linéaires. Mais, dans la réalité la plupart des processus sont complexes et de comportement généralement non stationnaire et non linéaire où peu de connaissance sur le comportement est disponible, une étape de modélisation de leur comportement peut être difficile et parfois impossible. De plus, même si le système est linéaire ou un modèle linéarisé si possible, la dynamique de la boucle fermée est non linéaire de fait de la présence des contraintes. Dans ce cas, l'emploi d'un modèle non linéaire est nécessaire pour balayer cette faiblesse. Cette nécessité d'introduire des modèles non linéaires dans la formulation de la commande prédictive est aujourd'hui bien connue par: La commande prédictive non-linéaire ou encore, la commande non-linéaire à horizon fuyant (nonlinear receding horizon control).

Le succès de la commande prédictive dépend du degré de précision du modèle du système à contrôler. Dans l'étape de conception, il est impératif de bien prédire le comportement dynamique, pour cela il faut avoir une connaissance précise des constituants du processus. Donc, la commande prédictive non linéaire, nécessite la disponibilité d'un modèle fiable, fidèle et précis décrivant le système à commander et reflétant ses non-linéarités et ses complexités dynamiques. Parmi les choix possibles pour la réalisation d'une structure de modèle non linéaire, les modèles neuroflous présentent une solution prometteuse grâce à leurs capacités prouvées, théoriquement et pratiquement, à l'approximation des systèmes non linéaires. S'appuyant sur les avancées de la théorie de la modélisation, il est possible d'inclure un système neuro-flou

en MPC de manière à avoir des performances satisfaisantes quand le modèle du processus est non linéaire.

Le problème majeur qui se pose lorsqu'on utilise des prédicteurs non linéaires de type Neuroflou dans cette nouvelle construction est que le problème d'optimisation à résoudre en ligne est non linéaire et non convexe. Généralement on utilise des méthodes itératives comme la programmation séquentielle quadratique et les méthodes de type Newton-Raphson pour la résolution de ce genre de problème. Malheureusement ces méthodes itératives nécessitent un temps de calcul excessif et la convergence vers un minimum global n'est pas assurée. Contrairement aux ces méthodes, les algorithmes génétiques sont des méthodes stochastiques caractérisées par une grande robustesse et possèdent la capacité d'éviter les minimums locaux pour effectuer une recherche globale. De plus, ces algorithmes n'obéissent pas aux hypothèses de dérivabilité qui contraignent pas mal de méthodes classiques destinées à traiter des problèmes d'optimisation non linéaire. Ils reposent sur un codage de variables organisées sous forme de structures chromosomiques et ils explorent l'espace de recherche en basant sur principes de l'évolution naturelle de Darwin pour déterminer une solution optimale, les algorithmes génétiques peuvent apporter une solution à l'optimisation du critère de la commande prédictive non linéaire plus adaptée que les autres méthodes.

L'objectif de ce travail est d'étudier une procédure qui permet d'utiliser des modèles Neuroflous pour la modélisation du système à commander et une approche à base d'algorithmes génétiques pour la résolution du problème d'optimisation sous contraintes dans une structure de commande prédictive des systèmes non linéaires.

Chapitre I:

Commande prédictive

Introduction

Si la plupart des procédés ont un comportement dynamique non linéaire, beaucoup de lois de commande prédictive appliquées sont de type linéaire et les équations du modèle sont linéarisées autour d'un point de fonctionnement. Actuellement les objectifs de commande étant plus exigeants, un modèle tenant compte de la non-linéarité valable dans une large plage de fonctionnement devient nécessaire. La résolution de ce problème qui est formulé en un problème d'optimisation non linéaire sous contraintes est actuellement possible grâce à la puissance des calculateurs. Cela doit être réalisé en un temps de calcul inférieur au temps de la période d'échantillonnage pour pouvoir appliquer cette commande en temps réel.

Ce chapitre a pour objectif de proposer un bref historique de cette approche, de résumer les principes de l'ensemble des techniques de commande prédictive, et enfin de définir le cadre dans lequel a été proposée la structure de commande prédictive non-linéaire.

I.1 Bref historique

Au début des années 1960, Propov [1] fut l'un des premiers à proposer explicitement une forme de commande prédictive basée sur un modèle en utilisant une méthode de programmation linéaire. L'idée est d'insérer, dans l'algorithme de commande, un élément de prédiction concernant l'évolution des sorties du procédé, donnée par un modèle. Le calculateur détermine alors, à l'instant d'échantillonnage présent, la séquence de commandes à appliquer sur un horizon de prédiction , à la prochaine période d'échantillonnage, seule la première composante de cette séquence est effectivement appliquée au procédé et la résolution recommence de la même façon en prenant en compte les nouvelles mesures du procédé et ainsi de suite.

La première génération de commande prédictive appliquée en milieu industriel a été initiée par Richalet [10] sous le nom Identiffcation et Commande (IDCOM) et par les ingénieurs de Shell sous le nom Dynamic Matrix Control (DMC), une liste dans l'annexe A, propose un aperçu des variantes de MPC les plus « classiques ». Dans ces approches, le modèle est de type boîte noire, l'objectif est de poursuivre une référence mais les contraintes ne sont pas encore prises en compte. Ces algorithmes possèdent un impact énorme sur la commande des procédés industriels et permettent de définir un exemple de commande prédictive basée sur un modèle.

La deuxième génération qui apparaît au début des années 1980 permet en plus la prise en compte de contraintes sur les entrées et les sorties en posant un problème quadratique (Quadratic Dynamic Matrix Control).

Enfin, la génération actuelle (SMOC , IDCOM-M, PCT, RPMC, [15]) permet de distinguer divers degrés de contraintes, permet de prendre en compte certains problèmes d'infaisabilité, utilise l'estimation d'état et permet de résoudre pour des systèmes stables ou instables en boucle ouverte divers objectifs de commande.

I.2 Principe de base

Le principe de la commande prédictive consiste à créer pour le système à commander un effet anticipatif par rapport à une trajectoire à suivre connue à l'avance,

en se basant sur la prédiction du comportement futur du système et en minimisant l'écart de ces prédictions par rapport à la trajectoire au sens d'une certaine fonction coût, tout en respectant des contraintes de fonctionnement. Cette idée est simple et pratiquée de façon assez systématique dans la vie quotidienne. Par exemple, le conducteur d'une véhicule connaît la trajectoire de référence désirée à l'avance (la route) sur un horizon de commande fini (son champ visuel), et en prenant en compte les caractéristiques de la voiture (modèle mental du comportement du véhicule), il décide quelles actions (accélérer, freiner ou tourner le volant) il faut réaliser afin de suivre la trajectoire désirée. Seule la première action de conduite est exécutée à chaque instant, et la procédure est répétée à nouveau pour les prochaines actions.

La commande MPC (Model Prédictive Control) présente un certain nombre d'avantages par rapport aux autres méthodes, parmi lesquels on trouve :

- elle peut être utilisée pour commander une grande variété de processus, ceux qui sont avec des dynamiques simples à ceux plus complexes, par exemple, les systèmes à retard, ou instable.
- Le réglage de ses paramètres relativement facile la rend accessible aux personnes avec des connaissances limitées en automatique.
- Le cas multivariable se traite facilement.
- Son caractère prédictif permet de compenser les retards et les temps morts.
- Le traitement de contraintes imposées sur le système à commander peut être inclus dans l'obtention de la loi de commande.
- Elle est très performante lorsque les consignes ou trajectoires à suivre sont connues à l'avance (ce qui est le cas dans plusieurs processus industriels comme les machines numériques et les robots).

La détermination de la loi de commande prédictive se fait par résolution, d'un problème de commande optimale à horizon fini comme il est illustré dans la figure suivante. À partir d'une trajectoire de référence à suivre connue à l'avance, en faisant à chaque période d'échantillonnage les étapes suivantes :

1. Calculer les prédictions des variables de sortie $\hat{y}$ sur un horizon de prédiction sur la sortie N_p.
2. Minimiser un critère à horizon fini en fonction de : erreurs de prédictions futures, écarts entre la sortie prédite du système et la consigne future.
3. Obtenir une séquence de commandes futures sur un horizon de commande inférieure ou égale à N_p.
4. Appliquer uniquement la première valeur de cette séquence sur le système.
5. Répéter ces étapes à la période d'échantillonnage suivante, selon le principe de l'horizon fuyant.

L'ensemble est résumé sur la figure suivante :

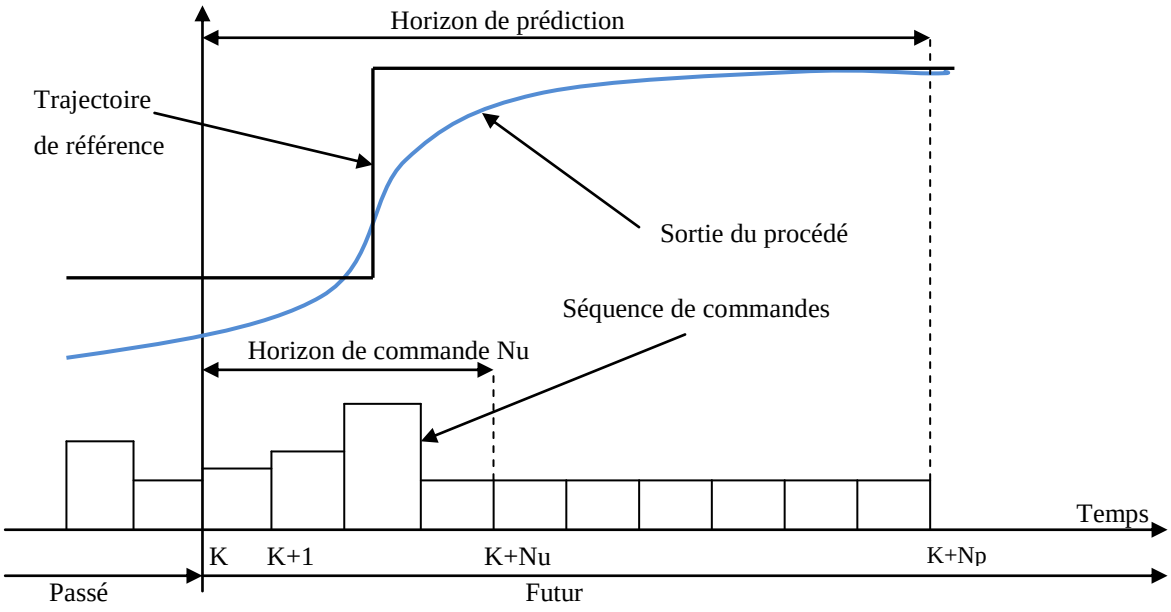


Figure I.1 : Stratégie de la commande prédictive.

Remarque :

Dans le cas général d'un système sous contraintes, la minimisation du critère prédictif nécessite la résolution effective d'un problème d'optimisation en ligne. Seule la commande prédictive des systèmes linéaires invariants dans le temps, restreinte au cas sans contraintes, ne nécessite pas la résolution effective de ce problème d'optimisation en ligne, car le correcteur est à son tour linéaire invariant et sa description analytique peut être obtenue hors-ligne. Malheureusement, le type du

système à commander envisagé dans ce travail, non-linéaire et sous contraintes, implique qu'il faut s'orienter vers des techniques prédictives non-linéaires nécessitant la mise en œuvre d'une stratégie d'optimisation qui prend en compte les contraintes imposées sur le système, comme envisagé ci-dessous.

I.3 Eléments de la Commande Prédictive

Les difficultés liées à l'implémentation d'une commande prédictive non linéaire ont conduit au développement d'une stratégie de commande à optimisation sur un horizon fini. Cette nouvelle commande porte différentes dénominations (commande optimale non linéaire à horizon fini, commande non linéaire par horizon glissant ou commande prédictive non linéaire). Nous retenons pour la suite comme dénomination la commande prédictive non linéaire (figure I.2) qui implique les trois aspects suivants :

- **Modèle de prédiction** : un modèle du système pour prédire l'évolution future des sorties sur l'horizon de prédiction Np : $\hat{Y} = [\hat{y}(k)\ \hat{y}(k+1)\ \ldots \hat{y}(k+Np)\ ]$.
- **Méthode d'optimisation :** pour calculer une séquence de commandes sur l'horizon de commande $Nu{:}\theta = [u^*(k)\ u^*(k+1)\ \ldots u^*(k+Nu)]$qui minimise le critère d'optimisation J en satisfaisant les contraintes imposées par l'utilisateur, sachant que $u^*(k+i) = u^*(k+Nu)$ pour $Nu \leq i \leq Np$.
- **Principe de l'horizon glissant :** qui consiste à déplacer l'horizon $k \rightarrow k+1$ à chaque période d'échantillonnage après l'application de la première commande $u^*(k)$de la séquence optimale ainsi obtenue.

Remarque :

Généralement, une des raisons principales du succès de la commande prédictive sur les procédés relativement lents est le temps suffisamment long pour pouvoir résoudre le problème d'optimisation associé avant la fin de la période d'échantillonnage [k; k + 1] (le temps réel). Par contre si le système à commander est relativement rapide où le critère associé au problème d'optimisation est non convexe, la période d'échantillonnage est trop courte pour permettre le calcul de la séquence de commandes recherchée.

Dans la figure I.2 on montre la structure basique, de la stratégie de commande prédictive.

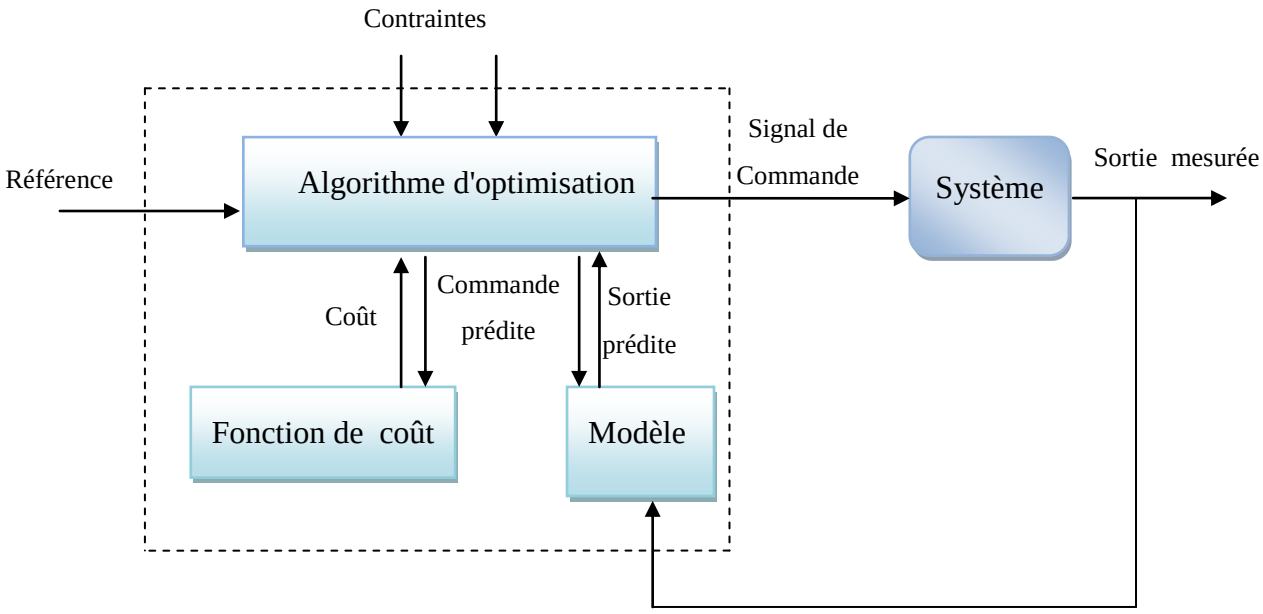


Figure I.2 : Structure de base d'une commande prédictive

Pour l'implémentation de la stratégie prédictive, la structure de base de la figure I.2 est mise en œuvre. Un modèle sert à prédire les futures sorties du système, grâce aux valeurs courantes et passées de la commande et aux commandes optimales futures. Ces dernières sont calculées par une méthode d'optimisation, qui prend en compte la fonction de coût (qui dépend aussi des consignes futures), et éventuellement des contraintes. Donc, Le modèle du système à commander joue un rôle central dans la commande prédictive. Le modèle choisi doit être capable de prendre en compte la dynamique du processus pour prédire précisément les sorties futures, les éléments de la commande prédictive qui doit intervenir lors de la conception sont traités dans les paragraphes suivants.

I.3.1 Fonction objectif

Dans le cas général, la commande prédictive consiste à minimiser une fonctionnelle *J* appelée critère de performance, fonction de coût ou fonction objectif :

$$J_Q(u, y, w) = \int_T F\big(u(T), y(T), w(T)\big)dT \qquad (I.1)$$

Ce critère dépend des commandes appliquées (l'entrée du système) u, des grandeurs de sortie du système y et du comportement désiré w. Toutes ces grandeurs évoluent en fonction du temps.

Pour un système continu, Test un intervalle continu $[t, t+T]$ où T représente le temps futur de prédiction. L'ensemble Q regroupe les contraintes sur la sortie et la commande imposées le long de l'horizon. Ces contraintes sont liées à la dynamique du système et, d'autre part, des contraintes fonctionnelles agissant sur les entrées et les sorties du système.

Dans ce cas où, on utilise un prédicteur non linaire, il n'existe pas de solution analytique, la résolution du problème de commande prédictive est obtenue, par un algorithme d'optimisation numérique implémenté sur un calculateur, donc, le problème est ramené à un problème d'optimisation en dimension finie.

L'intervalle d'optimisation est une succession d'éléments temporels pour lequel les objectifs traduisent des comportements désirés ou seulement en segments successifs du temps, la fonction objectif devient alors sous forme discrète :

$$J_{Q_k} = \sum_{Q_k} (.)\Delta Q_k \qquad (I.2)$$

En discrétisant ce critère, deux valeurs s'introduisent naturellement dans la commande prédictive :

- la longueur des suites d'échantillons de commande, c'est-à-dire l'horizon de commande N_u.
- La longueur sur laquelle est évaluée le critère de performance, c'est-à-dire l'horizon de prédiction N_P.

Les divers algorithmes MPC proposent différentes fonctions de coût pour obtenir la loi de commande. L'objectif principal consiste à faire en sorte que la sortie future

pour l'horizon de prédiction considéré s'approche de la meilleure façon possible de la trajectoire de référence $\{w(k)\}$ tout, en même temps, pénalisant l'effort de commande $\{\Delta u(k)\}$ nécessaire. Une expression générale de fonction objectif [15] adaptée à cette tâche est donnée par:

$$J(u,y,w) = \sum_{i=N1}^{Np} [y(k+i/k) - w(k+i/k)]^2 + \sum_{i=1}^{Nu} \lambda\, [\Delta u(k+\ i\ -1/k)]^2 \qquad (\text{I}.3)$$

Remarque :

Dans quelques méthodes de la commande prédictive le deuxième terme, relatif à l'effort de commande, n'est pas pris en compte.

Donc, le problème de la commande prédictive est la détermination de la séquence des commandes qui permet de minimiser le critère de performance choisi tout en assurant une vérification des contraintes, Une séquence de N_u Commande est appliquée de l'instant présent k jusqu'à la fin de l'horizon de commande, c'est-à-dire à $k + N_u$. Ensuite, les commandes appliquées jusqu'à la fin de l'horizon de prédiction $k + N_p$ sont prises égales au dernier élément de la séquence. Cela signifie qu'il doit réaliser une estimation des prédictions des valeurs de la sortie *y* aux instants d'échantillonnage futurs en fonction des valeurs futures de l'entrée u , c'est l'étape de la modélisation.

On appelle *w* la grandeur de consigne dont la valeur est supposée connue non seulement à l'instant *k* présent, mais également pendant les Np instants d'échantillonnages suivantes, avec :

- $w(k+i)$ consigne appliquée à l'instant $k+i$.
- $y(k+i)$ sortie prédite à l'instant $k+i$.
- $Nu \leq Np$ et $\Delta u(k+\ j)\ =\ 0, \forall\ i \geq Nu$.
  - Nu est l'horizon de commande.
  - $N1$ est l'horizon d'initialisation.
  - Np est l'horizon de prédiction.
  - λ est le facteur de pondération de la commande.
- Δ est l'opérateur de différence $\Delta u(k) = u(k) - u(k-1)$.

Le coefficient λ permet de donner plus ou moins de poids à la commande par rapport à la sortie, de façon à assurer la convergence lorsque le système de départ présente un risque d'instabilité [15].

I.3.2 Modélisation

Les différentes stratégies du MPC emploient différents modèles pour représenter la relation entre la sortie et l'entrée du système. Donnons une vision rapide de ces types de modélisation qui peut se faire selon deux méthodes :

1. Modèle de connaissance :

Il s'agit ici de tenir en compte des phénomènes physiques mis en jeu. On fait alors intervenir des bilans d'énergie, de masse, etc. Le modèle est dans ce cas rarement simple en terme d'entrée-sortie. Il est principalement non linéaire, décrit par un ensemble d'équations différentielles avec seulement la variable temps comme variable indépendante. Ce modèle est décrit par des équations aux dérivées partielles.

La complexité du modèle dépend alors des exigences de description souhaitées mais surtout du niveau de précision exigé pour le comportement désiré du système. L'importance des divers phénomènes peut se quantifier et le modèle obtenu permet de simuler le procédé avec d'autres caractéristiques physiques et dimensionnelles. D'autre part, il est évident que plus le modèle est fidèle au procédé, au sens physique du terme, meilleure sera la prédiction de l'évaluation du comportement du procédé. Cependant la méthode nécessite une connaissance précise dans le domaine concerné.

2. Modèle de comportement global entrée-sortie :

À partir d'un modèle de type boîte noire, choisi a priori, il s'agit ici d'effectuer une estimation de ses paramètres. Ceux-ci sont déterminés en fonction de données expérimentales d'entrée-sortie .l'avantage de l'approche peut s'avérer plus simple et plus rapide que dans le cas précédent. Par ailleurs, il peut être très difficile, de mettre en équation le comportement de nombreux systèmes. Le modèle n'a a priori aucune signification physique, surtout s'il est de nature complexe. D'autre part, et contrairement

à la première méthode, on peut plus difficilement simuler le comportement d'un procédé ayant d'autres caractéristiques physiques et dimensionnelles.

Remarque :

Le choix de la méthode se fait bien sûr en fonction de la précision des objectifs à atteindre et des informations disponibles. D'un point de vue pratique, la méthode d'identification reste encore la plus utilisée, car c'est la plus simple et la plus rapide de mise en œuvre. Cependant, les problèmes posés étant de plus en plus complexes et non linéaires, l'utilisation d'une représentation non linéaire tend à s'étendre.

