Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus

Yacine Oussar

THÈSE de DOCTORAT de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité :

ROBOTIQUE

présentée

par **Yacine OUSSAR**

pour obtenir le titre de **DOCTEUR de l'UNIVERSITÉ PARIS VI**

Sujet de la thèse :

Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus.

Soutenue le 06 Juillet 1998

devant le jury suivant :

Mme	S. THIRIA	Rapporteur
M.	S. CANU	Rapporteur
M.	G. DREYFUS	Examinateur
M.	P. GALLINARI	Examinateur
M.	S. KNERR	Examinateur
M.	L. PERSONNAZ	Examinateur

A mon Père, ma Mère et Zina.

"Me tenant comme je suis,
un pied dans un pays et l'autre en un autre,
je trouve ma condition très heureuse,
en ce qu'elle est libre".

René Descartes
(Lettre à la princesse Elisabeth de Bohême,
Paris 1648).

Avant d'intégrer le laboratoire d'Électronique de l'ESPCI, je connaissais Monsieur le Professeur Gérard DREYFUS de réputation. Je ne savais pas alors que j'aurais un jour la chance de mener mon travail de thèse au sein de son équipe.

Mes plus vifs remerciements sont donc adressés au Professeur Gérard DREYFUS qui m'a témoigné de sa confiance en m'accueillant dans son laboratoire. Au cours de ces années de thèse, sa disponibilité sans faille, son suivi, son souci de la valorisation des travaux accomplis, son calme inébranlable devant les difficultés, ont beaucoup contribué à l'aboutissement de ce travail de thèse. Qu'il trouve ici toute ma reconnaissance.

Monsieur Léon PERSONNAZ, Maître de Conférences, a guidé mes premiers pas dans la recherche en encadrant mes deux premières années de thèse. Je resterai toujours impressionné par sa rigueur et son sens de la critique. Je tiens à lui exprimer mes remerciements pour ses relectures de mon mémoire et ses remarques.

Pendant ces années de thèse, Mademoiselle Isabelle RIVALS, Maître de Conférences, et moi avons partagé le même bureau, ce qui m'a permis à plusieurs reprises de bénéficier de ses connaissances. Je dois la remercier pour sa grande disponibilité.

J'adresse de vifs remerciements à Madame le Professeur Sylvie THIRIA, qui a accepté d'examiner mon mémoire de thèse, et qui a manifesté son intérêt pour mon travail.

Je tiens à exprimer ma reconnaissance à Monsieur le Professeur Stéphane CANU pour avoir examiné mon manuscrit avec beaucoup d'attention. Ses remarques constructives m'ont permis d'améliorer la version finale de mon mémoire.

Je suis très honoré que Monsieur le Professeur Patrick GALLINARI ait accepté de consacrer un peu de son temps, en cette période chargée de l'année, pour faire partie de mon jury.

Je tiens à remercier également Monsieur Stefan KNERR d'avoir également accepté d'être membre de mon jury, effectuant ainsi un "retour aux sources" en dépit de ses nombreuses activités.

Au cours de ces années de thèse au laboratoire d'Électronique, j'ai eu la chance de côtoyer Brigitte QUENET, Maître de Conférences, dont l'amitié et le

soutien m'ont beaucoup apporté. Mon travail a bénéficié de ses conseils et de ses encouragements.

Comment aurais-je pu m'initier aux systèmes informatiques en réseau sans la précieuse aide de Pierre ROUSSEL, Maître de Conférences, qui grâce à son administration rigoureuse des ressources informatiques du laboratoire, nous assure une bonne disponibilité des stations de travail ? J'ai beaucoup apprécié son sens de l'humour et sa convivialité.

Au travers de nombreuses discussions avec Hervé STOPPIGLIA, j'ai beaucoup appris sur les techniques de sélection utilisées dans ce mémoire. Je l'en remercie vivement.

Je voudrais adresser ici ma profonde reconnaissance à un ancien membre du laboratoire d'électronique qui par sa sympathie, son aide et ses encouragements a suscité en moi un véritable sentiment fraternel. C'est de Dominique URBANI que je veux parler .... Merci Doumé !

J'adresse enfin ma plus vive reconnaissance à Monique et François Zwobada qui sont devenus "ma famille française".