- **Modèle de prédiction du processus :**

Le modèle du système joue donc un rôle central dans la commande prédictive. Le modèle choisi doit être capable de rendre compte de la dynamique du processus pour prédire précisément les sorties futures et aussi doit être simple à implémenter et à comprendre.

La méthodologie prédictive requiert la définition d'un prédicteur à *Np*-pas en avant qui permette d'anticiper le comportement du processus dans le futur sur un horizon fini. Pour cela, à partir de la forme du modèle, on détermine la sortie estimée à l'instant k+i, connaissant la sortie à l'instant *k.*

Dans ce mémoire, un modèle neuro-flou est utilisé pour l'approximation du modèle entrée/sortie d'un système dynamique non linéaire. Dans cette structure de prédiction, les valeurs passées des entrées et des sorties sont utilisées pour prédire la sortie actuelle du système.

Le prédicteur associé est donné par :

$$y_p(k) = f[y(k-1), \dots, y(k-n_a), u(k-n_k), \dots, u(k-n_b-n_k+1)] \quad \text{(I. 4)}$$

Le vecteur de mesures de sorties et entrées passées s'écrit sous la forme suivante :

$$\varphi(k) = [y(k-1), \dots, y(k-n_a), u(k-n_k), \dots, u(k-n_b-n_k+1)] \quad \text{(I. 5)}$$

La sortie prédite est paramétrée en fonction des paramètres du modèle neuro-flou :

$$y_{NF}(k) = f_{NF}[\varphi(k), v] \quad (I.6)$$

Où

$\boldsymbol{y_{NF}}$: est la prédiction neuro-flou d'un pas en avant de sortie;

v : Paramètres du modèle neuro-flou, voir (II. 17);

$na\ , nb\ et\ nk$: sont respectivement les ordres du système et le retard.

La construction du prédicteur neuro-flou, de telle sorte que les sorties du réseau soient proches des sorties désirées, sera traitée dans le deuxième chapitre.

I.3.3 Optimisation

Contrairement au cas linéaire où le problème est convexe, la CPNL nécessite la résolution d'un problème non linéaire non convexe. On propose dans ce mémoire une discussion du problème d'optimisation en utilisant les algorithmes génétiques présentés dans le troisième chapitre, premièrement on va formuler le problème d'optimisation sous contraintes sous forme d'un problème sans contraintes mais pénalisé, puis on applique les algorithmes génétiques pour résoudre ce problème.

Les prochains paragraphes se proposent d'analyser les différentes contraintes considérées habituellement dans l'industrie des processus [15] en point de vue des restrictions physiques, de la sécurité et du point de vue du comportement désiré, et de présenter la forme dans laquelle elles doivent être formulées pour les décrire dans l'étape d'optimisation.

I.3.3.1 Types des contraintes

Les différentes méthodologies de la commande MPC permettent d'anticiper la violation des restrictions compte tenu de leur caractère prédictif.

▪ **Restrictions sur l'amplitude du signal de commande :**

Les restrictions sur l'amplitude du signal de la commande, assez fréquentes en pratique (pour prendre en compte, par exemple, des effets de saturation des actionneurs), peuvent s'exprimer au moyen de l'inégalité suivante:

$$u_{min} \leq u(.) \leq u_{max} \quad \text{(I. 7)}$$

Ces contraintes sont à satisfaire sur tout l'horizon Nu d'optimisation :

$$u_{min} \leq u(k) \leq u_{max} \quad \text{(I. 8)}$$

Où:$U(k) = [u(k)u(k+1) \ldots u(k+Nu-1)]^T$ le vecteur de dimension Nu.

▪ **Restrictions sur la vitesse de variation du signal de commande :**

Les restrictions sur l'augmentation du signal de commande prennent une forme très simple, et peuvent être exprimées au moyen de l'inégalité:

$$\Delta u_{min} \leq u(k+1) - u(k) \leq \Delta u_{max} \quad \text{(I. 9)}$$

Ou

$$\Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max} \quad \text{(I. 10)}$$

Sachant que $\Delta u(k) = u(k+1) - u(k)$

▪ **Restrictions sur l'amplitude de la sortie :**

Il est très fréquent de trouver comme spécification désirée dans les processus commandés que leur sortie se trouve dans une plage autour d'une trajectoire désirée, par exemple, dans les cas de poursuite d'un certain profil avec une certaine tolérance. Ce type de condition peut être introduit pour le système de commande le forçant à ce que la sortie du système soit à tout moment comprise dans la bande constituée par la trajectoire indiquée plus ou moins la tolérance ceci se traduit par une inégalité de la forme:

$$y_{min} \leq y(k) \leq y_{max} \quad \text{(I. 11)}$$

▪ **Restrictions sur la vitesse de variation du signal de sortie :**

Les restrictions sur l'augmentation du signal de sortie peuvent être exprimées au moyen de l'inégalité:

$$\Delta y_{min} \leq \Delta y(k) \leq \Delta y_{max} \qquad (I.12)$$

Avec : $\Delta y(k) = y(k) - y(k-1)$

I.3.3.2 optimisation sous contraintes

Pour un problème d'optimisation sous contraintes, en notant :

$\theta = [u(k)\ u(k+1)\ \dots u(k+N_u-1]^T$ L'argument d'optimisation, en un instant k, les problèmes présentés se ramènent au suivant :

$$\min_{\theta \in \mathbb{R}^{N_u}} J(\theta)\ avec \begin{cases} g_{i_{min}} \leq g_i(\theta) \leq g_{i_{max}} & i = 1, \dots, m \\ h_i(\theta) = 0 & i = 1, \dots, r \\ J: \mathbb{R}^{N_u} \to \mathbb{R} \end{cases} \qquad (I.13)$$

Où :

m est le nombre de $g(\theta)$ contrainte de tout type inégalité.

r est le nombre de $h(\theta)$ contrainte de tout type égalité.

Les contraintes décrites précédemment $(I.8), (I.10), (I.11)\ et\ (I.12)$ peuvent s'écrire sous la forme $g_{i_{min}} \leq g_i(\theta) \leq g_{i_{max}}$.

▪ **Notion de faisabilité**

La faisabilité est l'existence d'une solution au problème d'optimisation sous contraintes traduisant le problème de commande. Il s'agit de vérifier que la commande, solution de l'algorithme, permet d'assurer la stabilité du processus en boucle fermée [17]. Cette notion de faisabilité est importante dans le cadre d'une utilisation en temps réel.

On appellera domaine faisabilité, l'ensemble des solutions vérifiant les contraintes. On note :

$$C = \{\theta I g_{i_{min}} \leq g_i(\theta) \leq g_{i_{max}} \; et \; h_i(\theta) = 0 \} \qquad (I.14)$$

On note que dans l'approche de commande prédictive, la résolution du problème (I. 13) doit être effectuée plusieurs fois à un instant donné et change à chaque instant. Même si cela était intégrable dans un logiciel, les méthodes existantes sous cette forme s'adressent à des problèmes de types généraux, nécessitent un temps de calcul lourd pour notre cas.

- **Principe de la méthode :**

Le principe est de remplacer le problème primal par un problème pénalisé. À partir du problème(I. 13), on définit une fonction de pénalité à valeurs positives *l* liée aux contraintes. Elle est alors ajoutée, par pondération d'un coefficient de pénalité positif *M*, au critère de performance *J*, ce qui permet de définir la nouvelle fonction coût J_{tot} à minimiser. Le nouveau problème non contraint, mais pénalisé est :

$$\begin{cases} \min_{\theta} J_{tot}(\theta) = J(\theta) + M.l(g_i(\theta), h_j(\theta)) \\ \theta \in \mathbb{R}^{N_u} \end{cases} \qquad (I.15)$$

Le principe est alors de rechercher la solution à ce problème d'optimisation pénalisé et de choisir le poids *M* pour que la quantité $M.l(g_i(\theta), h_j(\theta))$ soit suffisamment prise en compte au cours de la résolution du problème.

Il n'y a aucune méthode générale pour prendre en compte des contraintes. La méthode utilisée dans notre travail est basée sur les fonctions de pénalités .

En utilisant les fonctions de pénalités on abouti au problème sans contraintes suivantes :

Où $P()$ est une fonction de pénalité. Si aucune violation des contraintes ne se produit, c'est-à-dire, si toutes les contraintes sont satisfaites, alors $P(\theta)$ est mise à zéro; sinon on pose $P()$ égale à une valeur positive donnée par Michalewicz [3], comme suit:

$$P(\theta) = M.\left[\sum_{i=1}^{m}(g_i^+(\theta))^2 + \sum_{i=1}^{r}(h_i(\theta))^2\right] \qquad (I.17)$$

Où:

M :un paramètre de pénalité positif.

$g_i^+(\theta) = Max\big(0, g_i(\theta)\big), pour\ 1 \leq i \leq m.$

$h_i(\theta), pour\ 1 \leq i \leq r$

Algorithme :

Après la reformulation du problème d'optimisation de la commande prédictive, les étapes de calcul du MPC restent les mêmes:

1. À chaque instant k, en disposant d'un modèle de connaissance de la sortie du système, on fait la prédiction de la sortie pour un certain horizon Np, les sorties prédites sont dénotées $y(k+i/k)$ où $k = 1,2\ldots, Np$.

2. La prédiction de la sortie, est utilisée pour calculer le vecteur des futurs signaux de commande $\{u(k+i/k), i = 0,1,\ldots, Nu-1\}$ à travers l'optimisation d'une fonction objectif J_{tot}.

3. Le premier élément$\{u(k)\}$ du vecteur du signal de commande optimale $\{u(k+i/k), i = 0,1,\ldots, N-1\}$ issu du problème précédent est appliqué au système et le reste est rejeté car à l'instant suivant ,la nouvelle sortie $\{y(k+1)\}$ est disponible et en conséquence l'étape 1 est répétée selon le concept de l'horizon fuyant (glissant).

I.3.3.3 Choix des horizons

Comme décrit précédemment, la difficulté du temps continu a été contournée par une discrétisation du temps et de l'argument commande sur un temps fini. Cela a introduit les horizons de prédiction (Np) et de commande (Nu). Le problème connu en commande prédictive depuis ses origines est leur détermination optimale.

Dans le cas de modèles linéaires, des méthodes existent pour bien fixer ces paramètres puisqu'on peut plus facilement établir les réponses de tels systèmes dans le cas de poursuite de consignes dont les dynamiques sont du même ordre que celles du système linéaire à commander [17].

Dans le cas général, cela reste un problème ouvert puisqu'un horizon optimal dépend de la dynamique de la consigne à poursuivre, mais aussi de l'influence des contraintes sur le comportement du procédé [19].

- **Choix de l'horizon de prédiction Np :**

Le choix de l'horizon de prédiction Np joue un rôle important tant par la quantité d'informations fournie à l'algorithme, que du point de vue de la faisabilité numérique du problème d'optimisation. Sa détermination est basée sur des considérations physiques ayant trait au comportement du modèle en boucle ouverte, à l'objectif à atteindre ainsi que dans la prise en compte des contraintes. Cependant, s'il n'existe pas encore de méthode permettant de choisir l'horizon de prédiction optimal vis-à-vis du problème posé, le choix d'un horizon de prédiction variable dans le temps peut être mieux qu'avec un horizon Np constant [19].

D'autre part, pendant les horizons de prédiction, il faut pouvoir prédire le comportement futur du système en y incluant les écarts possibles avec le modèle dus ou non à la commande. Donc, un compromis est à trouver pour ce paramètre entre une grande période de prédiction assurant la maîtrise sur un temps plus long et un petit horizon garantissant de meilleures prédictions du fait de l'information plus adaptée concernant l'écart futur entre le procédé et son modèle.

- **Choix de l'horizon de commande N_u :**

En ce qui concerne le choix de l'horizon de commande *Nu*, une valeur élevée permet a priori, en ayant plus de degrés de liberté, d'atteindre des objectifs plus difficiles. Cependant, le choix de *Nu* = 1 est reconnu comme étant suffisant dans la plupart des cas.

- **Choix du facteur de pondération de la commande λ :**

On peut interpréter le facteur de pondération λ comme 'l'équilibre de la balance'. En effet, si $\lambda=0$, on minimise uniquement dans le critère quadratique, la différence entre la consigne et la sortie prédite. Il peut donc en résulter une commande très forte pouvant faire diverger le processus réel. D'autre part, si λ est très élevé, on pondère alors excessivement la commande qui n'est plus assez 'dynamique' pour mieux suivre la consigne.

Conclusion

Dans ce chapitre, on a décrit la méthodologie de la commande prédictive. On a fait une brève présentation des caractéristiques les plus importantes que présentent les principales méthodes de commande prédictive. On a décrit les principaux éléments qui apparaissent dans ces méthodologies, c'est-à-dire, le modèle de prédiction et la fonction objectif. On a mentionné les principes éléments en abordant les contraintes les plus posées dans la pratique. Tout d'abord, la problématique issue du cahier des charges se traduit en un problème d'optimisation sous contraintes initialement posé en dimension infinie. Une discrétisation du temps et une autre concernant l'argument de commande ont alors permis de poser ce problème en dimension finie. Cela a également permis d'introduire les paramètres de réglage essentiels de la commande prédictive. La structure de commande en boucle fermée a été présentée permet d'intégrer dans le problème d'optimisation sous contraintes en dimension finie non seulement les mesures issues du procédé, mais aussi son comportement futur par le biais du modèle. Cette approche a été décrite pour une problématique générale. L'objet du chapitre qui suit est de présenter les outils méthodologiques nécessaires à l'extension de l'utilisation de la stratégie de commande prédictive du cadre classique à celui où le système à commander est décrit par un modèle neuroflou.

Chapitre II:

Systèmes Neuro-Flous

Introduction

On a supposé dans le premier chapitre qu'à chaque instant, on a une prédiction du comportement futur du système non linéaire à commander, alors dans ce chapitre, on va présenter la structure de notre prédicteur. Notre problème de prédiction peut être formalisé de la manière suivante : étant donné une séquence de commande futur données à l'instant *'k'* et les mesures de sorties passées du procédé , le modèle de prédiction choisi doit prédire l'évolution de la sortie sur le long de l'horizon de prédiction *'$k:k+Np$'* avec une certaine précision suffisante .Des développements actuels sur le problème de prédiction montrent que les performances des systèmes neuro-flous dépassent celles d'autres méthodes en termes de précision et d'efficacité [23].

Dans ce chapitre, on va présenter un rappel sur réseaux de neurones, la logique floue puis une vue globale sur les systèmes neuro-flous et enfin l'architecture de prédicteur proposée.

II.1 Rappels sur les réseaux de neurones artificiels (RNA)

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur calcule une sortie unique sur la base des informations qu'il reçoit.

II.1.1 Neurone formel

Le premier modèle du neurone formel a été présenté par Culloch et Pitts (figure II.1). D'une façon générale, un neurone formel est un élément de traitement qui fait une sommation pondérée de n entrées $p_1 \dots, p_R$. Si cette somme dépasse un certain seuil (fonction d'activation f), le neurone est activé et transmet une réponse dont la valeur est celle de son activation. Si le neurone n'est pas activé, il ne transmet rien, comme il est indiqué par la figure suivante:

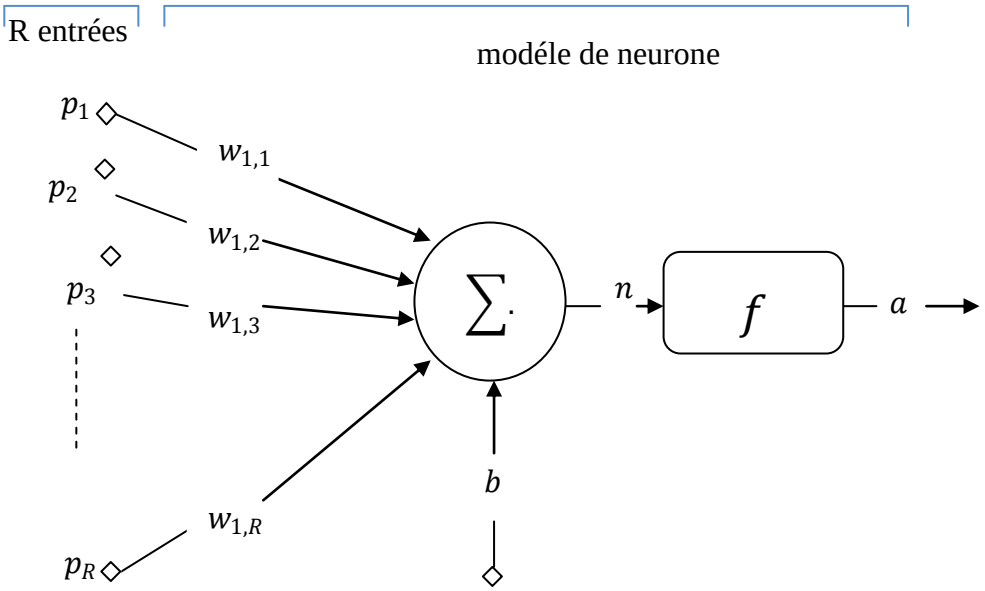


Figure II.1 : représentation du neurone artificiel

Avec :

- $\boldsymbol{P_1, \dots, P_R}$ **:** entrées.
- $\boldsymbol{W_{1.1}, \dots, W_{1.R}}$ **:** poids sur les entrées.
- $\boldsymbol{b}$: biais; (déplacement horizon. de f).
- $\boldsymbol{n = net}$ **:** niveau d'activation**.**

- $\boldsymbol{f}$**:** fonction de transfert.
- $\boldsymbol{a = out}$**:** sortie du neurone.

Les entrées du neurone sont désignées par P_j $(j = 1, n)$**.** Les paramètres W_j reliant les entrées aux neurones sont appelées poids synaptique ou tout simplement poids. La somme pondérée des signaux d'entrée constitue l'activation du neurone. C'est une fonction qui définit l'activité du neurone, elle est appelée aussi fonction de seuillage ou de transfert.

II.1.2 La fonction d'activation (transfert)

L'objectif de cette fonction dite aussi fonction d'activation est de rendre la sortie bornée. Le tableau suivant illustre les fonctions de transfert les plus utilisées :

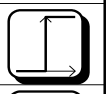
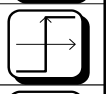
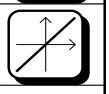
Nom de la fonction	Relation entrée/sortie	Forme
seuil	$a = \begin{cases} 0 & \text{si } n < 0 \\ 1 & \text{si } n \geq 0 \end{cases}$	
seuil symétrique	$a = \begin{cases} -1 & \text{si } n < 0 \\ 1 & \text{si } n \geq 0 \end{cases}$	
linéaire	$a = n$	
linéaire saturée	$a = \begin{cases} 0 & \text{si } n < 0 \\ n & \text{si } 0 \leq n \leq 1 \\ 1 & \text{si } n > 1 \end{cases}$	
linéaire saturée symétrique	$a = \begin{cases} -1 & \text{si } n < -1 \\ n & \text{si } -1 \leq n \leq 1 \\ 1 & \text{si } n > 1 \end{cases}$	
linéaire positive	$a = \begin{cases} 0 & \text{si } n < 0 \\ n & \text{si } n \geq 0 \end{cases}$	
sigmoïde	$a = \dfrac{1}{1+e^{-n}}$	
tangente hyperbolique	$a = \dfrac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	
compétitive	$a = \begin{cases} 1 & \text{si } n \text{ maximum} \\ 0 & \text{autrement} \end{cases}$	

Tableau II.1 : Fonction d'activations les plus utilisées

II.1.3 Définition des réseaux de neurones artificiels (RNA)

Un réseau de neurones est une structure de traitement parallèle et distribuée d'information, comportant plusieurs éléments, de traitement (neurones), avec topologie spécifique d'interconnexion entre ces éléments, et une loi d'apprentissage pour adapter les poids des connexions. Chaque élément de traitement à une sortie unique branchée à plusieurs connexions collatérales qui transmettent le même signal, qui est la sortie du neurone. dans un réseau de neurones donné, l'information est traitée par un grand nombre très important d'autres processeurs.

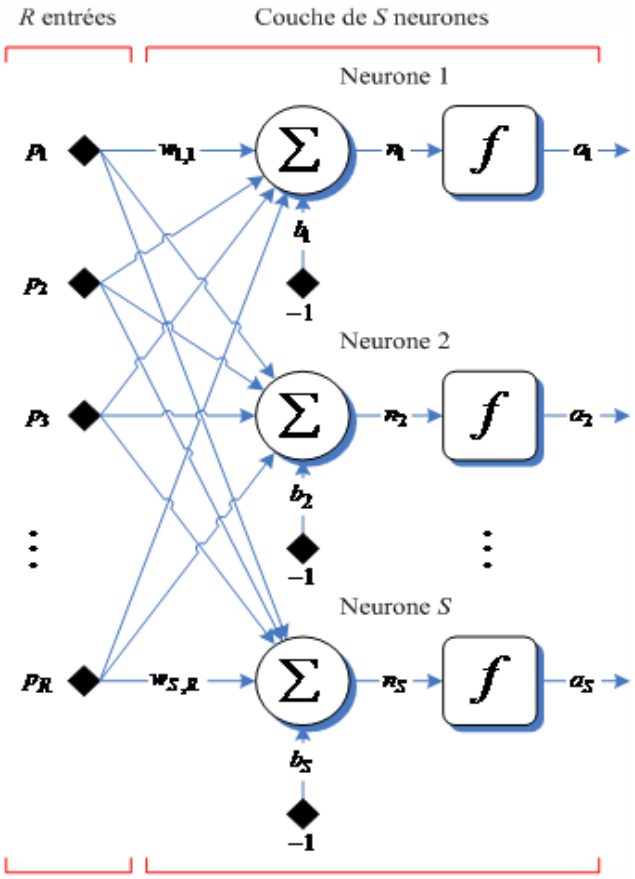


Figure II.2 : Une couche de S neurone.

Les caractéristiques de (RNA) sont :

- Un nombre fini de processeurs élémentaires (neurones).
- Liens pondérés passant un signal d'un neurone vers d'autres.
- Plusieurs signaux d'entrée par neurone.

- Un seul signal de sortie

II.1.4 Apprentissage

L'apprentissage est donc défini comme étant tout algorithme d'ajustement des coefficients synaptiques (poids). Donc, les variables modifiées pendant la phase d'apprentissage sont généralement les poids des connexions entre neurones. Le réseau est testé plusieurs fois au fur et à mesure que l'on ajuste les poids, avant qu'il satisfasse à une réponse désirée. Une fois le but est réalisé les poids seront fixé et on peut alors passer à la phase d'utilisation du réseau, principalement il existe deux types :

- **Apprentissage supervisé :** Dans ce type d'apprentissage, on présente au réseau une entrée et la sortie désirée correspondante, les coefficients synaptiques sont alors ajustés dans le but de minimiser un critère de coût. Une fois l'apprentissage est effectué, le réseau est apte a accomplir la tâche prévue. Les performances du réseau sont évalués à l'aide d'un ensemble d'exemples (de même nature que l'ensemble apprentissage ou d'entraînement) dit *ensemble de test*. La méthode la plus utilisée pour ce type d'apprentissage est la rétropropagation**.**
- **Apprentissage non supervisé :** Dans ce type d'apprentissage l'adaptation des coefficients synaptiques n'est pas basée sur la comparaison avec une certaine sortie désirée, mais c'est le réseau qui organise lui-même les entrées qui lui sont présentées de façon à optimiser une certaine fonction de coût, sans lui fournir d'autres éléments de réponses désirées. Cette propriété est dite Auto-organisation (self organisation).