TABLE DES MATIÈRES

TABLE DES MATIÈRES DÉTAILLÉE

Introduction

Grâce aux résultats théoriques et pratiques obtenus au cours des dernières années, les réseaux de neurones sont devenus un outil de plus en plus utilisé dans divers domaines (industrie, banque, services). Ils demeurent toutefois un sujet d'un grand intérêt pour les chercheurs qui désirent améliorer les performances de ces réseaux et étendre leur champ d'applications.

La propriété fondamentale des réseaux de neurones, l'approximation universelle parcimonieuse, fait de ceux-ci une représentation mathématique très avantageuse pour la modélisation statique et dynamique non linéaire de processus. L'utilisation de neurones sigmoïdaux était initialement justifiée par une analogie biologique ; mais celle-ci est devenue caduque pour la conception de systèmes de traitement de signaux ou de modélisation de processus. Il est donc légitime d'explorer les possibilités d'utilisation d'autres types de neurones [Sontag93].

Cet effort de recherche d'une alternative aux réseaux de neurones "classiques" s'est tout d'abord dirigé vers les réseaux de fonctions radiales, en particulier gaussiennes. Ils ont notamment été mis en œuvre en Automatique non linéaire : modélisation de processus et commande. Les techniques de construction de ces réseaux aboutissent généralement à des modèles peu parcimonieux. En revanche, ils possèdent des propriétés plus intéressantes que les réseaux de neurones pour la synthèse de lois de commandes stables [Sanner92].

Récemment, des familles de fonctions, issues du traitement du signal et de l'image, appelées *ondelettes* ont été utilisées pour résoudre des problèmes d'approximation de fonctions [Pati93, Zhang92]. Ces ondelettes sont plus compliquées que les fonctions utilisées pour les réseaux de neurones classiques. En revanche, elles possèdent quelques propriétés prometteuses pour la modélisation de processus.

L'objectif principal de ce travail était donc l'étude de la mise en œuvre des fonctions ondelettes pour la modélisation statique (qui avait déjà été abordée par d'autres auteurs), et pour la modélisation dynamique de processus (qui, à notre connaissance, n'avait jamais été étudiée). Nous avons considéré deux approches issues de la transformée en ondelettes :

- **L'approche fondée sur la transformée continue,** très proche de celle des réseaux de neurones classiques, dont nous nous inspirons pour mettre au point une méthodologie de construction de *réseaux*

d'ondelettes. Elle permet d'envisager des réseaux bouclés (que nous proposons dans ce mémoire) et non bouclés.

• **L'approche fondée sur la transformée discrète**, propre aux fonctions ondelettes, qui permet de tirer parti des propriétés et des spécificités de ces fonctions pour la mise au point de procédures originales pour l'apprentissage de réseaux d'ondelettes.

Parmi les résultats théoriques concernant les bases de fonctions ondelettes, il a été prouvé que cette famille de fonctions possède la propriété d'approximation universelle. En revanche, il n'existe pas de résultat équivalent à celui des réseaux de neurones concernant la propriété de parcimonie. De ce fait, et sur la base des exemples que nous étudions conjointement avec des réseaux d'ondelettes et de neurones sigmoïdaux, nous nous proposons de faire une évaluation de la parcimonie des réseaux d'ondelettes.

De plus, nous avons systématiquement utilisé, pour l'estimation des paramètres des réseaux que nous avons mis en œuvre, deux algorithmes d'optimisation du second ordre : l'algorithme de BFGS et celui de Levenberg–Marquardt. Le premier a été largement utilisé pour l'apprentissage de réseaux bouclés et non bouclés. En revanche, des résultats sur l'utilisation du second pour l'apprentissage de réseaux bouclés sont, à notre connaissance, totalement absents de la littérature consacrée aux réseaux de neurones. Nous avons donc systématiquement cherché à comparer les résultats obtenus à l'aide de ces algorithmes, sous divers points de vue.