L'apprentissage comprend généralement quatre étapes :

- *Initialisation des poids synaptiques du réseau,* dans la pratique, cela se fait par des petites valeurs aléatoires non nulles.
- *Présentation du signal d'entrée :* le signal d'entrée est présenté en entrée du réseau, une sortie réelle sera calculée. Ce calcul est effectué de proche en proche de la couche d'entrée vers la couche de sortie, appelé "propagation d'activation".
- *Calcul d'erreur* : dans le cas où on a un apprentissage supervisé, l'erreur tiendra compte de la différence entre le signal d'entrée et le signal de référence.

- *Calcul du vecteur de correction* : à partir du vecteur d'erreur, on détermine la correction à apporter sur les poids synaptiques des connexions et aux seuils des neurones.

▪ **La rétropropagation :**

l'algorithme de rétropropagation du gradient qui est le plus connu pour réaliser l'adaptation des réseaux multicouches. Il s'agit d'une méthode d'apprentissage supervisé, fondée sur la modification des poids du réseau dans le sens contraire à celui du gradient de l'erreur par rapport à ces poids. La mesure de performance utilisée est l'erreur quadratique :

$$J = \frac{1}{2}\sum_{i}[a_i - s_i]^2 \qquad \text{(II. 1)}$$

Ou i parcourt les indices des neurones de sortie, et a_i et s_i représentent respectivement la sortie mesurée et sortie désirée pour ces neurones. Les poids du réseau sont modifiés en suivant la règle :

$$\Delta W_{ij} = -\eta \frac{\partial J}{\partial W_{ij}} \qquad \text{(II. 2)}$$

Où η est une constante positive appelée *pas d'apprentissage*. Le calcul de la quantité $\frac{\partial J}{\partial W_{ij}}$ se fait en partant de la couche de sortie et en se déplaçant vers la couche d'entrée. Cette propagation, suivant le sens inverse de celui de l'activation des neurones du réseau, justifie le nom de l'algorithme.

Cet algorithme, présenté ici dans sa version la plus simple, possède de nombreuses variantes. En utilisant la méthode du deuxième ordre pour le calcul du gradient, on abouti à une formule requérante :

$$\Delta W_{ij}(t) = -\eta \frac{\partial J}{\partial W_{ij}} + \alpha \Delta W_{ij}(t-1) \qquad \text{(II. 3)}$$

α est une constante appelée *moments d'inertie*, et t représente le temps.

II.2 Rappels sur les systèmes flous

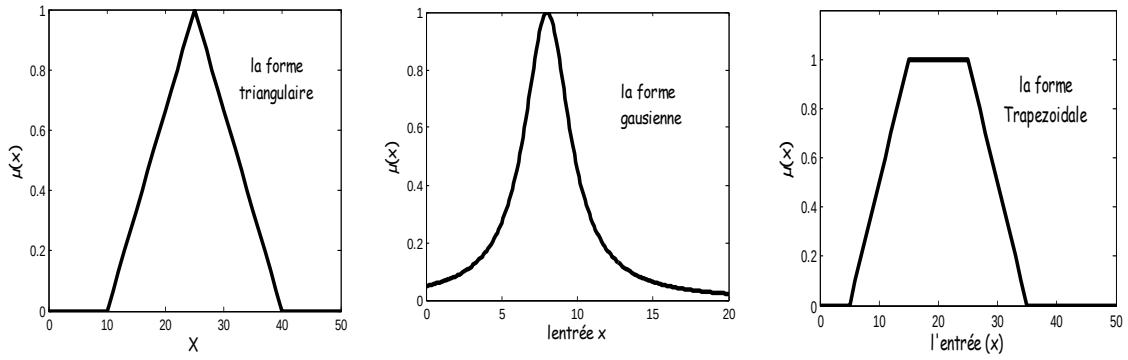
Les systèmes flous peuvent être considérés comme des systèmes logiques qui utilisent des règles linguistiques pour établir des relations entre leurs variables d'entrée et de sortie. Ils sont apparus pour la première fois dans les années soixante dix avec des applications dans le domaine du contrôle des processus.

Aujourd'hui, les applications des systèmes flous sont très nombreuses outre la commande, ils sont utilisés aussi pour la modélisation . Nous présentons brièvement quelques notions de base de ces systèmes.

II.2.1 Ensembles flous

La notion d'ensemble flou a été proposée par Zadeh [25] en introduisant un caractère graduel de l'appartenance d'un élément à un ensemble donné. Cela permet une meilleure représentation des termes et des connaissances vagues que nous manipulons dans notre vie quotidienne.

Mathématiquement, un ensemble flou A d'un univers de discours U, est caractérisé par une fonction d'appartenance (Figure. II.3), notée μ_A, a une valeur dans l'intervalle [0,1] et qui associe à chaque élément x de U un degré d'appartenance $\mu_A(x)$ indiquant le niveau d'appartenance de x à A. $\mu_A(x) = 1\ et\ \mu_A(x) = 0$ correspondent respectivement à l'appartenance et la non-appartenance.



Forme triangulaire Forme gaussienne Forme trapézoïdale

Figure. II.3: Différents types de fonctions d'appartenance.

L'allure de la fonction d'appartenance est à choisir selon l'application traitée. La figure II.3, illustre les différentes formes des fonctions d'appartenance les plus utilisées. Cependant, dans certaines applications où l'on doit dériver la fonction d'appartenance, on choisira par exemple des fonctions de type gaussien, continûment dérivables.

II.2.2 Variables linguistiques

Une variable linguistique appelée aussi attribut linguistique peut être définie à partir du triplet (x, U, T_x) où x est une variable définie sur l'univers de discours U et $T_x = A_1, A_2, ..$ est un ensemble composé de sous ensembles flous de U qui caractérise x. On associe souvent à chaque sous-ensemble flou de T_X une valeur ou un terme linguistique (étiquette).

Il est généralement imposé que les ensembles flous A_i doivent satisfaire la condition suivante:

$$\forall x, \exists i, \mu_{A_i}(x) \neq 0 \qquad \text{(II. 4)}$$

Cette condition appelée dans la littérature, propriété d'assurance (coverage property) , exige que chaque élément soit affecté au moins à un ensemble flou avec un degré d'appartenance non nul. À cette condition, on ajoute souvent une propriété supplémentaire qui est le respect de la sémantique : les sous ensembles doivent interpréter réellement les termes linguistiques qui leurs ont associés.

II.2.3 Règles et opérateurs flous

On appelle proposition floue élémentaire, une proposition de type X est A où (X, U, T_x) est une variable linguistique et A un sous-ensemble de T_x. Une telle proposition possède un degré de vérité égal à $\mu_A(x)$ où x est une valeur réelle de X. D'une manière générale, on peut combiner ces propositions élémentaires à l'aide des opérateurs logiques de conjonction et de disjonction ('et' et 'ou') mis en œuvre respectivement par des T-normes et T-conormes [25]. Le degré de vérité des nouvelles propositions obtenues peut être calculé entre autres par les équations suivantes:

Conjonction: (X est A) ET (Y est B)

– *minimum* $(\mu_A(x), \mu_B(y))$

– *produit* $\mu_A(x) \times \mu_B(y)$

Disjonction: (X est A) OU (Y est B)

– *maximum* $(\mu_A(x), \mu_B(y))$

– *somme* $\mu_A(x) + \mu_B(y) - \mu_A(x) \times \mu_B(y)$

L'opérateur d'implication permet d'introduire la notion de règle floue qui caractérise les relations de dépendance entre plusieurs propositions floues:

$$Si\ (X_1\ est\ A_1)\ ET\ (X_2\ est\ A_2)\ Alors\ (Y\ est\ B) \qquad \text{(II. 5)}$$

Où X_1, X_2 et Y sont des variables linguistiques et A_1 , A_2 et B sont des sous-ensembles flous. Dans cette dernière formulation la partie (X_1 est A_1) ET (X_2 est A_2) est appelée prémisse de la règle et la partie (Y est B) est appelée conclusion (conséquent).

II.2.4 Structure générale d'un système flou

De manière classique, le fonctionnement interne d'un système flou repose sur la structure présentée par la figure suivante qui inclut quatre blocs:

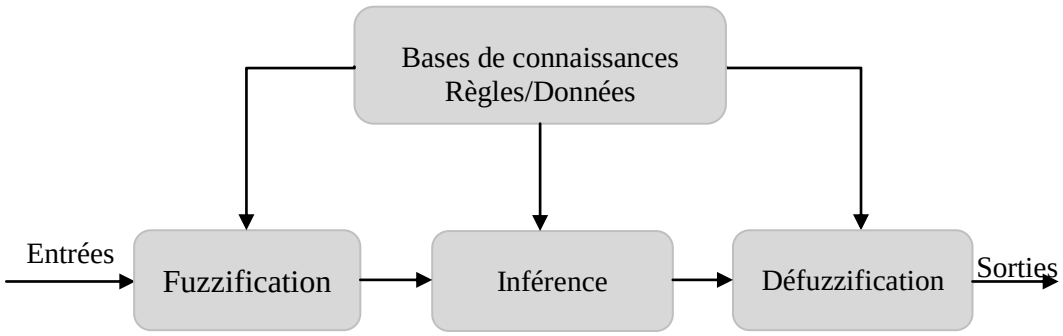


Figure II.4 : Structure d'un système flou.

▪ **La base de connaissances :** elle contient les définitions des fonctions d'appartenance (formes et paramètres) associées aux variables d'entrée/sortie ainsi que l'ensemble des règles floues.

▪ **La Fuzzification** consiste à calculer, pour chaque valeur d'entrée numérique, les degrés d'appartenance aux ensembles flous associés et prédéfinis dans la base de données du système flou. Ce bloc réalise la transformation des entrées numériques en informations symboliques floues utilisables par le mécanisme d'inférence.

▪ **Le mécanisme d'inférence** consiste d'une part à calculer le degré de vérité des différentes règles du système et d'autre part à associer à chacune de ces règles une valeur de sortie.

Cette valeur de sortie dépend de la partie conclusion des règles qui peut prendre plusieurs formes. Il peut s'agir d'une proposition floue, et l'on parlera dans ce cas de règle de type Mamdani:

$$Si\ (\ldots\ldots)\ Alors\ Y\ est\ B, B\ ensemble\ flou$$

Il peut également s'agir d'une fonction réelle des entrées, et l'on parlera dans ce cas de *règle de type Sugeno*:

$$Si\ (\ldots\ldots)\ Alors\ y\ =\ f(x_1, x_2, \ldots, x_n)$$

où $x1, \ldots, xn$ sont les valeurs réelles des variables d'entrées.

Dans le cas d'une règle de type Mamdani, la sortie est un sous-ensemble flou obtenu à partir de celui présent dans la conclusion de la règle, soit en lui appliquant un facteur d'échelle égal au degré de vérité de la permisse, on parle alors dans ce cas de la méthode d'inférence PRODUIT, soit en le tronquant à la valeur de ce degré de vérité et on parle dans ce cas de la méthode d'inférence MINIMUM [25].

▪ **La défuzzification** consiste à remplacer l'ensemble des valeurs de sorties des différentes règles résultant de l'inférence par une valeur numérique unique représentative de cet ensemble. Dans le cas des règles de type Sugeno, le calcul se fait simplement par une somme normalisée des valeurs associées aux règles floues.

Dans le cas de règles de Mamdani, le calcul de la valeur numérique de sortie s'effectue en deux étapes:

1. Composition des règles : Une fois la phase d'inférence terminée, il s'agit de regrouper (par union) les sous ensemble flous issus de l'inférence pour en obtenir un seul ensemble représentatif des différentes conclusions des règles floues. Comme méthode de composition, on peut citer en particulier les compositions MAXIMUM (en général couplée avec l'inférence MINIMUM) et SOMME (en général couplée avec l'inférence PRODUIT).

La première consiste à caractériser l'ensemble des sorties par une fonction d'appartenance égale au maximum des fonctions d'appartenance des sous-ensembles flous. La deuxième consiste à faire la somme de fonctions d'appartenance des sous-ensembles issus de l'inférence .

2. Passage du symbolique vers le numérique : C'est la phase de défuzzification proprement dite qui permet de générer une valeur numérique à partir de l'ensemble obtenu par composition des règles.

II.3 Systèmes Neuroflous

Les systèmes Neuroflous permettent de combiner les avantages de deux techniques complémentaires. Les systèmes flous fournissent une bonne représentation des connaissances. L'intégration de réseaux de neurones au sein de ces systèmes flous améliore leurs performances grâce à la capacité d'apprentissage de réseaux de neurones. Inversement, l'injection des règles floues dans les réseaux de neurones, souvent critiques pour leur manque de lisibilité, clarifie la signification des paramètres du réseau et facilite leur initialisation, ce qui représente un gain de temps de calcul considérable pour leur identification.

Figure II.5 : Principe du système Neuroflou.

Afin de clarifier les définitions, nous proposons dans ce chapitre une présentation bref des quelques types des systèmes Neuroflous.

II.3.1 Définition

Les systèmes Neuroflous 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 comme il est montré sur la figure suivante.

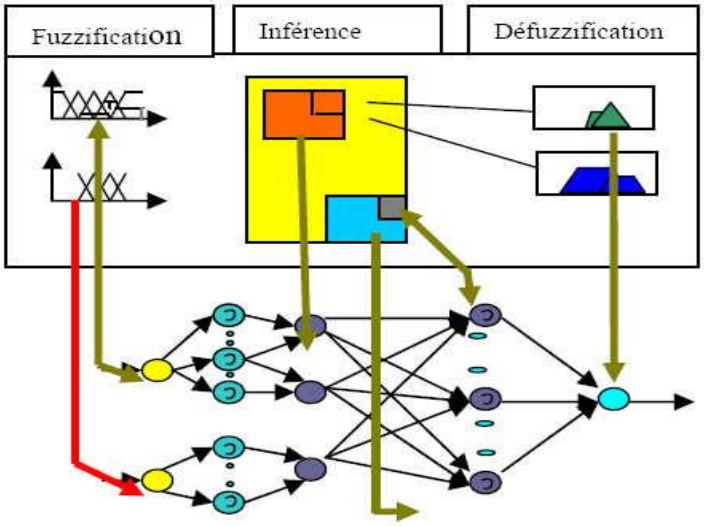


Figure II.6 : Principe de fonctionnement d'un Réseau Neuroflou.

II.3.2 Méthodes des combinaisons neuro-floues

Il existe quatre grandes catégories de combinaisons des réseaux de neurones avec la logique floue [23]:

1. **Réseau flou neuronal :** Dans ces réseaux, les techniques floues sont employées pour augmenter les possibilités du processus d'apprentissage et d'exécution des réseaux de neurones.
2. **Système neuronal/flou simultanément :** Le réseau de neurone et le système flou fonctionnent ensemble sur la même tâche, mais sans s'influencer, c à d. ni l'un ni l'autre n'est employé pour déterminer les paramètres de l'autre. Habituellement le réseau neuronal traite les entrées, ou post-traite les sorties du système flou.

3. **Modèles neuro-flous coopératifs :** Le réseau de neurone 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 de neurone. C'est une forme simple des systèmes neuro-flous.
4. **Modèles neuro-flous hybrides :** Les approches neuro-floues modernes sont de cette forme. Un réseau neuronal et un système flou sont combinés dans une architecture homogène. Le système peut être interprété comme un réseau neuronal spécial avec des paramètres flous ou comme un système flou mis en application sous une forme distribuée parallèle.

II.3.3 Architectures neuro-floues

Plusieurs architectures ont été développées, on peut les classer en trois types:

II.3.3.1 Première architecture

La première méthode neuro-floue consiste au codage du système d'inférence floue sous la forme d'un réseau de neurones multicouches dans lequel les poids correspondent aux paramètres du système. L'architecture du réseau dépend du type de règle et des méthodes d'inférences, d'agrégation et de défuzzification choisies. Cette situation est schématisée sur la figure II.7.Les fonctions d'appartenance intervenante dans les règles sont considérées comme des paramètres ajustés par l'intermédiaire des poids entrant dans la première couche cachée. Les conclusions w_i des règles sont également des paramètres ajustables par l'intermédiaire des poids associés à la dernière couche.

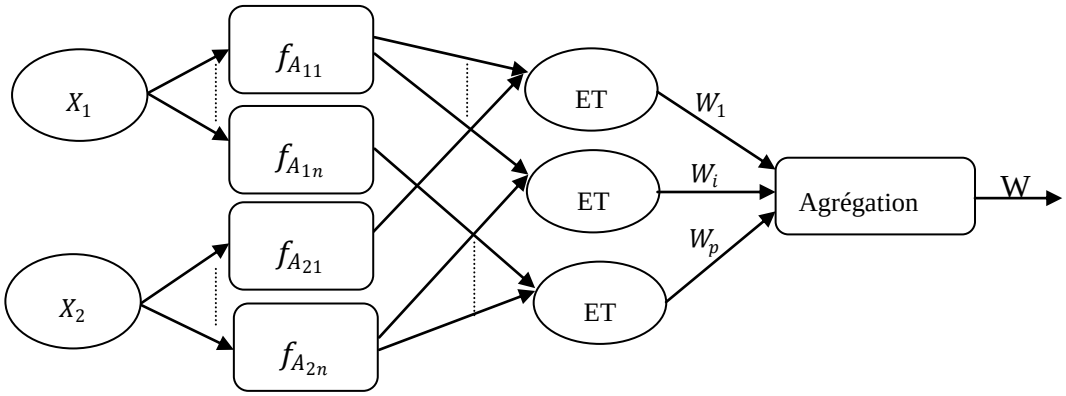


Figure II.7 : Première architecture Neuro-floue

Comme il est indiqué la figure ci-dessus, les données vont subir trois étapes de calcul :

▪ **La première couche :** s'occupe du calcul des degrés d'appartenance de chaque entrée. Les paramètres de cette couche vont caractériser la fonction d'appartenance. Le calcul des degrés d'appartenance s'effectue par des neurones spécialisés, dont la fonction d'activation est la fonction d'appartenance. La fonction la plus utilisée dans ce type d'architecture est la fonction gaussienne.

▪ **La deuxième couche cachée** évalue en parallèle les prémisses des règles. Les paramètres de cette couche définissent dans ce cas l'opérateur de conjonction ET. Plusieurs méthodes ont été proposées pour l'évaluation floue de l'opérateur ET (cette évaluation est appelée aussi calcul de la valeur de vérité). Généralement on utilise celle qui a été proposée par Lukasiewicz [23], cette dernière présente une certaine facilité d'implémentation neuronale satisfaisante. Elle est définie par :

$$ET[f(A_1(x_1), f(A_2(x_2)] = MAX(0, f(A_1(x_1) + f(A_2(x_2) - 1) \qquad \text{(II. 7)}$$

▪ **La dernière couche :** réalise l'opération d'inférence. Ayant calculé les valeurs de vérités produites par les opérateurs de conjonction, on doit maintenant établir une procédure permettant de déduire une sortie en fonction de ces valeurs de vérités. Cette procédure est réalisée soit en choisissant parmi ces valeurs, celle qui est la plus représentative, c'est à dire celle qui a le plus grand degré de vérité, soit en calculant le barycentre de toutes les valeurs. Une telle approche peut être effectuée par un seul neurone, où les valeurs de vérités sont pondérées par les poids synaptiques de ce neurone.

Les poids synaptiques du neurone qui calcule le barycentre (qui fait la défuzzification) sont d'une grande importance, car ce sont eux qui pondèrent les résultats des règles. Un apprentissage serait indispensable à appliquer sur cette couche (c'est-à-dire sur les poids synaptiques du neurone chargé de la défuzzification).

▪ **Algorithme d'apprentissage :** Cet algorithme est basé sur l'algorithme de la rétropropagation du gradient descendant dont les entrées du neurone chargé de la défuzzification sont normalisées. Cette normalisation est nécessaire car elle permet de manipuler des valeurs inférieures ou égales à l'unité. Une telle approche est réalisée par la division de chaque entrée de ce neurone, sur la somme de toutes ses entrées.

II.3.3.2 Deuxième architecture

Cette méthode utilise des réseaux de neurones et des systèmes flous associés en série ou en parallèle. Plusieurs variantes sont ainsi possibles :

- Le réseau de neurones fonctionne en amont du système flou (figure II.8). Les variantes d'entrées du système flou sont déterminées à partir des sorties du réseau de neurones (dans le cas où elles ne sont pas mesurables directement) ou encore un réseau de neurones effectue une tâche de classification ou de reconnaissance de formes, suivie d'un système flou d'aide à la décision.

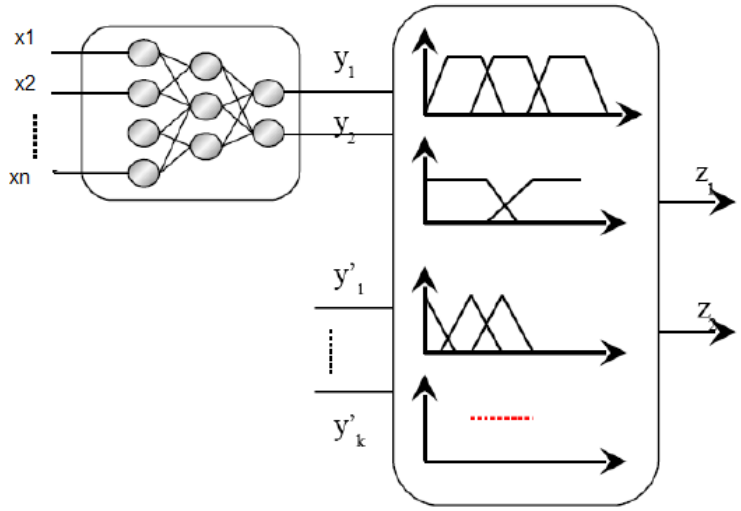


Figure II.8 : Deuxième architecture des réseaux neuro-flou réalisation en série

- Un réseau de neurones qui fonctionne en aval du système flou (figure II.9), dans le but d'ajuster les sorties d'un système de commande floue à de nouvelles connaissances obtenues, les variables de sorties étant les erreurs sur les variables de sortie du système flou.

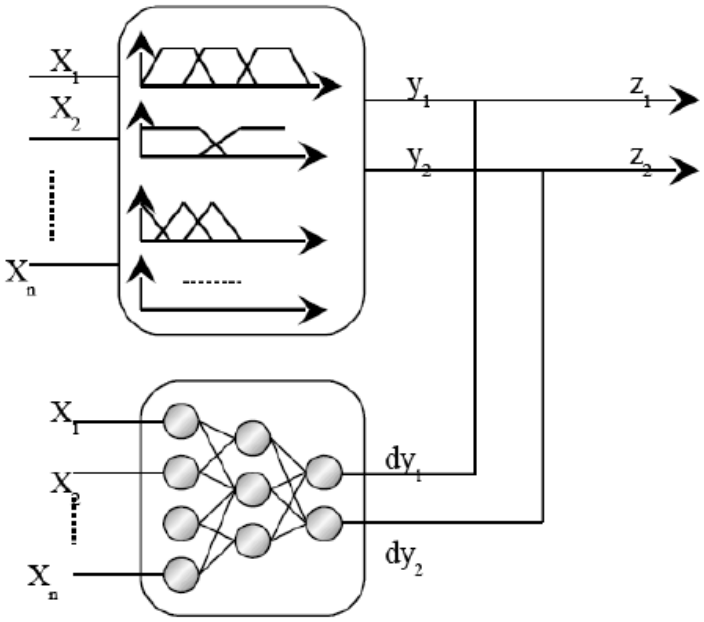


Figure II.9 : Réseau de neurones fonctionnant en aval d'un système flou

II.3.3.3 Troisième architecture

Ce type d'association entre réseau de neurones et systèmes flous correspond à l'utilisation des réseaux de neurones pour remplacer toutes ou quelques composantes d'un système flou. De tels réseaux servent à l'apprentissage des fonctions d'appartenance, au calcul de l'inférence, à la réalisation de la phase d'agrégation et de défuzzification. Ils peuvent réaliser l'extraction des règles floues, en analysant la corrélation qui existe entre les entrées et les sorties du réseau de neurones.

Ces approches ont une grande importance car elles sont capables de résoudre ces problèmes :

- La détermination et l'apprentissage des fonctions d'appartenances.
- La détermination des règles.
- L'adaptation à l'environnement du système.

II.3.4 Quelques types des systèmes neuroflous

Diverses associations des méthodes et architectures neuro-floues ont été développées depuis 1988, [23]. La suivante montre quelques modèles des systèmes

neuro-flous. FALCON (a) et GARIC (b) interprètent la règle floue avec une structure à 5 couches, NEFCLASS (c) interprète la règle floue avec une structure à 3 couches et ANFIS (d) interprète la règle floue avec une structure à 6 couches.

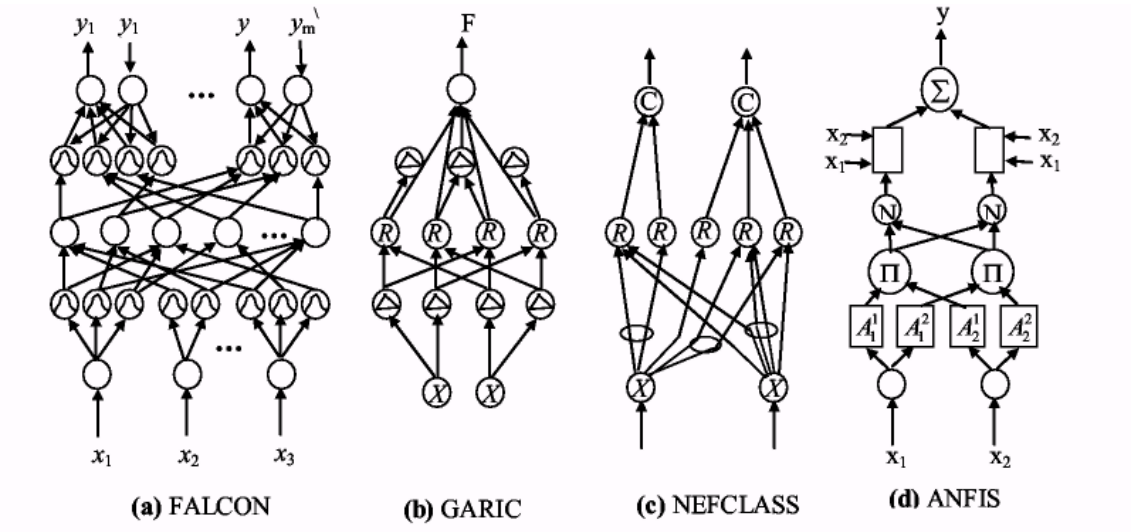


Figure II.10 : Différents types des systèmes neuro-flous

- **FALCON (Fuzzy Adaptive learning Control Network):**

II s'agit de modèles à 5 couches, utilisant la fuzzification en entrée et la défuzzification en sortie. Ceci correspond à 1'interprétation juste de la technique de Mamdani. La précision accrue des résultats provoque une lenteur dans 1'exécution du système. Ce modèle est rarement utilisé en pratique mais il reste le meilleur pour la commande [23].

- **NEFCLASS :**

Modèle utilisé généralement en classification, il est constitué de 3 couches : Une couche d'entrée avec les fonctions d'appartenance, une couche cachée représentée par des règles et une couche de sortie définissant les classes [23] .Ce modèle est facile à mettre en application, il évite 1'étape de défuzzification, tout en étant précis dans le résultat final, avec une rapidité bien supérieure aux autres architectures types.

- **NEFPROX (Neuro Fuzzy function approximator) :**

Modèle obtenu par 1'association des deux architectures NEFCLASS et NEFCON, il est utilisé dans différentes applications comme la classification et l'approximation de fonctions [23]. NEFCLASS utilise un algorithme supervisé pour définir les règles

floues, le NEFCON utilise un algorithme d'apprentissage non supervisé avec le calcul de 1'erreur de sortie. Les deux modèles emploient la rétropropagation afin de définir les sous-ensembles flous. Comparé au modèle ANFIS, NEFPROX est beaucoup plus rapide, mais ANFIS donne de meilleurs résultats en approximation. Le NEFPROX est, le premier système interprétable et lisible, dédié à l'approximation de fonction. Néanmoins, ses résultats en classification restent moins bons que ceux donnés par le NEFCLASS.

- **ANFIS (Adaptative-Network-based Fuzzy Inference System):**

ANFIS représente un système à inférence floue mis en application dans le cadre des réseaux adaptatifs. II utilise la procédure d'apprentissage hybride. Cette architecture affine les règles floues obtenues par des experts humains pour décrire le comportement d'entrée-sortie d'un système complexe. Ce modèle donne de très bons résultats en poursuite de trajectoire, approximation non linéaire, commande dynamique et traitement du signal.

III.4 Modèle de prédiction neuro-flou

La commande prédictive non linéaire nécessite la disponibilité d'un modèle fiable, fidèle et précis décrivant le système à commander et reflétant ses non-linéarités et ses complexités dynamiques. Parmi les choix possibles pour la réalisation d'une structure de modèle non linéaire, les réseaux de neuroflous présente une solution prometteuse grâce à leurs capacités prouvées, théoriquement et pratiquement, à l'approximation des fonctions non linéaires avec une précision arbitrairement choisie *a priori.*

Notre travail traite plus spécifiquement le système ANFIS (Adaptive Neuro-Fuzzy Inference System) proposé par Jang, [26] avec une certaine modification au niveau de la phase d'apprentissage. Le système ANFIS est une classe de réseau adaptatif. Il peut être vu comme un réseau de neurones non bouclé pour lequel chaque couche est un composant d'un système neuro-flou et, à ce titre, c'est un "approximateur "universel. Il est ainsi utilisé dans différentes applications de prédictions.

Un système ANFIS réalise une approximation linéaire de la variable de sortie en décomposant l'espace des entrées en différents espaces flous.

III.4.1 Architecture de l'ANFIS

ANFIS (***Adaptive Network Based Fuzzy Inference System***) c'est un système d'inférence adaptatif neuro-flou qui consiste à utiliser un réseau neurone à 5 couches pour lequel chaque couche correspond à la réalisation d'une étape d'un système d'inférence floue de type Takagi Sugeno. Pour la simplicité, nous supposons que le système d'inférence flou à deux entrées *x* et *y*, et z comme une sortie. Supposer que la base de règle contient deux règles floues de type Takagi-Sugeno.

Règle1 :

$$SI\ x\ est\ \mathbf{A1}\ et\ y\ est\ \mathbf{B1}\ ALORS\ z1 = p1\,x + q1\,y + r1 \qquad (II.8)$$

Règle2 :

$$SI\ x\ est\ \mathbf{A2}\ et\ y\ est\ \mathbf{B2}\ ALORS\ z2 = p2\,x + q2\,y + r2 \qquad (II.9)$$

L'ANFIS à une architecture posée par cinq couches comme représenté sur la figure suivante :

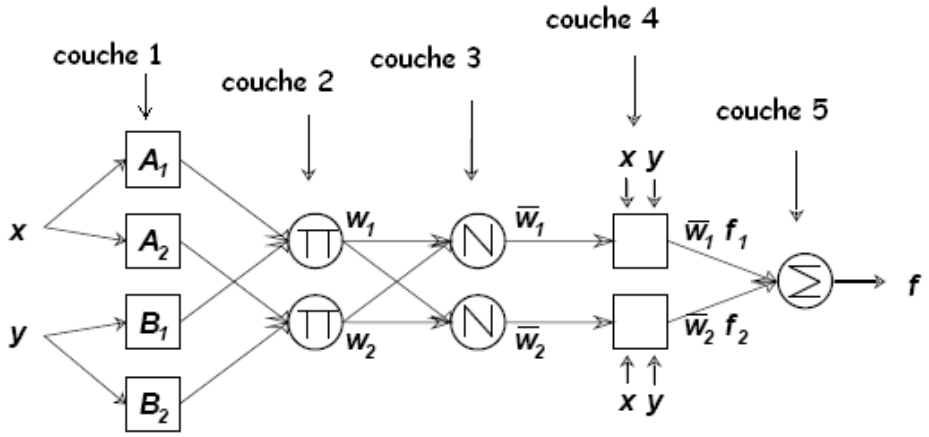


Figure II.11 : L'Architecture de l'ANFIS.

1. La première couche (Fuzzification)

Cette couche (layer 1) permet la **"fuzzification "**des entrées *x* et *y* . Chaque neurone dans cette couche correspond à une variable linguistique. Les entrées x et y sont passées par la fuzzification en utilisant des fonctions d'appartenances des variables linguistiques A_i et B_j, (généralement sont des formes triangulaires, trapézoïdales ou Gaussiennes). Par exemple,la fonction d'appartenance Gaussienne est définie par:

$$\mu_{A_i}(x) = exp\left[-\frac{1}{2}\frac{(x-\bar{x}_i)^2}{{\sigma_{x_i}}^2}\right] \quad \text{(II. 10)}$$

$$\mu_{B_j}(y) = exp\left[-\frac{1}{2}\frac{\left(y-\bar{y}_j\right)^2}{{\sigma_{y_j}}^2}\right] \quad \text{(II. 11)}$$

Où $\bar{x}$ est le centre et. σ la largeur de la fonction d'appartenance.

Les sorties de la première couche sont :

$$x_{1,i} = \mu_{A_i}(x) \quad \text{pour i} = 1,2$$
$$y_{1,j} = \mu_{B_j}(y) \quad \text{pour j} = 1,2$$

Alors,la valeur obtenue $\mu_{A_1}(X)$ représente le degré d'appartenance de la valeur x à l'ensemble A.

2. La deuxième couche (règles)

Chaque nœud correspond à une T-Norme floue (l'opérateur T-Norme permet de réaliser l'équivalent d'un "ET" booléen). Il reçoit la sortie des nœuds de fuzzification et calcule sa valeur de sortie grâce à l'opérateur produit (Cet opérateur est généralement utilisé mais il en existe d'autres : max, min ...).

La fonction d'activation des neurones ***i*** de la première couche :

$$w_i = min\left\{\mu_{A_i}(x), \mu_{B_j}(y)\right\}, i = 1,2, j = 1,2 \quad \text{(II. 12)}$$

ou

$$w_i = \mu_{A_i}(x) \times \mu_{B_j}(y), i = 1,2, j = 1,2 \quad \text{(II. 13)}$$

3. La troisième couche(Normalisation)

Cette couche normalise les résultats fournis par la couche précédente. Les résultats obtenus représentent le degré d'implication de la valeur dans le résultat final.

$$\overline{w}_i = \frac{w_i}{\sum_{i=1}^{2} w_i} \quad \text{(II. 14)}$$

L'ensemble des sorties de cette couche sont appelées :les poids normalisés.

4. La quatrième couche (conséquents)

Chaque nœud de cette couche est relié aux entrées initiales. On calcule le résultat en fonction de son entrée et d'une combinaison linéaire du premier ordre des entrées initiales (Approche de TAKAGI - SUGENO).

$$f_i^4 = y_i = \overline{w}_i \times (p_i x_1 + q_i x_2 + r_i) \quad \text{(II. 15)}$$

Où $\overline{W_i}$est la sortie de la troisième couche, et $\{p_i, q_i, r_i\}$ sont l'ensemble des paramètres désignés sous le nom : *conséquents*.

5. La couche de sortie(sommation)

Elle est constituée d'un seul neurone qui calcule la somme des signaux de la couche précédente,alors:

$$y = \sum_{i=1}^{2} y_i \quad \text{(II. 16)}$$

La figure suivante représente un système ANFIS, à 2 entrées chaque entrée repartie en trois sous ensemble floue et 9 règles.

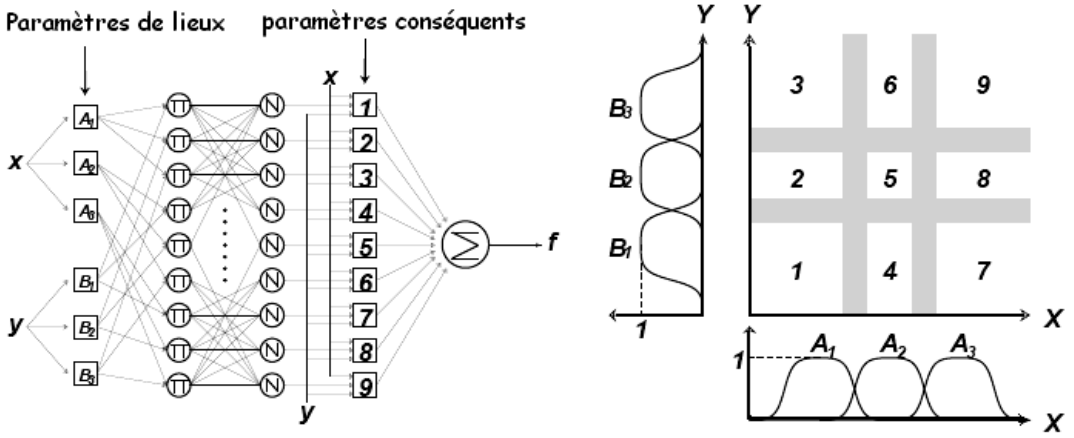


Figure II.12 : Exemple ANFIS à 2 entées avec 9 règles.

Les différentes couches	**Type des couches**	**Le nombre de neurone dans la couche**
Couche 0	Les entrées	n
Couche 1	Fuzzification	$(p.n)$
Couche 2	Les règles	p^n
Couche 3	La normalisation	p^n
Couche 4	Linéarisation des fonctions	p^n
Couche 5	Somme	1

Tableau II.2 : les différentes couches d'un système ANFIS

Tel que :

n : le nombre des entrées.

p : le nombre des sous ensembles flous d'entrée (partition flou).

Noter que les neurones dans ANFIS ont différentes structures:

- Valeurs [fonction d'appartenance définie par différentes formes].
- Règles [habituellement produit].
- Normalisation [division de somme et d'arithmétique].
- Fonctions [régressions linéaires et multiplication avec $\overline{W}$, tel que $\overline{W}$ est la normalisation du poids w].
- la sortie [Somme Algébrique].

III.4.2 Algorithme d'apprentissage

En supposant des ensembles flous du type gaussien, le réseau ANFIS décrit, ci avant comporte 14 paramètres devant être optimisés (8 concernant les fonctions gaussiennes et 6 à la linéarisation des sorties des règles). À cette fin, une base d'apprentissage est nécessaire.

L'apprentissage consiste en la correction des paramètres (prémisses et conséquents) du réseau afin de généraliser une fonction de transfert (inconnue *a priori*) entre les entrées et la sortie du réseau. Celle-ci est constituée d'un ensemble de couples "entrées /sortie "connus (relevé de données). Le déploiement des algorithmes d'apprentissage sur cette base de données permet de construire **une fonction d'approximation** (de prédiction dans notre cas) de la sortie (sortie désirée) à partir des nouveaux vecteurs d'entrée.

Les points forts de l'ANFIS consistent dans le mécanisme d'inférence distributif et. Par contre les règles sont non interprétables et l'apprentissage se fait hors ligne.

Nous allons présenter l'algorithme proposé pour l'apprentissage (i.e optimisation) de la structure du réseau neuro-flou. Le principe de cet algorithme est très simple: il consiste dans un premier temps et après avoir choisi les formes des fonctions d'appartenances ont mis les 14 valeurs à optimiser sous forme vectorielle :

$$v = \left[p_i \; q_i \; r_i \;\; \sigma_{x_i} \; \sigma_{y_j} \;\; \bar{x}_i \;\; \bar{y}_j\right] \;\; , i = 1{,}2, j = 1{,}2 \qquad \text{(II. 17)}$$

Ensuite on réécrit les mesures et les entrées sous forme de matrice $M = [nxm]$

$n : nombre\ d'entries$

m : $nombre\ d'échantillons\ de\ mesures$

Pour obtenir les données appropriées et nécessaires pour remplir la matrice M, il faut choisir les signaux d'excitation à appliquer aux entrées du système. L'entrée appliquée sur le procédé à commander pour collecter les données doit avoir certaines caractéristiques :

- elle ne doit pas être constante, ou par paliers.
- elle doit être homogène à la sortie en terme de valeur : si la fonction de transfert est une sigmoïde.

Le critère à minimiser pour la mesure de performance reste l'erreur quadratique donnée par l'équation:

$$J = \frac{1}{2}\sum_{k=1}^{m}(y(k) - \hat{y}_{NF}(k))^2 \qquad \text{(II. 18)}$$

Et on peut utiliser ce critère:

$$J = \sum_{k=1}^{m}|y(k) - \hat{y}_{NF}(k)| \qquad \text{(II. 19)}$$

où $y(k) d\ et\ \hat{y}_{NF}(k)$ sont respectivement la sortie désirée et la sortie actuelle du réseau neuroflou.

Enfin, un algorithme génétique est appliqué tout en minimisant le critère J.les résultats de simulations présentés dans le quatrième chapitre montre l'efficacité du modèle ANFIS dans la modélisation des systèmes non linéaires.

III.4.3 Architecture du prédicteur

La structure du modèle neuroflou, définie dans la partie précédente, correspond à une prédiction avec un pas en avant $(Np = 1)$. Pour concevoir un prédicteur de plusieurs pas en avant, un ensemble de réseaux de neuroflous placés en cascade est utilisé; le nombre de réseaux correspond au nombre égal à l'horizon de prédiction Np.Les entrées du premier réseau sont données par le procédé, tandis que

dans le reste, la sortie de chaque réseau ANFIS alimente l'entrée de celui qui le suit et les entrées de commande sont données par l'algorithme d'optimisation qui minimise la fonction objectif (figure II.13). La structure de tous les réseaux neuroflous est identique à celle décrite dans le paragraphe précédent.

La figure suivante montre le principe d'un prédicteur neuroflou avec *Np(horizon de prédiction)* pas en avant pour chaque instant k.

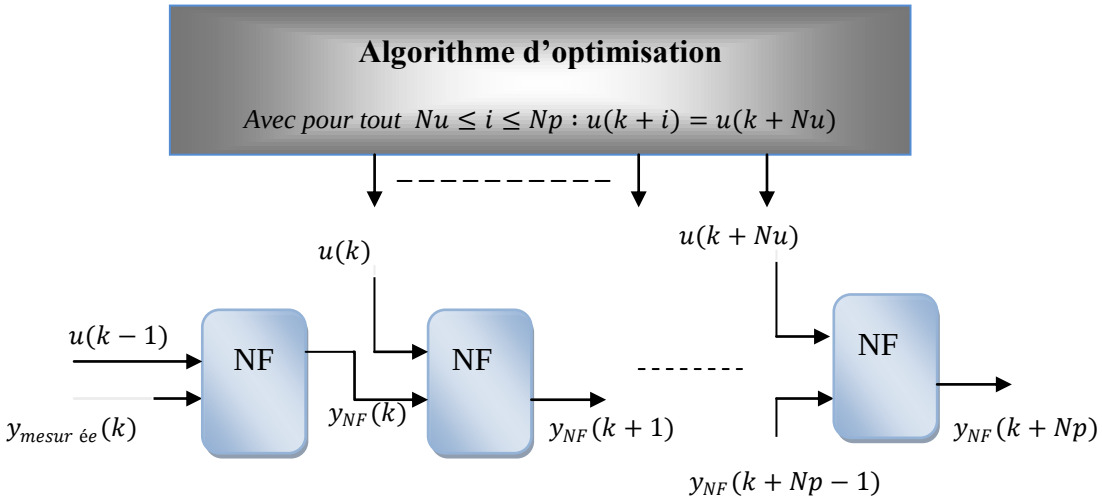


Figure II.13 : Prédicteur neuroflou (NF) avec un horion de prédiction p .

Conclusion

Nous avons présenté dons ce chapitre une description générale réseaux de neurones, la théorie floue, et Neuro-floue et nous nous sommes intéressés principalement aux systèmes ANFIS, alors nous pouvons dire que : L'utilisation conjointe des méthodes neuronales et floues permet de tirer l'avantage des qualités de l'une et de l'autre. Principalement, des capacités d'apprentissage des premières et de la lisibilité et la souplesse des dernières. L'architecture du système neuroflou adopté dans ce chapitre montre des bonnes performances dans les simulations des systèmes non linéaires, dans le quatrième chapitre nous allons essayer de l'inclure dans l'architecture de la commande prédictive en utilisant les algorithmes génétique comme méthode d'optimisation.