Le *chapitre I* du présent mémoire est consacré à des définitions et rappels concernant la modélisation, statique et dynamique de processus ; nous présentons notamment des considérations méthodologiques pour la construction de modèles "boîte noire", que nous avons mises en œuvre tout au long de ce travail. Cette approche s'inscrit dans la continuité de travaux antérieurs effectués au sein du laboratoire [Nerrand92, Rivals95a, Urbani95]. Nous décrivons ensuite les algorithmes d'optimisation employés pour l'estimation des paramètres des réseaux de fonctions, qu'il s'agisse de neurones à fonctions dorsales ou d'ondelettes fondées sur la transformée continue.

Le *chapitre II* présente les réseaux de neurones classiques que nous avons mis en œuvre pour la modélisation statique et dynamique de processus. Nous considérons deux types de fonctions dorsales : la fonction tangente hyperbolique, exemple de sigmoïde (qui est la brique des réseaux classiques), et la fonction gaussienne.

Pour cette dernière, nous proposons une procédure agissant en cours d'apprentissage, qui permet d'améliorer l'utilisation de chacun des neurones. Ces considérations sont illustrées par un exemple.

Le *chapitre III* est consacré aux réseaux d'ondelettes fondés sur la transformée continue. Après une brève présentation des fonctions ondelettes, nous proposons des algorithmes d'apprentissage de réseaux d'ondelettes bouclés pour une modélisation entrée–sortie et d'état. Les résultats présentés dans ce chapitre ont été publiés partiellement dans un article accepté pour publication dans la revue Neurocomputing [Oussar98], reproduit en annexe de ce mémoire.

Le *chapitre IV* aborde la modélisation de processus par des réseaux d'ondelettes fondés sur la transformée discrète. La particularité des bases d'ondelettes utilisées dans ce contexte ne permet pas d'apprentissage fondé sur une technique de gradient. De ce fait, la construction de ces réseaux est effectuée à l'aide de méthodes de sélection dans une bibliothèque d'ondelettes. Nous proposons dans ce chapitre une procédure qui met en œuvre ces bases d'ondelettes pour initialiser les coefficients de réseaux fondés sur la transformée continue, avant l'apprentissage de ceux-ci.

Les considérations développées dans les chapitres précédents sont appliquées, dans le *chapitre V*, à la modélisation d'un processus simulé, et d'un processus réel. Nous présentons d'abord les résultats obtenus avec des réseaux bouclés de fonctions dorsales et d'ondelettes. Ensuite, nous confrontons les performances réalisées par deux algorithmes du second ordre sur les deux types de réseaux.

CHAPITRE I

Modélisation de processus et estimation des paramètres d'un modèle

I. INTRODUCTION.

Dans la première partie de ce chapitre, nous rappelons les notions de processus et de modèle, ainsi que divers termes utilisés fréquemment dans le cadre de la modélisation. Dans la seconde partie, nous aborderons le problème de l'estimation des paramètres d'un modèle et nous présenterons les algorithmes qui ont été utilisés dans notre travail.

II. DÉFINITION D'UN PROCESSUS ET D'UN MODÈLE.

II.1 Processus.

Un processus est caractérisé par :

- une ou plusieurs grandeurs de sortie, mesurables, qui constituent le résultat du processus,
- une ou plusieurs grandeurs d'entrée (ou facteurs), qui peuvent être de deux types :
  - des entrées sur lesquelles il est possible d'agir (entrées de commande),
  - des entrées sur lesquelles il n'est pas possible d'agir (perturbations) ; ces dernières peuvent être aléatoires ou déterministes, mesurables ou non mesurables.

Les processus peuvent être de toutes natures : physique, chimique, biologique, écologique, financier, sociologique, etc.

II.2 Modèles.

II.2.1 Qu'est ce qu'un modèle ?

Nous nous intéressons ici aux modèles mathématiques, qui représentent les relations entre les entrées et les sorties du processus par des équations.
Si ces équations sont algébriques, le modèle est dit *statique*. Si ces équations sont des équations différentielles ou des équations aux différences récurrentes, le modèle est dit *dynamique*, respectivement *à temps continu* ou *à temps discret*.
Un modèle est caractérisé par son *domaine de validité*, c'est-à-dire par le domaine de l'espace des entrées dans lequel l'accord entre les valeurs des sorties du processus calculées par le modèle, et leurs valeurs mesurées, est considéré comme satisfaisant compte tenu de l'utilisation que l'on fait du modèle.

II.2.2 Buts d'une modélisation.