Chapitre III:

Algorithmes génétiques

Introduction

Les algorithmes génétiques sont des algorithmes d'optimisation basés sur la théorie de l'évolution des espèces de Charles Darwin qui décrit l'évolution des systèmes biologiques selon le principe de la sélection naturelle. Contrairement aux méthodes classiques d'optimisation, ils sont caractérisés par une grande robustesse et ils peuvent éviter les minimums locaux lors de la recherche d'un minimum global. De plus, ces algorithmes ne nécessitent pas de calculer des dérivées qui présentent un grand obstacle pour les méthodes itératives classiques destinées à optimiser le type de problème décrit dans le premier chapitre (I. 16), telle que la programmation séquentielle quadratique (SQP). Lorsqu'il y a trop de minimums locaux, l'algorithme de SQP peut s'arrêter sur une solution non optimale. Cependant, l'algorithme génétique a la propriété d'exploration aléatoire sur l'espace de recherche et cela peut donner une solution efficace du problème de la commande optimale à horizon fini.

Le présent chapitre a pour objectif de proposer un résumé sur les principes des algorithmes génétiques dans le cadre d'optimisation.

III.1 Principe des algorithmes génétiques

Les algorithmes génétiques présentent des qualités intéressantes pour la résolution de problèmes d'optimisation non linéaire. Leurs fondements théoriques tentent de simuler le processus d'évolution des espèces dans leur milieu naturel. La génétique représente un individu par un code, c'est-à-dire un ensemble de données (appelés chromosomes), identifiant complètement l'individu. La reproduction est, dans ce domaine, un mixage aléatoire de chromosomes de deux individus, donnant naissance à des individus enfants ayant une empreinte génétique nouvelle, héritée des parents. La mutation génétique est caractérisée dans le code génétique de l'enfant par l'apparition d'un chromosome nouveau, inexistant chez les individus parents. Ce phénomène génétique d'apparition de " mutants" est rare, dans le sens d'une meilleure adaptation au milieu naturel. La disparition de certaines espèces est expliquée par " les lois de survie " selon lesquels seuls les individus les mieux adaptés auront une longévité suffisante pour générer une descendance. Les individus peu adaptés auront une tendance à disparaître. C'est une sélection naturelle qui conduit de génération en génération à une population composée d'individus de plus en plus adaptés.

Un algorithme génétique est construit de manière tout à fait analogue. Dans l'ensemble des solutions d'un problème d'optimisation, une population de taille N est constituée de N solutions (les individus de la population) convenablement marquées par un codage identifie complètement. Une procédure d'évaluation est nécessaire à la détermination de la force de chaque individu de la population. Viennent ensuite une phase de sélection et une phase de recombinaison (opérateurs de croisement et de mutation) qui génèrent une nouvelle population d'individus, qui ont de bonnes chances d'être plus fortes que ceux de la génération précédente. De génération en génération, la force des individus de la population augmente et après un certain nombre d'itérations, la population est entièrement constituée d'individus tous forts, soit de solutions quasi-optimales du problème posé.

Le fonctionnement d'un AG est alors passe par les phases suivantes :

1. **Initialisation** : une population initiale de taille N chromosomes est tirée aléatoirement.

2. **Évaluation** : chaque chromosome est décodé puis évalué.

3. **Reproduction:** création d'une nouvelle population de N chromosomes par l'utilisation d'une méthode de sélection appropriée.

4. **Opérateurs génétiques:** croisement et mutation de certains chromosomes au sein de la nouvelle population.

5. **Retour** à la phase 2 tant que la condition d'arrêt du problème n'est pas satisfaite.

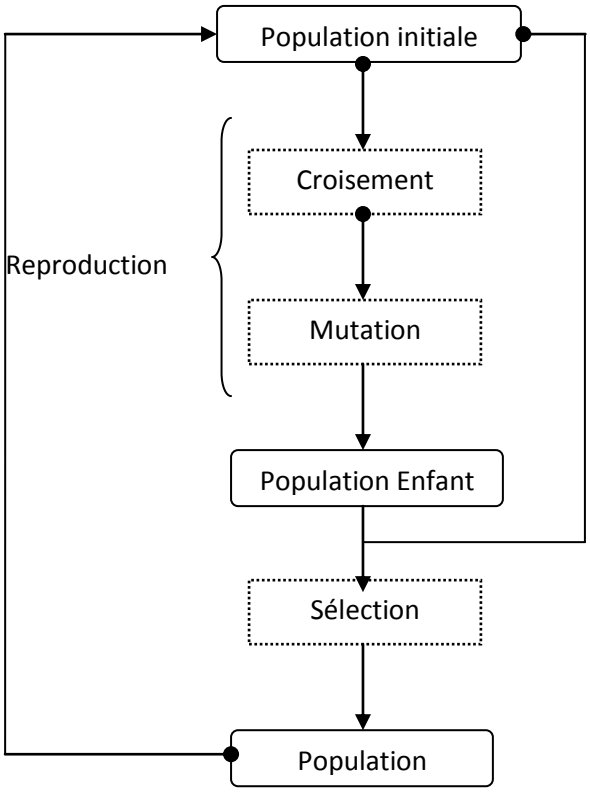


Figure III.1: Principe de fonctionnement d'un algorithme génétique .

Un algorithme génétique a pour but de rechercher un optimum d'une fonction définie sur un espace borné, et repose sur les points suivants :

- un principe de codage de l'élément de population : la qualité du codage des données conditionne le succès de la recherche de l'optimum. On retrouve deux formes de codage : le codage binaire (parfois GRAY) et le codage réel.
- un mécanisme de génération de la population initiale non homogène. Le choix de la population conditionne la rapidité de la convergence vers l'optimum.
- une fonction à optimiser qui retourne une fonction d'adaptation (*fitness*) de l'individu.
- un mécanisme de sélection des individus permettant d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les moins bons.
- des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace des autres solutions possibles. L'opérateur de croisement (*crossing-over*) recompose les gènes d'individus existant dans la population. L'opérateur de mutation a pour but de garantir l'exploitation de l'espace de solutions.
- des paramètres de dimensionnement : la taille de la population, critères d'arrêt, probabilité d'application des opérateurs génétiques.

III.2 Termes et définitions

III.2.1 Population :

C'est l'ensemble des individus, ou encore l'ensemble des chromosomes d'une même génération. Une population P est constituée des individus c_i avec $i = 1 \dots \mu$:

$$P = \{c_1, \dots, c_i, \dots, c_\mu\} \qquad \text{(III. 1)}$$

Généralement, la taille de la population reste constante tout au long de l'algorithme génétique.

III.2.2 Individu

Les individus correspondent aux « solutions » possible de la fonction à optimiser $f(x)$. Ces solutions doivent être « codées » pour que le traitement puisse être effectué par l'algorithme génétique. Cette représentation codée d'une solution est

appelée chromosome, et est composée de gènes. Chaque gène peut représenter une variable, un élément de la solution, ou encore une partie plus abstraite. La manière la plus utilisée de codage par algorithme génétique est le codage en vecteurs. Chaque solution est représentée par un vecteur. Ce vecteur peut être binaire ou encore de n'importe quel type discret dénombrable (entier, caractères, etc.). On pourrait également utiliser nombres réels, mais dans ce cas, il faut également revoir les opérations qui modifient le contenu des chromosomes (la fonction qui crée aléatoirement les chromosomes et les opérateurs génétiques). La simplicité veut que les chromosomes soient uniformes, c'est-à-dire que tous les gènes sont du même type. Cependant, si on tient, compte encore une fois des opérations qui modifient le contenu des chromosomes, on peut assez aisément construire des vecteurs d'éléments de type différents.

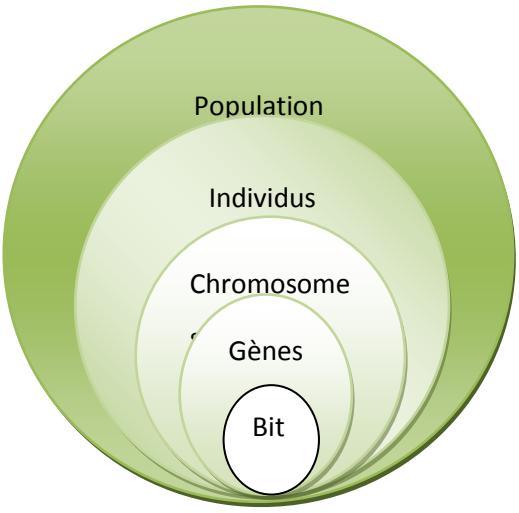


Figure III.2 : Les cinq niveaux d'organisation d'un algorithme

III.2.3 Codage

Toutes les valeurs qu'il peut prendre une solution sont composées de ce gène, on doit trouver une manière de coder chaque allèle différent de façon unique (établir une bijection entre la solution et sa représentation codée).

Un chromosome est une suite de gènes, on peut par exemple choisir de regrouper les paramètres similaires dans un même chromosome et chaque gène sera repérable par sa position.

Chaque individu est représenté par un ensemble de chromosomes, et une population est un ensemble d'individus. Il y a deux principaux types de codage utilisables, et on peut passer de l'un à l'autre facilement :

III.2.3.1 Le codage binaire

Chaque gène dispose du même alphabet binaire {0, 1} Un gène est alors représenté par une suite binaire.

Étant donné une fonction objectif $f(x)$ où le vecteur x constitue de n variable x_i avec $i = 1 \dots n$. La valeur inférieure et supérieure des variables x_i est donnée par $x_{i_{min}}$ et $x_{i_{max}}$ respectivement .En codage binaire, la variable x_i est convertie en premier lieu à une valeur normalisée $x_{i_{norm}}$ donnée par:

$$x_{i_{norm}} = \frac{x_i - x_{i_{min}}}{x_{i_{max}} - x_{i_{min}}} \qquad \text{(III. 2)}$$

Où : $0 \leq x_{i_{norm}} \leq 1$

Ensuite, le nombre normalisé $x_{i_{norm}}$ est converti en nombre binaire c_i. Le nombre de bits de code binaire est dépend de la précision voulue. Le nombre binaire c_i constitué donc de m bits et il est représenté comme suit:

$$c_i = \langle b[1], \dots, b[j], \dots, b[m] \rangle \quad avec\ b[j] \in \{0,1\}\ \forall\, j = 1, \dots, m \qquad \text{(III. 3)}$$

Le codage binaire commence à montrer ses limites. En effet, deux éléments voisins ne codent pas nécessairement deux éléments proches dans l'espace de recherche. Cet inconvénient peut être évité en utilisant un "codage de Gray" : le codage de Gray est un codage qui a comme propriété qu'entre un élément n et un élément n + 1, donc voisin dans l'espace de recherche, un seul bit diffère.

III.2.3.2 Le codage réel

Cela peut-être utile notamment dans le cas où, on recherche à optimiser une fonction réelle. Pour des problèmes d'optimisation dans un domaine continu, une représentation avec les nombres réels est réalisée directement et facilement. Un individu c_i constitué d'un vecteur de nombres réel x_i dont chaque élément du vecteur correspond à une caractéristique de l'individu, donc, un gène. Par conséquent, aucun codage ou décodage est nécessaire. Cela conduit à une mise en œuvre plus simple et plus efficace. La précision pour un codage réel dépend seulement de la puissance de calculateur utilisé.

III.2.4 Fitness

Le calcul de la qualité d'un individu est une étape essentielle dans les algorithmes génétiques. Cette fonction donne, en valeur numérique (habituellement réelle), à la qualité d'un individu. C'est selon cette valeur numérique que sont calculées les chances de sélection de cet individu. Les algorithmes génétiques étant une technique d'optimisation, ils cherchent la meilleure qualité,donc l'optimisation de la fonction de qualité.

La fonction objectif ou critère $f()$ du problème de l'optimisation est une fonction scalaire d'un vecteur de dimension n. Le vecteur x constitue de n variables x_j avec $j = 1,..,n$ qui représente un point dans l'espace réel $\mathbb{R}^n$. Un individu c_i est constitué donc de n gènes:

$$C_i = [c_{i1}, \dots, c_{ij,\dots,c_{in}}] \qquad \text{(III. 4)}$$

Dans la formulation originale de GA, les individus ont été représentés comme nombres binaires qui se composent de deux bits 0 et 1. Dans ce cas, le codage binaire et le codage Gray peuvent être utilisés. Un individu codé en binaire est appelé un chromosome. Dans la représentation réelle, un individu se compose de vecteur de nombres réels.

Pour des problèmes d'optimisation sans contraintes la fonction de fitness ne dépend que de la fonction objectif du problème d'optimisation, La fitness d'un individu est déterminée par la fonction $F(x)$. Dans ce cas, il suffit de mettre la fonction de fitness égale à la fonction objectif.

Par conséquent, pour le problème de l'équation (III.4), on aura :

$$F(x) = f(x) \qquad \text{(III. 5)}$$

III.3 Génération

Au départ d'un algorithme génétique, il faut créer une population d'individus. Ces individus sont générés par une fonction simple. Cette fonction affecte à chaque individu, une valeur aléatoire pour chacun de ses gènes. L'algorithme génétique peut également utiliser comme population de départ une population déjà créée a priori. La transition d'une population P_g à celle prochaine P_{g+1} est appelée la génération où g désigne le nombre de la génération. Dans la figue. III.2 Les opérations exécutées pendant une phase génération sont représentées schématiquement. L'évolution de la population continue à travers plusieurs générations, jusqu'à ce que le problème soit résolu .Dans la plupart des cas, le processus s'arrête quand le nombre maximal de générations g_{max} est atteint.

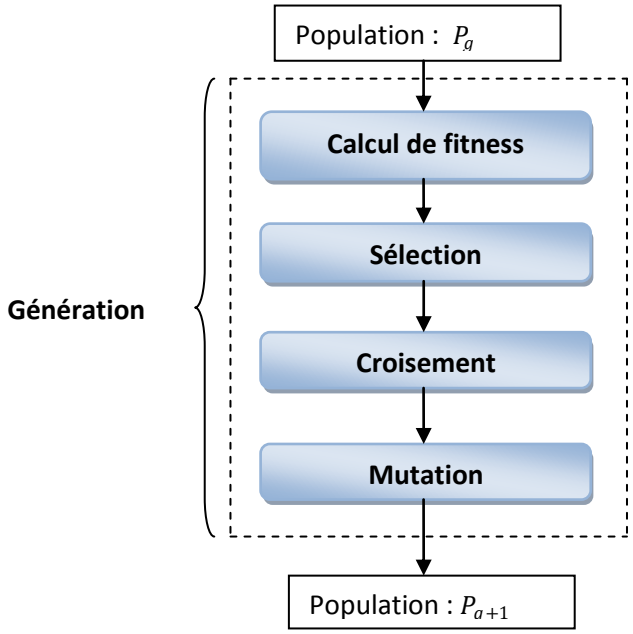


Figure III.3: Représentation des opérations exécutées pendant une génération

L'application de la sélection et des opérateurs génétique, croisement et mutation produit une nouvelle population d'une population existante. Dans la section suivante, ces trois opérateurs sont décrits.

III.3.1 Sélection

Cet opérateur est chargé de définir quels seront les individus de P qui vont être dupliqués dans la nouvelle population $\tilde{P}$ et vont servir de parents (application de l'opérateur de croisement).Soit n le nombre d'individus de P, on doit sélectionner $n/2$ (l'opérateur de croisement nous permet de repasser à n individus).

Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population.

III.3.2 Croisement

Le croisement (crossover) utilisé par les algorithmes génétiques est la transposition informatique du mécanisme qui permet, la production de chromosomes qui héritent partiellement des caractéristiques des parents. Son rôle fondamental est de permettre la recombinaison des informations présentes dans le patrimoine génétique de la population. Cet opérateur est appliqué après avoir appliqué l'opérateur de sélection sur la population *P*; on se retrouve donc avec une population $\hat{P}$ de *n/2* individus et on doit doubler ce nombre pour que notre nouvelle génération soit complète. On va donc créer de manière aléatoire *n/4* couples et on les fait se "reproduire». Les chromosomes (ensembles de paramètres) des parents sont alors copiés et recombinés de façon à former deux descendants possédants des caractéristiques issues de deux parents. Détaillons ce qui se passe pour chaque couple au niveau de chacun de leurs chromosomes :

Un, deux, voire jusqu'à $l_g - 1$ (où lg est la longueur du chromosome) points de croisements sont tirés au hasard, chaque chromosome se retrouve donc séparé en "segments". Puis chaque segment du parent 1 est échangé avec son "homologue" du parent 2 selon une probabilité de croisement p_c . De ce processus résultent deux fils pour chaque couple et notre population est $\hat{P}$ contient donc bien maintenant n individus.

On peut noter que le nombre de points de croisements ainsi que la probabilité de croisement p_c permettent d'introduire plus ou moins de diversité. En effet, plus le nombre de points de croisements sera grand et plus la probabilité de croisement sera élevée plus, il y aura d'échange de segments, donc d'échange de paramètres, d'information, et plus le nombre de points de croisements sera petit et plus la probabilité de croisement sera faible, moins le croisement apportera de diversité.

- **Croisement en codage binaire :**

En binaire la méthode la plus courante consiste à prendre un nombre aléatoire k compris entre 1 et la longueur l_g de la chaîne. À partir des chaînes de caractères de deux individus parents, on crée deux nouvelles chaînes par permutation de tous les caractères

compris entre les positions $k+1\ et\ l_g$ des chaînes. On peut aussi choisir plusieurs lieux de coupe et échanger les parties de chaîne comprises entre les coupes (croisement multiple).

En binaire, deux individus sont sélectionnés au hasard. Ensuite, ils sont divisés au point du croisement i en deux parties et des nouveaux composants. Les deux individus (parents) sont donnés par:

$$C_1 = \left[c_{1,1}, c_{1,2}, \dots, c_{1,i}, c_{1,i+1}, \dots, c_{1,n}\right]$$

$$C_2 = \left[c_{2,1}, c_{2,2}, \dots, c_{2,i}, c_{2,i+1}, \dots, c_{2,n}\right]$$

Après que l'opération du croisement réalisé au point du croisement i, deux nouveaux individus résultent:

$$C_1^{new} = \left[c_{1,1}, c_{1,2}, \dots, c_{2,i}, c_{2,i+1}, \dots, c_{2,n}\right]$$

$$C_2^{new} = \left[c_{2,1}, c_{2,2}, \dots, c_{1,i}, c_{1,i+1}, \dots, c_{1,n}\right]$$

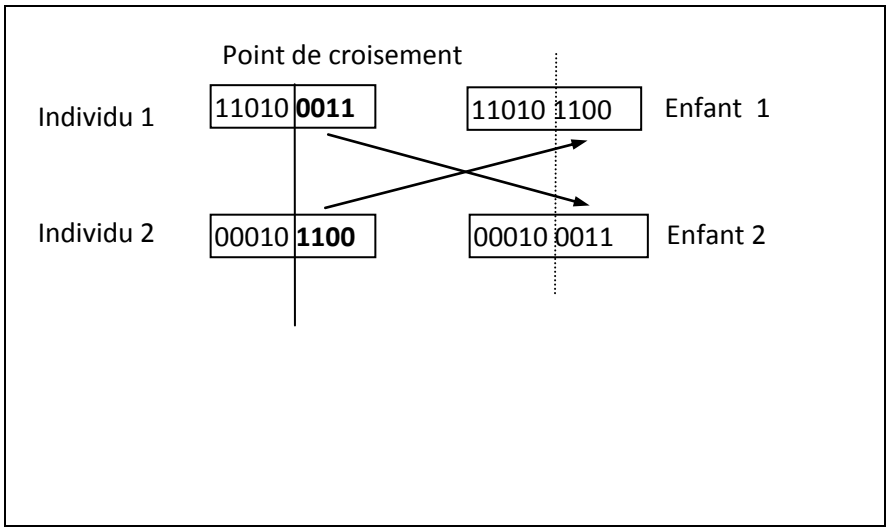


Figure III.3: Croisement dans une Représentation binaire

▪ **Croisement en codage réel :**

Si l'on travaille en codage réel, le croisement le plus courant se fait par échange de gènes ; on peut échanger des gènes au hasard ou générer de nouveaux gènes par combinaison linéaire des gènes des 2 parents (croisement arithmétique). Un individu est un nombre réel. L'utilisation de techniques du croisement arithmétiques donne des bons résultats dans la solution de problèmes de l'optimisation non linéaire . On prend $c_1 \, et \, c_2$ deux individus qui sont à reproduire. Les deux fils $c_1^{new} \, et \, c_2^{new}$ sont des combinaisons linéaires de leurs parents $c_1 \, et \, c_2$:

$$c_1^{new} = \lambda c_1 + (1 - \lambda) c_2 \qquad \text{(III. 6)}$$

$$c_2^{new} = \lambda c_2 + (1 - \lambda) c_1 \qquad \text{(III. 7)}$$

$$\lambda \in [0,1]$$

III.3.2 Mutation

La mutation est un autre composant important des algorithmes génétiques**.** Le rôle de cet opérateur est de produire des changements aléatoires, dans différents chromosomes. Une façon simple de réaliser la mutation est de modifier un ou plusieurs gènes. Dans les algorithmes génétiques, la mutation joue un rôle très important soit en remplaçant les gènes perdus dans la population durant le processus de sélection de telle façon à les essayer dans un nouveau contexte, ou en fournissant les gènes non présents dans la population initiale. L'opérateur de mutation modifie donc de manière complètement aléatoire les caractéristiques d'une solution, ce qui permet d'introduire et de maintenir la diversité au sein de notre population de solutions. Cet opérateur joue le rôle d'un "élément perturbateur", il introduit du "bruit" au sein de la population. Cet opérateur dispose de 4 grands avantages :

• Il garantit la diversité de la population, ce qui est primordial pour les algorithmes génétiques.

• Il permet d'éviter un phénomène connu sous le nom de *dérive génétique*. On parle de dérive génétique quand certains gènes favorisés par le hasard sont ainsi présents au

même endroit sur tous les chromosomes. Le fait que l'opérateur de mutation puisse entraîner de manière aléatoire des changements et il permet d'éviter la répétition de cette situation défavorable.

• La mutation permet d'atteindre la propriété d'ergodicité ; l'ergodicité est une propriété garantissant que chaque point de l'espace de recherche puisse être atteint.

▪ **Mutation en codage binaire :**

Dans cette représentation binaire, on choisit aléatoirement une position dans un gène puis on fait, une inversion de bit correspond, la figue. III.4 représente un exemple pour mutation binaire.

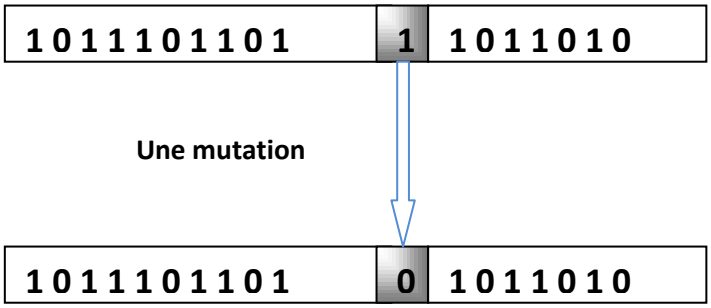


Figure III.4 : Principe de mutation

▪ **Mutation dans le codage réel :**

L'opérateur de la mutation a été développé originalement pour le codage binaire. Ensuite d'autres méthodes ont été développées en tenant compte la modification du gène dans le codage réel. Ces méthodes appliquent une distribution de la probabilité qui est définie sur le domaine des valeurs possibles pour chaque gène. Une nouvelle valeur de gène est calculée après cette distribution de la probabilité. L'opérateur de la mutation change au hasard d'un seul gène ou plus d'un individu sélectionné.

Considérant l'individu $c_i = [c_{i1}, \dots, c_{ij}, \dots, c_{in}]$ et le gène c_{ij} qui sélectionné pour la mutation. L'intervalle est donné par $c_{ij} = [c_{ij,min}\ c_{ij,max}]$, où $c_{ij,min}\ c_{ij,max}$ sont les bornes inferieur et supérieur respectivement.

Il y deux types de mutation dans le codage réel: mutation uniforme, et mutation non uniforme.

- **Mutation uniforme:** L'application de cet opérateur donne un individu $C_i^{new} = [c_{i1}, \dots, \tilde{c}_{ij}, \dots, c_{1,n}]$, où $\tilde{c}_{ij}$ est une valeur aléatoire .L'opérateur de la mutation est appliqué avec une probabilité p_m.
- **Mutation Non-uniforme:** L'application de cet opérateur donne un individu $C_i^{new} = [c_{i1}, \dots, \tilde{c}_{ij}, \dots, c_{1,n}]$, où $\tilde{c}_{ij}$ est une valeur aléatoire calculée par :

$$\tilde{c}_{ij} = \begin{cases} c_{ij} + \Delta(g, c_{ij_{max}} - c_{ij}), & si\ h = 0 \\ c_{ij} + \Delta(g, c_{ij} - c_{ij_{min}}), & si\ h = 1 \end{cases} \qquad \text{(III. 8)}$$

où $\Delta(g,)$est une fonction qui définit l'écart entre la nouvelle valeur et la valeur initiale à la génération g et h est nombre aléatoire qui prend les valeurs 0 ou 1.

III.4 Optimisation génétique

L'algorithme suivant présente le principe d'un GA pour résoudre un problème d'optimisation. Initialement, la fonction de fitness $F(x)$ est définie, en fonction du problème que doit être résolu. Les probabilités p_c , p_m, et la taille de population μ sont choisies. Les individus de la population P_0 initiale, sont initialisés au hasard. Donc, on commence la première génération en calculant la fitness $F(c_i)$ pour chaque individu de la population, avec $i = 1 \dots \mu$. En appliquant la sélection aux individus de la population P_g, une population $\tilde{P}$ transitoire se crée. Par l'application de l'opérateur de croisement avec une probabilité p_c une transition supplémentaire de $\tilde{P}$ donne $\hat{P}$. En appliquant l'opérateur de la mutation, avec la probabilité p_m, aux individus de la population $\hat{P}$ on obtient une nouvelle population P_{g+1} . Si le nombre de la génération maximal n'est pas atteint, alors la fitness est recalculée et les opérateurs génétiques sont réappliqués. Si $g = g_{max}$ alors, l'optimisation est terminée, et l'individu trouvé représente la solution du problème de l'optimisation.

L'algorithme suivant résume toutes ces étapes :

Algorithme Génétique :

Entries : $F(x), p_c, p_m$ *et* μ

Sortie: c_i .

Variables : g *et* g_{max}

Début $g = 0$

Initialiser: $\boldsymbol{P_g = \{c_1, \dots, c_i, \dots, c_\mu\}}$

Tant que $g \leq g_{max}$ **faire**

Calculer la fitness : $F(c_i)$

Sélection:$P_g \rightarrow \tilde{P}$

croisement: $\tilde{P} \rightarrow \hat{P}$

Mutation: $\hat{P} \rightarrow P_{g+1}$

Fin tant que

Retour c_i

Fin

Conclusion:

Les algorithmes génétiques sont des méthodes d'optimisation très utiles dans le cas non linéaire (évidemment, cette méthode fonctionne également pour les cas linéaires, mais dans ce cas, est inutilement puisque elle est lourde en temps et en calcul). Cette technique part du principe évolutif de la sélection naturelle de Darwin. Celle-ci énonçait que les individus les plus aptes à survivre (les meilleurs) se reproduiront plus souvent et auront plus de descendants. Ainsi, la qualité d'enfant génétique de la population sera augmentée, les gènes plus efficaces deviendront plus fréquent; la population s'améliore. Selon le même principe, un algorithme génétique part d'une population de solutions initiales, les fait se reproduire (les meilleures solutions ont plus de chances de se reproduire), créant ainsi la nouvelle génération de solutions. En répétant ce cycle plusieurs fois, on obtient une population composée de solutions meilleures, l'application de cette technique sur la commande prédictive non linéaire peut porte une solution prometteuse.

Chapitre IV:

Applications et résultats des simulations

Introduction

Les méthodes utilisées jusqu'ici pour résoudre le problème d'optimisation résultant de commande prédictive non linéaire étaient basées sur des méthodes itératives. Ces méthodes sont, généralement, rapides mais la solution globale peut s'avérer difficile à atteindre de fait que le système à commander est non linéaire. L'utilisation de ces méthodes conduit à plusieurs optimums locaux, sachant que le calcul lourd et compliqué des matrice hessienne et jacobienne et parfois impossible. Et pour éviter toutes ces problèmes, on propose une autre alternative pour résoudre les problèmes de commande prédictive en basant sur l'utilisation des algorithmes génétiques présentés dans le troisième chapitre. Les algorithmes génétiques ne garantissent pas de trouver la solution optimale globale d'un problème, mais ils sont généralement capables de trouver des solutions acceptables dans des délais acceptables **[25].**

Dans ce quatrième chapitre, on va essayer d'appliquer les algorithmes génétiques pour obtenir la loi de commande prédictive sur des systèmes non linéaires modélisés par des modèles Neuro-flous en utilisant de l'environnement MATLAB 7.1.

IV.1 Commande prédictive non linéaire

Le principe de base d'une loi de commande prédictive comme il est montré précédemment: Un modèle sert à prédire les futures sorties du système, grâce aux valeurs courantes et passées de la commande et aux commandes optimales futures. Ces dernières sont calculées par une méthode d'optimisation, qui prend en compte une fonction de coût, laquelle dépend aussi des consignes futures, et éventuellement des contraintes. L'objectif de ce chapitre est de proposer une stratégie de commande prédictive non linéaire à base de modèle Neuroflou, dont le but est de décrire le comportement futur du système non linéaire. En premier lieu, le réseau Neuroflou est entraîné hors ligne, c'est-à-dire les paramètres du modèle sont pris à la fin de l'application de toutes les données des couples d'apprentissage, pour avoir un réseau fixe. Ensuite, celui-ci utilisé en ligne pour donner les prédictions de la sortie à chaque instant de temps. La fonction de coût de la commande prédictive est aussi optimisée en ligne à chaque itération en utilisant les algorithmes génétiques.

IV.1.1 Position de problème

Le problème de la CPNL sous contrainte est transformé en problème sans contraintes en utilisant la méthode des pénalités décrite dans le premier chapitre, en utilisant les algorithmes génétiques, on cherche à minimiser (I. 15) , alors :

$$J_{tot}(\theta) = \begin{cases} J(\theta) & si\ \theta \in C \\ \big(J(\theta) + P(\theta)\big) & si\ \theta \notin C \\ \text{avec : } \min J^{k}{}_{tot}(\theta) = \theta^{k-1} & si\ \theta\ n'existe\ pas \\ C: l'\ espacefaisable & \\ \theta = [u(k), \dots, u(k+N_u-1)]^T & \end{cases} \quad \text{(IV. 1)}$$

La valeur de pénalité $P(\theta)$ tend vers zéro, si un individu se trouve dans l'espace faisable C, c'est-à-dire que la solution ne vérifie pas toutes les contraintes. Sinon la fonction de pénalité $P()$ est alors ajoutée à la fonction objectif qui donne une fitness pénalisée. Si l'optimum n'existe pas, le vecteur optimal prend la valeur précédente de la commande θ^{k-1}.

En tenant en compte des contraintes en utilisant les fonctions de pénalité, les individus infaisables sont pénalisés par la diminution de la fitness. Donc, la fonction de la fitness dépend de deux fonctions, la fonction objectif $J(\theta)$ et celle de pénalité $P(\theta)$.

La fonction de coût $J(\theta)$, est une fonction de l'erreur quadratique entre les prédictions neuroflous de sorties du système y_{Nf} et les trajectoires de référence w, une pondération de l'erreur quadratique entre l'incrémentation de la commande et le modèle de référence de commande est existe.

Les paramètres de la commande sont :

Nu est l'horizon de commande,

*N*1 est l'horizon d'initialisation,

*N*p est l'horizon de prédiction,

λ est le facteur de pondération de la commande,

On sait que :

$$y_{NF_{i+1/k}} = \hat{f}_{NF}(y_{i/k}, u_{i/k}), y_{0/k} = y_k \qquad (IV.2)$$

$$y_{NF_{i/k}} \in \mathbb{Y}, i \in [1, N_p] \qquad (IV.3)$$

$$u_{i/k} \in \mathbb{U}, i \in [1, N_u], et\ \forall i \geq N_c\ u_{i/k} = u_{N_c/k} \qquad (IV.4)$$

Les variables représentées avec un double indice séparé par une ligne horizontale sont les variables internes de prédiction neurofloue. Le second argument représente l'instant à partir duquel la prédiction est calculée. $y_{0/\,k} = y_k$ est l'état initial du système à contrôler à partir de l'instant *k*, $y_{NF_{i|k}}$ et $u_{i|k}$ sont respectivement la sortie prédite par le système neuroflou et la commande depuis *k* avec *i* indiquant l'instant de prédiction à partir de *k*.

La commande prédictive consiste à calculer le vecteur θ sur l'horizon de prédiction N_p et à appliquer le premier élément u_0 de cette séquence. Cette opération est ensuite effectuée à nouveau à l'instant suivant en faisant glisser l'horizon de prédiction de $[k: k+1]$ à $[k+1: k+2]$.

IV.1.2 Solution du problème via les AG

À chaque itération, appelée génération, est créée une nouvelle population avec le même nombre de chromosomes. Cette génération consiste en des chromosomes mieux "adaptés" à leur environnement tel qu'il est représenté par la fonction objectif (IV.1). Au fur et à mesure des générations, les chromosomes vont tendre vers l'optimum de la fonction objetif $J_{tot}(\theta)$. La création d'une nouvelle population à partir de la précédente se fait par application des opérateurs génétiques: la sélection, le croisement et la mutation, comme il est illustré sur la figure suivante.

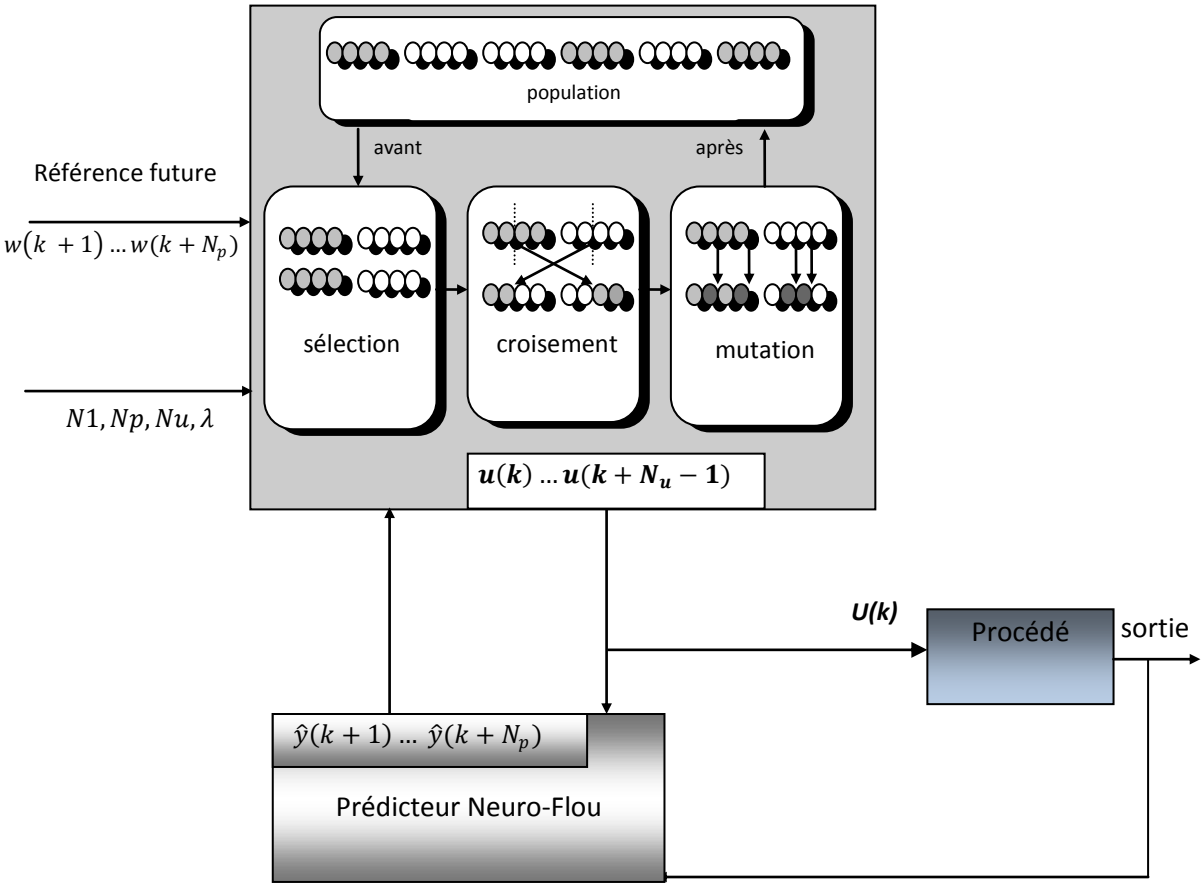


Figure IV.1 : Structure de la commande prédictive via les algorithmes génétiques

L'application de l'AG à la résolution du problème de la commande prédictive, nécessite de coder les solutions afin de constituer les chromosomes, et de trouver une fonction d'évaluation permettant une bonne discrimination entre les chromosomes et de définir les opérateurs génétiques qui seront utilisés.

L'algorithme de résolution commence avec la création d'une population P représentée par un vecteur de commande de taille égale à N constitué des individus. Ensuite, on mesure l'adaptation de chacun des individus en calculant la fonction de fitness de la commande prédictive $J_{tot}(\theta)$. La prochaine étape consiste à faire évoluer cette population vers une population plus adaptée à chaque génération en utilisant les trois différents opérateurs génétiques. Lorsque nous n'avons plus d'amélioration dans l'adaptation des individus de la population, l'algorithme donne le vecteur optimal de la commande θ, on applique seulement la première valeur u(k) et on répète l'algorithme pour l'instant suivant.

En utilisant les trois opérateurs que nous venons de décrire, les meilleurs individus se propagent de génération en génération en se combinant ou en échangeant leurs meilleurs caractéristiques. En favorisant les meilleurs individus, les régions les plus prometteuses de l'espace de recherche des commandes admissibles sont explorées, ce qui permet d'atteindre un optimum global.

Algorithme :

L'objectif souhaité est de trouver la commande $\theta = [u(k)\ u(k+1) \dots u(k+Nu-1)]^T$ qui minimise la fitness $J(\theta)$ pour chaque itération k. Par conséquent la topologie du chromosome, dans le cas général, est le suivant :

$u(k)$	$u(k+1)$	...	$u(k+N_u-1)$

Les paramètres $u(k)\ u(k+1) \dots u(k+N_u-1)$ peuvent être codés en binaire ou en utilisan, un codage réel. Après que le modèle neuroflou est entraîné hors ligne, à chaque instant k, l'algorithme de commande est déroulé selon les étapes suivantes:

- **Etape 0:**donner les paramètres de la commande $N1, Np, Nu, \lambda, et\ la\ reference$ à $suire : w(k), \dots, w(k+Np)$.
- **Etape 1 :** Générer la population de façon aléatoire.
- **Etape 2 :** Décoder chaque élément de la population.
- **Etape 3 :** utiliser le prédicteur neuroflou (IV.5) pour trouver les prédictions futures $y_{nf} = [y(k+1), \dots y(k+Np)]$.
- **Etape 4 :** Évaluer la population initiale en calculant la fonction objectif (IV.1) associée à chaque chromosome de la population.
- **Etape 5 :** Tant que la précision voulue ou le nombre maximal de génération ne sont pas atteints, répéter les opérations suivantes :

✓ Assigner une valeur dite Fitness à l'ensemble de la population.
✓ Sélection des individus pour la reproduction.
✓ Croisement des individus avec une probabilité de croisement.
✓ Mutation des individus avec une probabilité de mutation.
✓ calculer les prédictions de façon itérative.
✓ Évaluation des individus sélectionnés.
✓ Réinsertion dans la population courante.

- **Etape 6 :** donner le vecteur de la commande optimale $\theta_{opt} = [u(k)\ u(k+1 \dots u(k+Nu-1)T$.
- **Etape7:**Selon la stratégie de commande prédictive, la première commande u(k) est appliquée au procédé, tandis que le vecteur θ_{opt} est utilisé comme entrée pour le prédicteur neuroflou et le calcul se répète.
- **Etape 8 :** passer à l'étape 1.

IV.2 Simulations

L'entrée appliquée sur le procédé à commander pour construire notre prédicteur est un signal pseudo aléatoire SBPA (figure IV.2**)** qui est caractérisé par :

- Variable avec le temps.
- riche en fréquence, au sens de l'identification.
- L'amplitude varie selon les caractéristiques du système à modéliser.

Le modèle NF est constitué de cinq couches, les fonctions d'activation choisies sont de type sigmoïde.

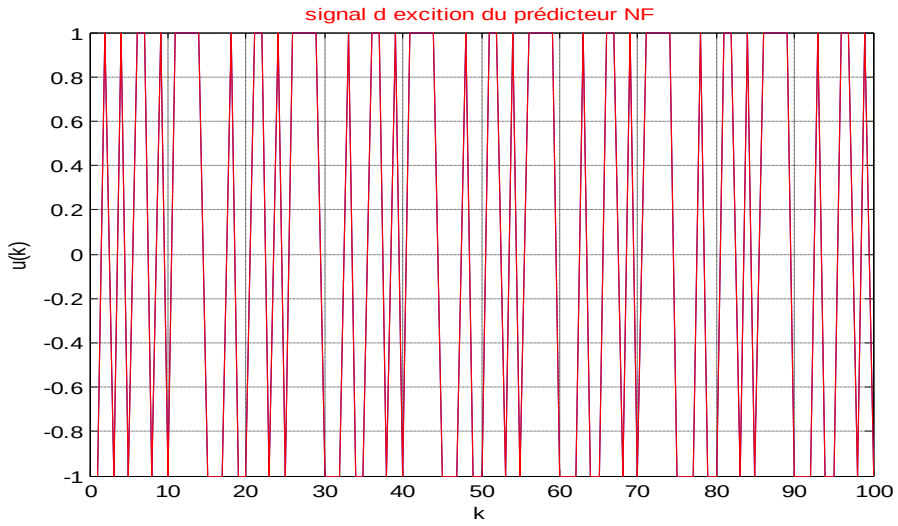


Figure IV.2 : signal pseudo aléatoire SBPA.

IV.2.1 Exemple 1

On considère le système non linéaire discret donné par l'équation suivante :

$$y(k+1) = 0.9 * y(k) * u(k) + u(k) \qquad (IV.5)$$

Un modèle neuroflou est obtenu en utilisant un couple d'entrées /sorties généré avec $u(k) \in [-1, +1]$ est un signal SBPA.les entrées du modèle NF sont $u(k), y(k)$ et la sortie est $y(k+1)$, en utilisant les algorithmes génétique pour la phase d'apprentissage après un nombre d'itération égale à N=2000, on obtient un critère $J = 0.0116$, le nombre d'échantillons simulées égale à Nb=100 ,la figure suivante montre la sortie du modèle neuroflou et la sortie du système à modéliser:

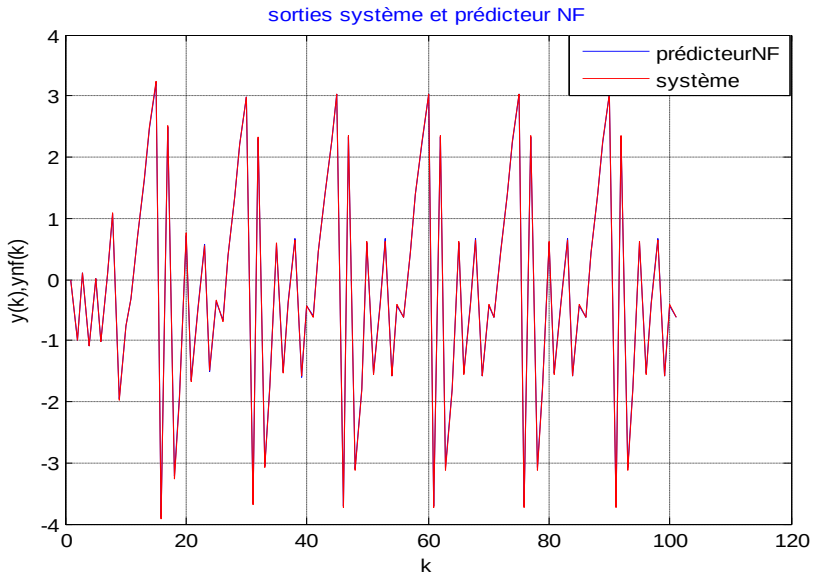


Figure IV.3: Réponses du système neuroflou et système non linéaire à modéliser (exemple1).

On a appliqué la commande prédictive à base du modèle neuroflou en utilisant les algorithmes génétiques pour calculer la séquence de commande optimale, sachant que le système à commander est sous les contraintes suivantes:

$$0 \leq u \leq 1$$

$$-0.5 \leq \Delta u \leq 0.5$$

$$0 \leq y \leq 3.5$$

$$-1 \leq \Delta y \leq 1$$

La référence à suivre est égale à:

$$w(k) = \begin{cases} 3 & pour\ 0 \leq k \leq 50 \\ 1 & pour\ 50 \leq k \leq 110 \\ 2 & pour\ 110 \leq k \leq 160 \end{cases}$$

Les paramètres de simulation sont: $N1 = 1, Np = 3, Nu = 1\ et\ \lambda = 0$, on obtient les figures suivantes :

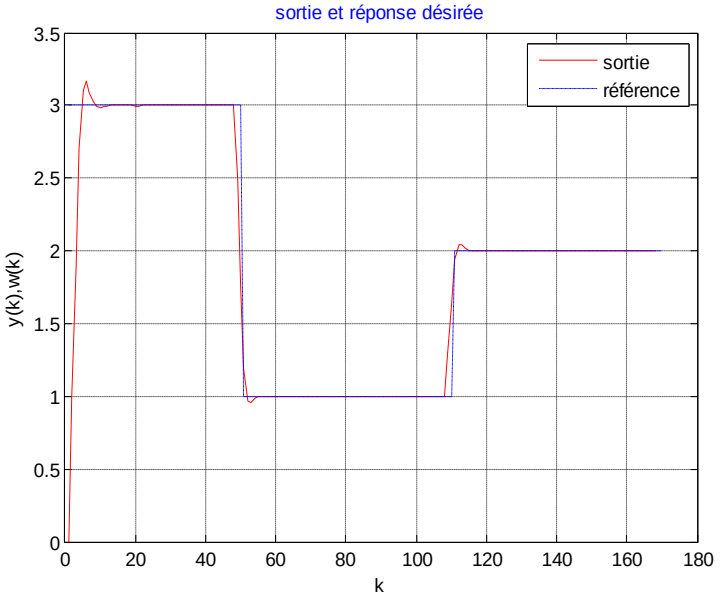


Figure IV.4 : Sorties du système non linéaire et sa référence à suivre (exemple 1).

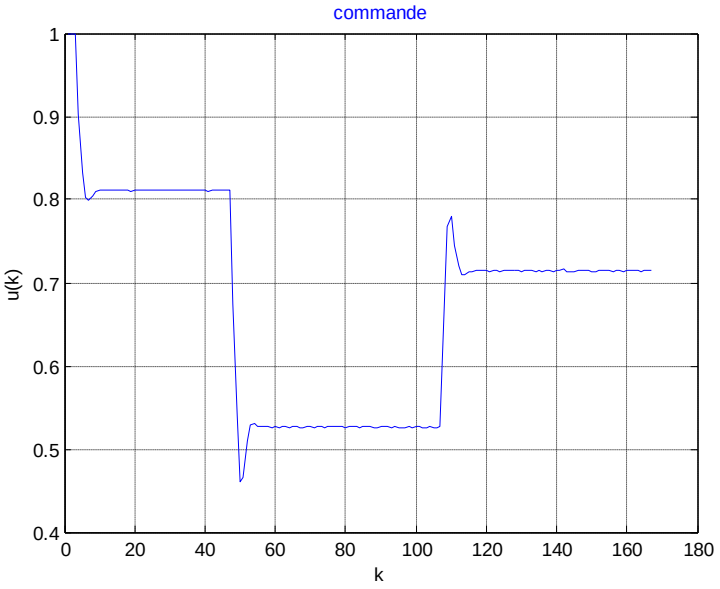


Figure IV.5: séquence de la commande (exemple 1).

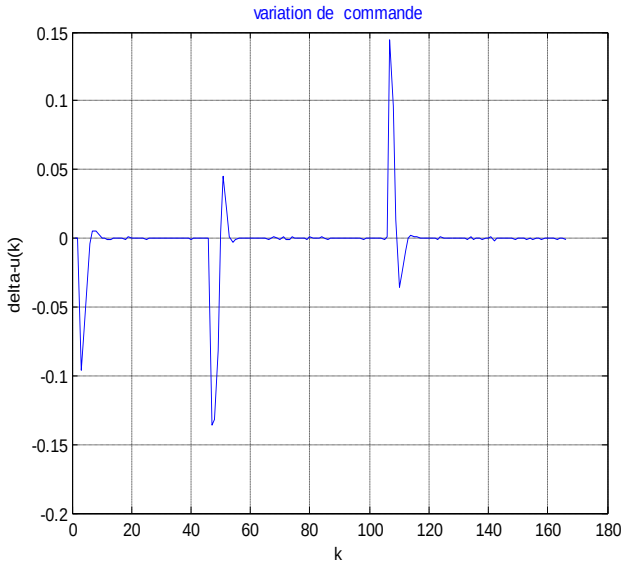


Figure IV.6: Variation de la commande (exemple 1).

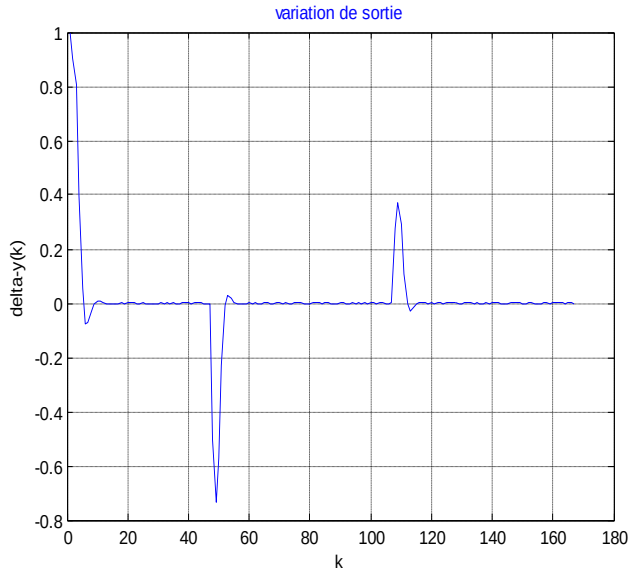


Figure IV.7: Variation de la sortie (exemple 1).

IV.2.2 Exemple 2

Soit les deux systèmes non linéaires discrets donnés par les équations suivantes :

- ***Premier système :***

$$y(k+2) = y(k) + u(k) * exp(-3 * |y(k)|) \qquad (IV.6)$$

Sous les contraintes suivantes:

$$-10 \leq u \leq 10$$

$$-2 \leq \Delta u \leq 2$$

$$0 \leq y \leq 2$$

$$-0.50 \leq \Delta y \leq 0.5$$

La référence à suivre donnée par :

$$w(k) = \begin{cases} 1.5 & pour\ 0 \leq k \leq 50 \\ 0.5 & pour\ 50 \leq k \leq 110 \\ 1 & pour\ 100 \leq k \leq 150 \end{cases}$$

- ***Deuxieme système :***

$$y(k+1) = y(k)^3 - 0.2 * |y(k)| * u(k) + 0.08 * u(k)^2 \qquad (IV.7)$$

Sous les contraintes suivantes:

$$0 \leq u \leq 4$$

$$-0.1 \leq \Delta u \leq 0.1$$

$$0 \leq y \leq 1$$

$$-0.1 \leq \Delta y \leq 0.1$$

La référence à suivre est égale à:

$$w(k) = \begin{cases} 0.7 & pour\ 0 \leq k \leq 50 \\ 0.4 & pour\ 50 \leq k \leq 110 \\ 0.6 & pour\ 110 \leq k \leq 170 \end{cases}$$

Les deux systèmes sont décris dans [26] .Les modèles neuroflous sont obtenus en utilisant des couples d'entrées /sorties générés par SBPA. Chaque modèle est constitué de cinq couches, la fonction d'activation choisie est de type sigmoïde, les entrées sont $u(k), y(k+1)$ pour le premier système et $u()\ , y(k)$ pour le deuxième, les sorties sont $y(k+2)\ , y(k+1)$ pour le premier et le deuxième respectivement.

Premièrement on fait un apprentissage des prédicteurs neuroflous et après un nombre d'itération égale à N=2000, les critère optimaux résultants des premier et

deuxième système sont : $J = 0.4261$ et $J = 0.0237$ respectivement, les figures suivantes montrent les sorties des modèles neuroflous avec les sorties réelles:

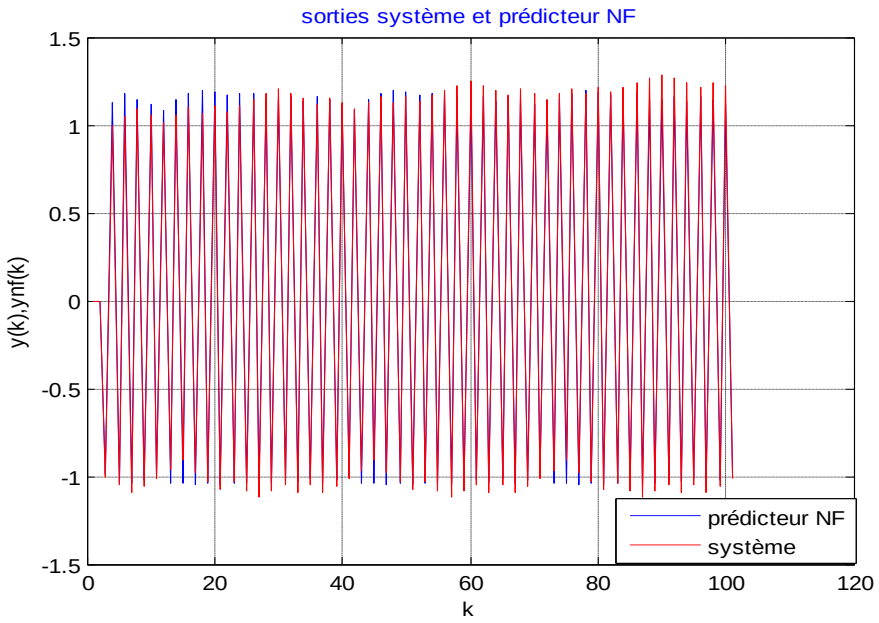


Figure IV.8: Réponses du système neuroflou et système non linéaire1.

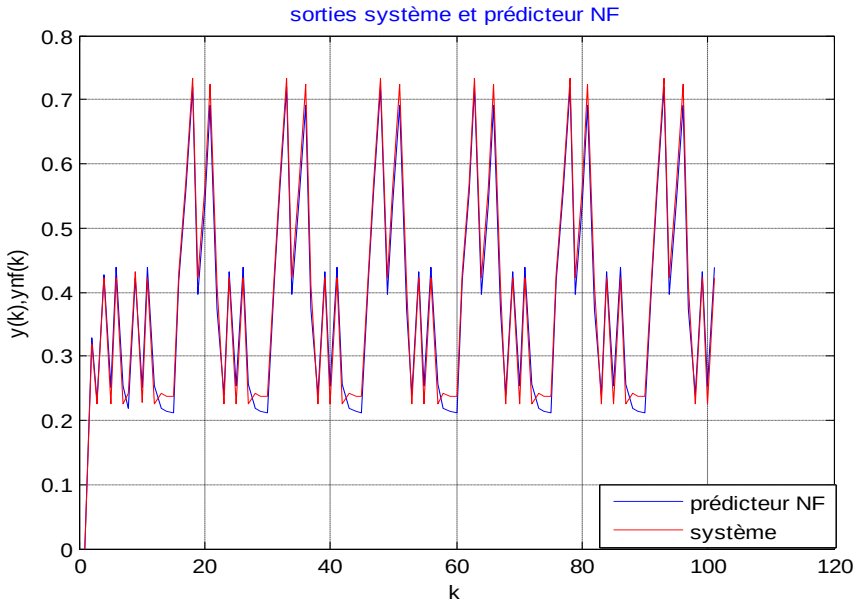


Figure IV.9: Réponse du système neuroflou et système non linéaire2

En appliquant la commande prédictive à base du modèle neuroflou et en utilisant les algorithmes génétiques, avec les paramètres de simulation:$N1 = 1, Np = 4, Nu = 1\ et\ \lambda = 0$ sur les deux systèmes non linéaires sous contraintes, les résultats de simulation étant donnés par les figures suivantes:

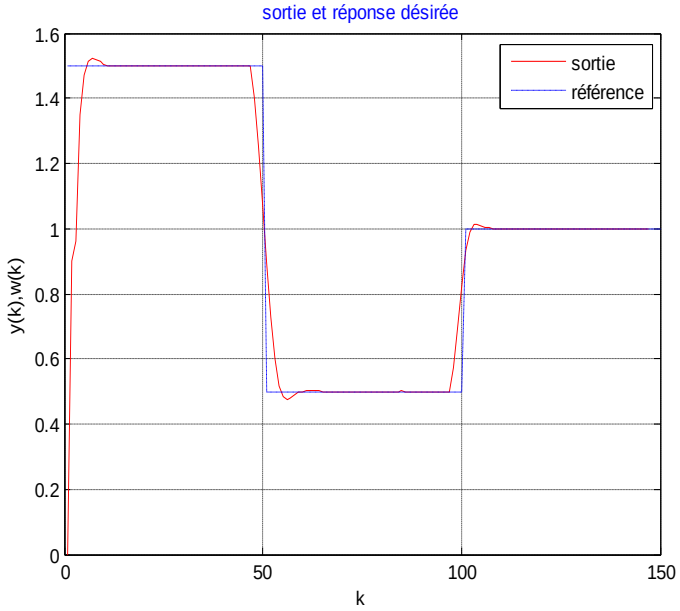


Figure IV.10: Sortie du système non linéaire1et référence.

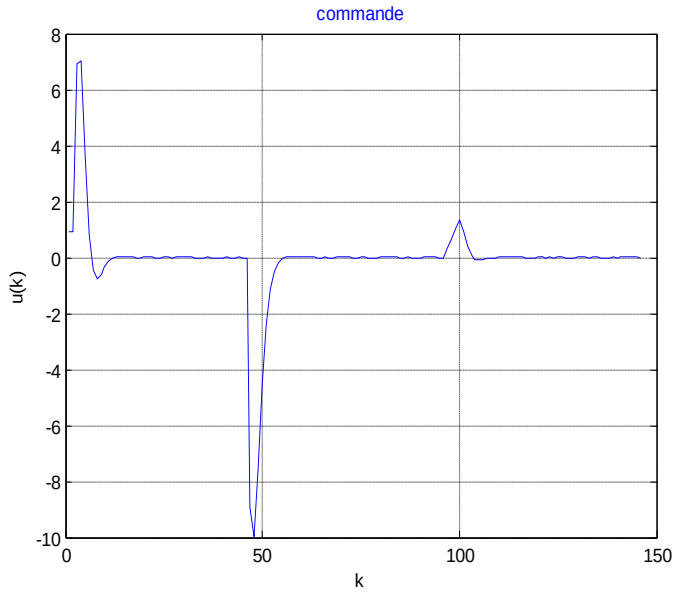


Figure IV.11: séquence de la commande de système non linéaire 1.

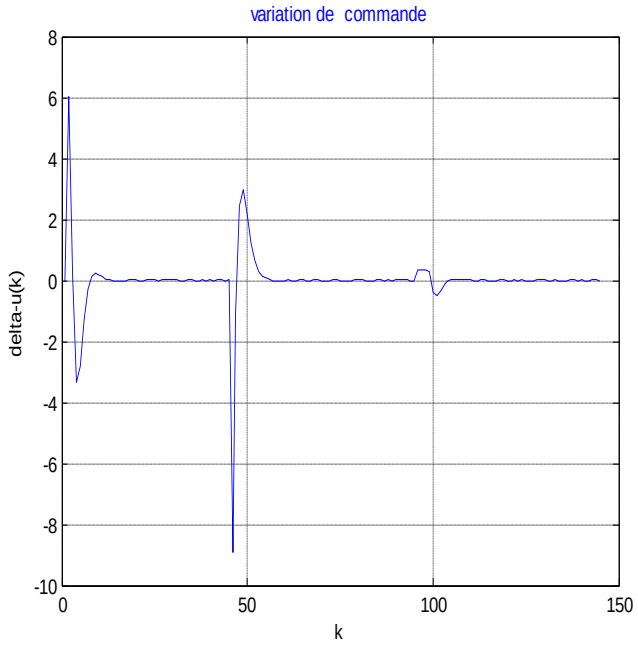


Figure IV.12: Variation de la commande système non linéaire 1.

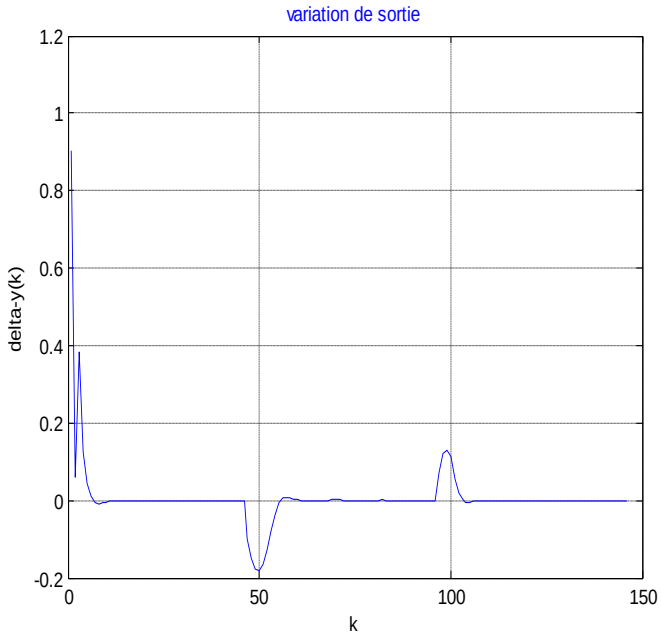


Figure IV.13: Variation de la sortie du système non linéaire 1.

En appliquant notre approche de la commande prédictive avec les paramètres de simulation: $N1 = 1, Np = 3, Nu = 1\ et\ \lambda = 0.8$ sur le deuxième système non linéaire sous contraintes, les résultats de simulation étant donné par les figures suivantes:

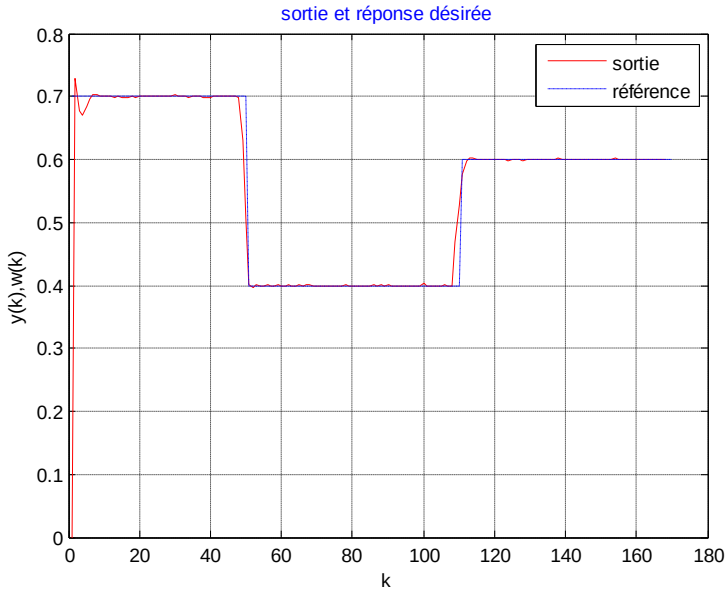


Figure IV.14 : Sortie du système non linéaire2 et référence à suivre.

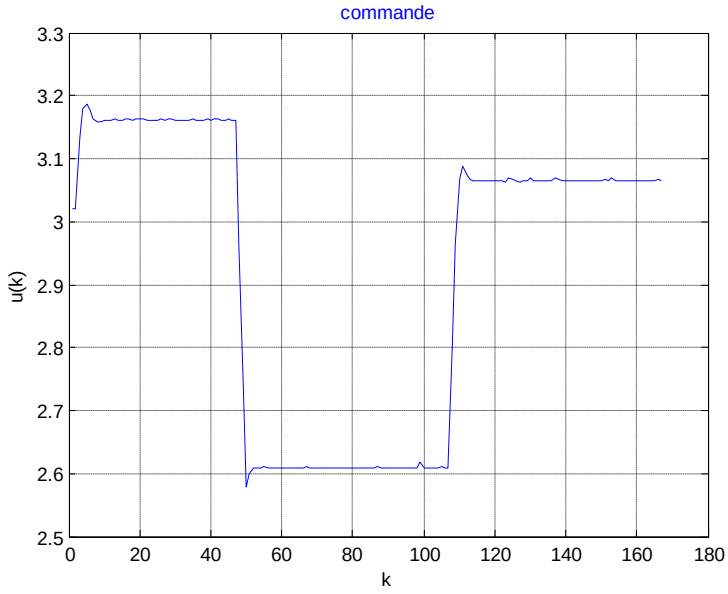


Figure IV.15: Séquence de la commande du système non linéaire 2.

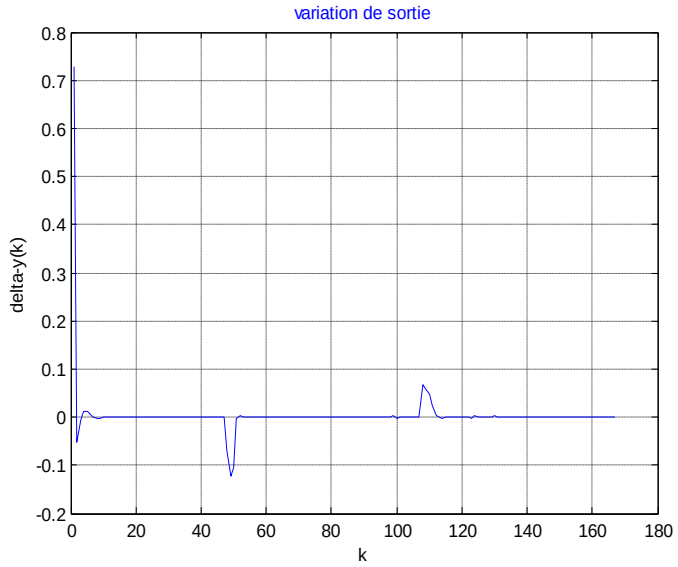


Figure IV.16: Variation de la sortie du système non linéaire 2.

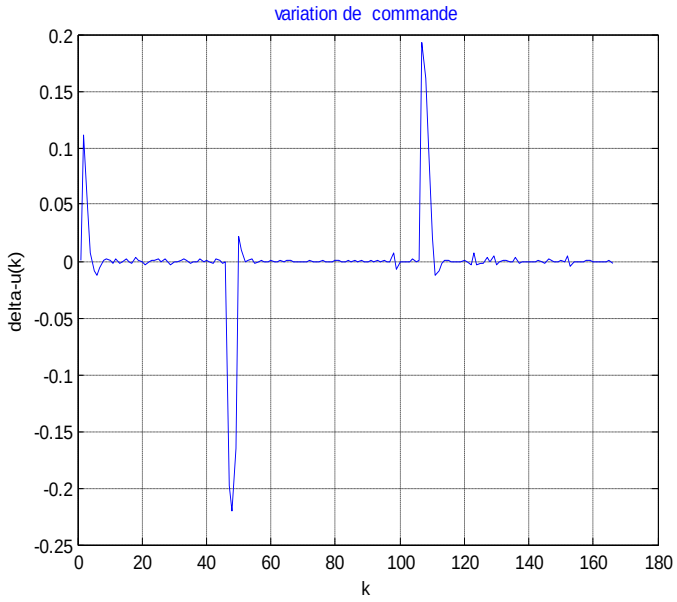


Figure IV.17: Variation de la commande du système non linéaire 2.

IV.2.3 Exemple 3 (Commande de température du système ‘ bain d'eau’)

Le but de cet exemple cité dans [27] est de démontrer l’efficacité de l’application de la commande prédictive neuroflou via les algorithmes génétiques sur

un système non linéaire réel, l'objectif est de forcer la température dans le bain à suivre la référence suivante :

$$w(k) = \begin{cases} 35^0C, & for\ k \leq 40 \\ 55^0C, for\ 40 < k \leq 80 \\ 75^0C, for\ 80 < k \leq 120. \end{cases}$$

La variation de température est donnée par le système discrétisé (IV. 9) qui est récris sous la forme suivante avec une certaine approximation :

$$y(k+1) = e^{-\alpha T_s} y(k) + \frac{\frac{\delta}{\alpha}(1-e^{-\alpha T_s})}{1+e^{0.5y(k)-40}} u(k) + [1 - e^{-\alpha T_s}]y_0 \qquad (IV. 9)$$

Où :

La sortie y (t) : est la température du bain.

L'entrée u (t) : est le flux de la chaleur que doit évacuée dans le bain.

Y_0 : est la température ambiante.

C : est la capacité thermique équivalente du système.

T_R : est la résistance thermique équivalente entre les parois du bain et l'air ambiant.

TR et C sont supposés constants,

Où α et δ sont des Constants. Les paramètres de système utilisés dans cet exemple sont α =1.0015e^{-4}, le δ =8.67973e^{-3} et les Y_0=25.0 (C_0), La limite inférieure d'entrée $u(k)$ est 0, avec 5V représente l'unité de tension de la commande. La période d'échantillonnage est T_s=30s.

le modèle neuroflou est obtenu en utilisant un couple d'entrées /sorties générés par l'application d'un signal pseudo-aléatoire compris entre $[-5, +5]$ .le modèle est constitué de cinq couches, la fonction d'activation choisie est de type sigmoïde, les

entrées sont $u(k), y(k)$ et la sortie $y(k+1)$ en utilisant les algorithmes génétique pour la phase d'apprentissage après un nombre d'itération égale à N=2000 ,on obtient un critère J=0.1874,le nombre d'échantillons simulées égale à Nb=100 ,la figure suivante montre la sortie du modèle neuroflou et celle du système à modéliser:

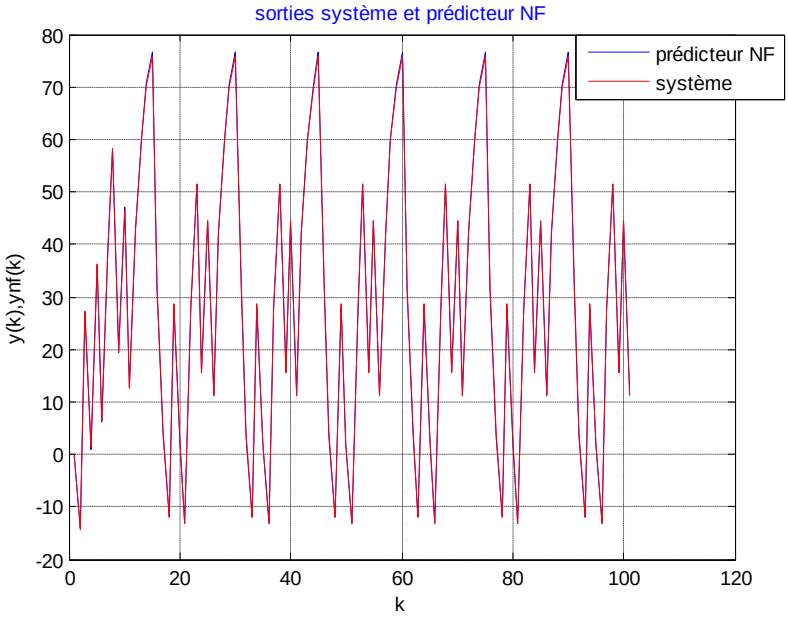


Figure IV.18: Réponses du système neuroflou et le système récurent de bain.

Les résultats de simulation sont donnés pour la référence suivante :

$$w(k) = \begin{cases} 35^0 & pour\ k \leq 40 \\ 55^0 & pour\ 40 \leq k \leq 80 \\ 75^0 & pour\ 80 \leq k \leq 120 \end{cases}$$

Sous les contraintes suivantes :

$$\begin{cases} 0 \leq u \leq 5 \\ -0.5 \leq \Delta u \leq 0.5 \\ 30 \leq y \leq 80 \\ -5 \leq \Delta y \leq 5 \end{cases}$$

Les paramètres de commande prédictive choisis sont $Np = 4, Nu = 1\ et\ \lambda = 0.5$ on obtient :

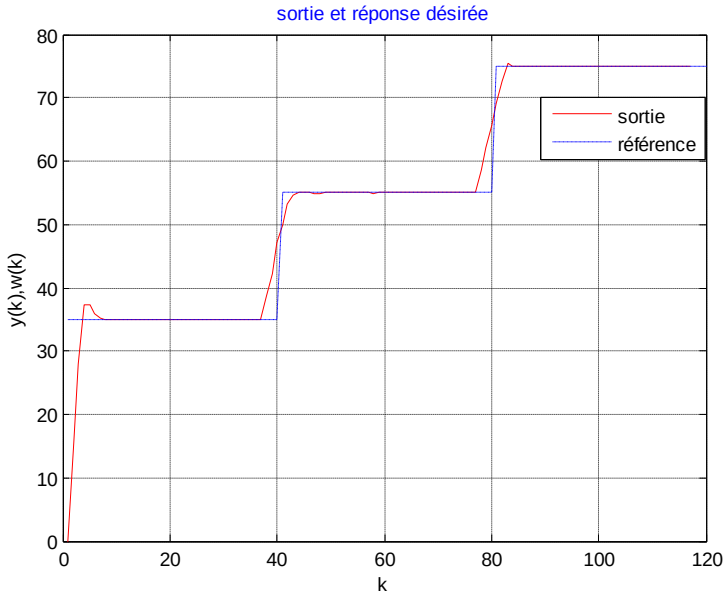


Figure IV.19: Réponses du système de bain et sa référence.

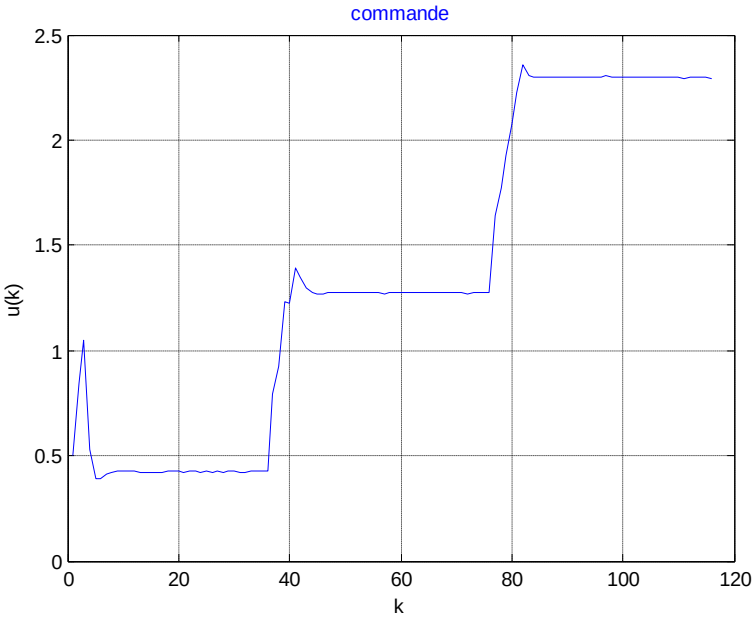


Figure IV.20: Séquence de commande (exemple 3).

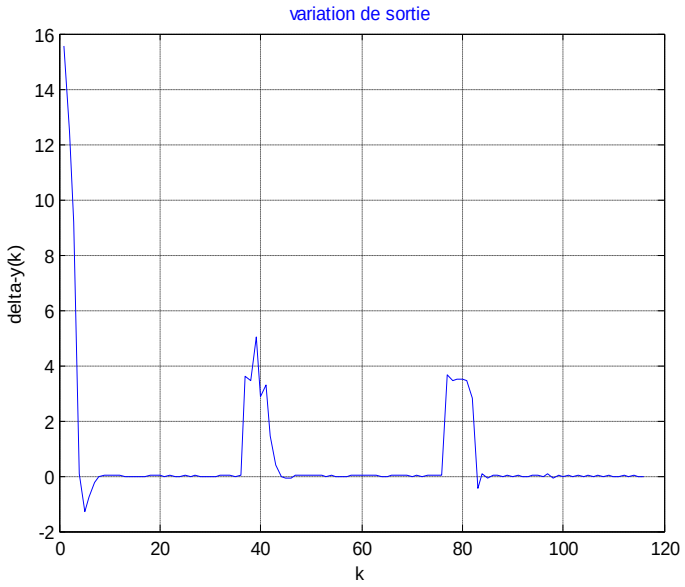


Figure IV.21: Variation de la sortie (exemple 3).

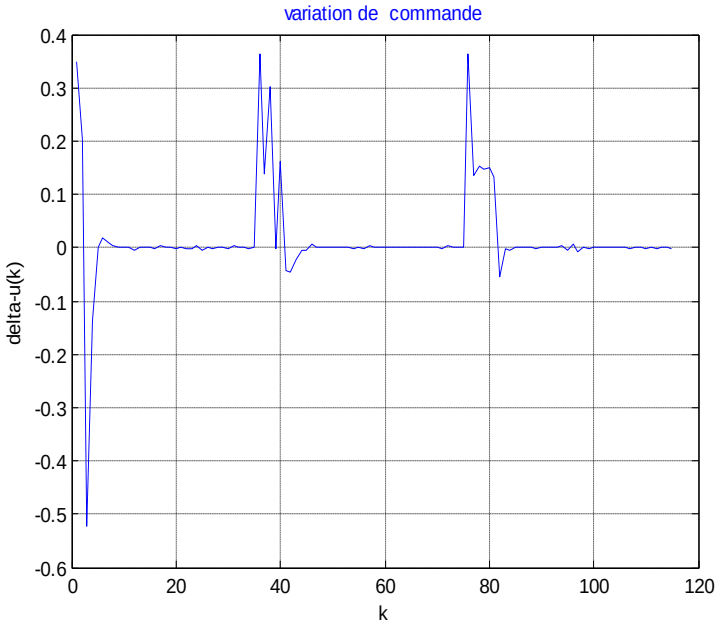


Figure IV.22: Variation de la commande (exemple 3).

IV.3 Discussion des résultats des simulations

L'horizon de commande est en général inférieur à l'horizon de prédiction, c'est-à-dire que le dernier élément de la séquence de commande est maintenu constant entre l'horizon de commande et l'horizon de prédiction, à la valeur obtenue au terme de

l'horizon de commande. Quelle que soit la valeur de l'horizon de commande, la séquence de commande optimale est toujours optimisée sur la totalité de l'horizon de prédiction.

L'intégration dans la boucle de commande d'un prédicteur neuroflou, réalisé à partir d'une combinaison de mesures entrées/sorties, permet de donner une prédiction satisfaisante du comportement du système contrôlé. La performance de ce prédicteur dépend d'horizon de prédiction si le l'horizon est grand les performances du prédicteur sont dégradées. Le couplage de la commande non-linéaire prédictive avec un modèle neuroflou est réalisé avec des bonnes performances avec une stabilité globale du système complet (procédé + commande + prédiction).La simulation numérique en utilisant le logiciel de MATLAB a permis la mise en œuvre de la commande non linéaire prédictive avec des bonnes performances dans un environnement qui simule la réalité par l'utilisation du sortie passée du procédé comme entrée du prédicteur en faisant à chaque itération une optimisation via les algorithmes génétique (simulation online).

Les simulations ont permis de valider notre approche en termes de convergence et de respect des contraintes et de stabilité du prédicteur. L'architecture génère une commande qui force le système à suivre la référence en respectant les contraintes.

Les performances de la commande prédictive sont étroitement liées à la longueur de l'horizon de prédiction. Dans le cas linéaire, il existe des méthodes systématiques pour fixer les horizons de prédiction et de commande. Dans le cas non linéaire, le problème reste ouvert puisque l'optimalité d'un horizon est fortement liée à la complexité du problème. Ainsi, il est clair que l'utilisation d'un horizon de prédiction court permet de réduire les coûts en temps de calcul nécessaire pour que l'algorithme génétique converge vers une solution optimale, mais un horizon de prédiction long peut provoquer l'instabilité en boucle fermée à cause des prédictions moins précises du prédicteur neuroflou.

Conclusion

L'optimisation contrairement au cas linéaire où le problème est convexe, la CPNL nécessite la résolution d'un problème non linéaire non convexe. Nous proposons une discussion du problème d'optimisation en utilisant les algorithmes génétiques. Cette méthode permet de se rapprocher de la solution optimale en calculant à chaque itération la fonction fitness des individus .la méthode de pénalités est utilisé pour intégrer les contraintes du problème dans la fonction à optimiser. Pour une utilisation en temps réel, l'optimisation en utilisant les algorithmes génétiques se contenter d'une solution sous optimale, à partir du moment où la solution obtenue en interrompant l'algorithme d'optimisation avant convergence (pour un nombre d'itérations fixé) est garantie faisable (solution respectant les contraintes associées au problème).les résultats de simulation montre les performances de l'application de l'approche proposée sur des systèmes non linéaires ,mais avec un temps de calcul un peut lourd, et cela nécessite des calculateurs très rapides.

Conclusion générale et perspectives

Conclusion générale et perspectives :

L'objectif de ce travail a été de voir les possibilités d'extension de la stratégie de commande prédictive à des systèmes non linéaires décrits par des systèmes neuro-flous. Nous avons décrit une nouvelle architecture de commande non linéaire. L'approche proposée repose sur l'utilisation de la commande prédictive. Cette structure utilise un modèle neuro-flou du processus pour prédire le comportement futur du système contrôlé afin de déterminer une commande optimale. Ainsi, cette architecture permet d'assurer le respect des contraintes qui peuvent intervenir dans une commande prédictives : saturation des actionneurs, des considérations économiques ou des limitations propres aux capteurs, imposent des limites sur les variables du processus à commander.

Le modèle de prédiction est conçu à partir d'un réseau neuro-flou avec des paramètres fixes après un apprentissage hors ligne. Cette modélisation non linéaire ne nécessite pas une connaissance du modèle du procédé, elle a seulement besoin d'informations sur les entrées/sorties du procédé. Le problème de la CPNL sous contrainte est transformé en problème sans contraintes en utilisant la méthode des pénalités décrite dans le chapitre trois, en utilisant les algorithmes génétiques, on cherche à minimiser la fonction de fitness. La charge de calcul pour la tâche d'optimisation dépend du nombre d'itérations nécessaire pour trouver la commande optimale à chaque instant de temps et aux valeurs des horizons. Un apprentissage en ligne du système neuro-flou peut être pris en considération pour avoir une commande adaptative. Cependant, cette procédure entraîne une augmentation de la charge de calcul de la commande et ne donne pas nécessairement les performances souhaitées. Par conséquent, l'outil pour une implantation en temps réel doit satisfaire les contraintes de la charge de calcul. Il est clair que notre étude, sur l'application de la commande prédictive neuro-flou au système non linéaire, aboutit à des résultats satisfaisants. La simulation numérique a permis la mise en œuvre de la commande non linéaire prédictive avec des bonnes performances dans un environnement proche de la réalité.

Si L'optimisation de fonction objectif est réalisée en ligne le temps d'exécution est devient lourd, car à chaque instant de temps, la commande calculée est le résultat d'une optimisation génétique ainsi que le calcul des échantillons futurs donnés par le prédicteur neuro-flou.

Comme perspective, il sera intéressant de prise en compte directe de l'erreur de prédiction dans la CPNL : l'erreur de prédiction est considérée comme une perturbation dans la conception du contrôleur et l'objectif est alors de rejeter cette perturbation, ou l'utilisation d'un prédicteur neuro-flou adaptatif, formulé comme une optimisation en ligne et nommé prédicteur à horizon glissant. Cette approche se ramène donc à la conception d'une CPNL robuste. Cependant la commande sera très coûteuse en temps de calcul car il est alors nécessaire de réaliser deux optimisations en ligne.

Annexe

Liste des variantes de la commande prédictive :

Toutes les variantes de stratégies de commande prédictive sont aujourd'hui regroupées sous le terme générique MPC, La liste ci-dessous propose un aperçu des plus « classiques » :

• **MPHC** (Model Predictive Heuristic Control), connue ensuite sous le nom de MAC (Model Algorithmic Control) Cette approche, appliquée aux systèmes industriels multivariables, basée sur des prédictions sur un horizon temporel long, impose des trajectoires de référence pour les sorties et minimise la variance de l'erreur.

• **DMC** (Dynamic Matrix Control) proposée par Shell utilise l'incrément de commande à la place de la commande dans le critère de performance pour un horizon fini de prédiction; cet algorithme est appliqué à des systèmes multivariables linéaires sans contraintes; l'erreur de poursuite est minimisée en spécifiant le comportement futur des sorties ; les commandes optimales sont calculées par la méthode des moindres carrés.

• **EHAC** (Extended Horizon Adaptive Control), stratégie de commande prédictive pour les systèmes monovariables, utilise des modèles E/S pour maintenir la sortie future (calculée via la résolution d'une équation diophantienne) le plus près possible de la consigne pendant une période donnée au-delà du retard pur du système.

• **EPSAC** (Extended Prediction Self-Adapted Control) introduit une commande constante pour un système non-linéaire (en linéarisant le système) et utilise un prédicteur sous-optimal à la place de la résolution de l'équation diophantienne .

• **GPC** (Generalized Predictive Control) Cette méthode la plus connue, basée sur un modèle de type CARIMA, introduit un horizon de prédiction sur la commande, agit conformément au principe de l'horizon fuyant et peut être appliquée aux systèmes à non minimum de phase, aux systèmes instables en boucle ouverte, aux systèmes avec retards purs variables.

• **PFC** (Predictive Functional Control) est, un algorithme prédictif simple, utilisé surtout pour des systèmes SISO industriels rapides, en permettant le réglage direct des paramètres (par exemple la constante de temps) associées au temps de monté; pour

garder la simplicité, un manque de rigueur en performance et surtout dans la garantie des contraints est associé à cet algorithme.

• **CRHPC** (Constrained Receding Horizon Predictive Control) propose de prendre en compte des contraints terminales sous forme « égalité » sur la sortie sur un horizon fini au-delà de l'horizon de prédiction.

• **MPC** (Model Predictive Control) formulée dans l'espace d'état par utilise le formalisme de la représentation d'état pour faciliter l'analyse de la stabilité et de la robustesse.

Références bibliographiques

Références bibliographiques:

[1]: E. S. Camacho, C. Bordons, “Model predictive control in process industry”, Springer, London, U.K. 1995.

[2]: M. Morari, J. H. Lee, “Model predictive control: past, present and future”, Computers and chemical engineering, 1999.

[3]: D. E. Goldberg, “Genetic algorithms in search, optimization and machine learning”, Addison Wesley, Reading, MA, 1989.

[4]: J. A. K. Suykens, B. L. R. Vandewalle,B. L. R. De Moor, “Artificial Neural Networks for Model and Control of Nonlinear Systems”, Kluwer Academic Publishers, Boston 1995.

[5]: J. Sjoberg, Q. Zhang, L. Ljung, A.Benveniste, “Non-linear black-box modeling in system identification: A unified approach”, Automatica, 31(1995) pp 1691-1724.

[6]: M. Schoenauer, S. Xanthakis, “Constrained GA optimisation”, proceedings of the 5th ICGA, Morgan Kaufman, 1993.

[7]: Y. Tan, A. R.Van Cauwenberghe, “Direct and Indirect Optimizing Neural Predictive Control”, October 1996

[8]: M. Boumehraz, K. Benmahammed, "Constrained Non-linear Model Based Predictive Control using Genetic Algorithms"

[9]: Ernesto Granado Migliore, "Commande Prédictive à base de programmation semi définie".

[10]: Gérard Ramond, "Contribution à la commande prédictive généralisée adaptative directe et applications", le grade de docteur en sciences de l'université Paris Xiorsay, septembre 2001.

[11]: Mémoire fin d'étude,"La Robustification de la loi de commande prédictive généralisée Sous contraintes", pour grade ingénieur, 2005

[12]: J. J. Espinosa, M. L. Hadjili, V. Werts, "Predictive Control Using Fuzzy Model Comparative Study", European Control Conference, Karlshur, Germany, 1999.

[13]: G.Lavielle- P.Brochot- A.Muriana - R.Espigol- J.J.Tanguy, "Modélisation et commande prédictive d'un générateur de vapeur implantées dans un automate", 2002

[14]: BOO, HONG, AMY, KHAIRIYAH, "formulation of model predictive control algorithm for nonlineair processes ", 2006.

[15]:Cristina Nicoletta, " Robustification de lois de commande prédictives multivariables",2008

[16]:Ghizlane HAFIDI , "Application de la commande prédictive non-linéaire à la commande de culture de bactéries *Escherichia coli*",2008

[17]: Pascal Dufour1, "Contribution à la commande prédictive des systèmes à paramètres répartis non linéaires", 2000.

[18]: Ernesto ,Granado Migliore, "Commande prédictive à base de programmation semi définie ", 2004.

[19]:Fadi Ibrahim, "Commande prédictive non linéaire d'un lit mobile simulé", 2006.

[20]:Mickael SAUVEE, "Contribution à l'aide aux gestes pour la chirurgie cardiaque à cœur battant. Guidage échographique par asservissement prédictif non linéaire",2006

[21]:Ján Mikleš · Miroslav Fikar , "Process Modelling, Identification, and Control" , springer 2007.

[22]:Giovanna Castellano, "A Neurofuzzy Methodology for Predictive Modeling", 2000

[23]:Otilia Elina VASILE, "Contribution au pronostic de défaillance par réseau neuro-flou",2008.

[24]:Nurnberger, D. Nauck, R. Kruse, "Neuro-fuzzy control based on the NEFCON-model: recent developments".

[25]: Nicolas PALLUAT, "Méthodologie de surveillance dynamique à l'aide des réseaux neuro-fous temporels",2006.

[26]:Brian Roffel , "Process Dynamics and Control", springer 2007 .

[27]: Wen Yu, "recent advances in Intelligent Control Systems ", springer 2009.