Un modèle peut être utilisé soit

- pour simuler un processus : à des fins pédagogiques, de détection d'anomalies de fonctionnement, de diagnostic de pannes, de conception assistée par ordinateur, etc.,
- pour effectuer la synthèse d'une loi de commande, ou pour être incorporé dans un dispositif de commande.

II.2.3 Classification des modèles.

II.2.3.1 Classification selon le mode de conception.

On distingue trois sortes de modèles en fonction des informations mises en jeu pour leur conception :

• **Les modèles de connaissance :** les modèles de connaissance sont construits à partir d'une analyse physique, chimique, biologique (ou autre suivant le type du processus), en appliquant soit les lois générales, fondées sur des principes (lois de la mécanique, de l'électromagnétisme, de la thermodynamique, de la physique quantique, etc.), soit les lois empiriques (finance, économie), qui régissent les phénomènes intervenant au sein des processus étudiés. Ces modèles ne comportent généralement pas de paramètres ajustables, ou des paramètres ajustables en très petit nombre.

Dans la pratique, il est toujours souhaitable d'établir un modèle de connaissance des processus que l'on étudie. Néanmoins, il arrive fréquemment que le processus soit trop complexe, ou que les phénomènes qui le régissent soient trop mal connus, pour qu'il soit possible d'établir un modèle de connaissance suffisamment précis pour l'application considérée. On est alors amené à concevoir des modèles purement empiriques, fondés exclusivement sur les résultats de mesures effectuées sur le processus.

• **Les modèles "boîte noire" :** les modèles "boîte noire" sont construits essentiellement sur la base de mesures effectuées sur les entrées et les sorties du processus à modéliser. La modélisation consiste alors à utiliser, pour représenter les relations entre les entrées et les sorties, des équations (algébriques, différentielles, ou récurrentes) paramétrées, et à estimer les paramètres, à partir des mesures disponibles, de manière à obtenir la meilleure précision possible avec le plus petit nombre possible de paramètres ajustables. Dans ce mémoire, nous désignerons fréquemment l'estimation des paramètres sous le terme *d'apprentissage*.

Le domaine de validité d'un tel modèle ne peut pas s'étendre au-delà du domaine des entrées qui est représenté dans les mesures utilisées pour l'apprentissage.

• **Les modèles "boîte grise" :** lorsque des connaissances, exprimables sous forme d'équations, sont disponibles, mais insuffisantes pour concevoir un modèle de connaissance satisfaisant, on peut avoir recours à une modélisation "boîte grise" (ou modélisation semi-physique) qui prend en considération à la fois les connaissances et les mesures. Une telle démarche peut concilier les avantages de l'intelligibilité d'un modèle de connaissance avec la souplesse d'un modèle comportant des paramètres ajustables.

II.2.3.2 Classification selon l'utilisation.

Indépendamment de la classification précédente, on peut distinguer deux types de modèles en fonction de l'utilisation qui en est faite.

• **Les modèles de simulation (ou simulateurs) :** un modèle de simulation est utilisé de manière indépendante du processus qu'il représente. Il doit donc posséder un comportement aussi semblable que possible à celui du processus. De tels modèles sont utilisés pour valider la conception d'un système avant sa fabrication (conception assistée par ordinateur en mécanique, en micro-électronique, ...), pour la formation de personnels (simulateurs de vols), pour la prévision à long terme, etc.
Du point de vue de la structure du modèle, les sorties passées, *mesurées sur le processus à modéliser*, ne peuvent constituer des entrées du modèle. L'estimation des paramètres et l'utilisation du modèle constituent deux phases successives et distinctes (apprentissage *non adaptatif*).

• **Les modèles de prédiction (ou prédicteurs) :** un modèle de prédiction est utilisé en parallèle avec le processus dont il est le modèle. Il prédit la sortie du processus à une échelle de temps courte devant les constantes de temps du processus. Les prédicteurs sont utilisés pour la synthèse de lois de commande, ou dans le système de commande lui-même (commande avec modèle interne).
Du point de vue de la structure du modèle, les sorties passées, mesurées sur le processus, peuvent constituer des entrées du modèle. L'estimation des paramètres et l'utilisation du modèle peuvent être effectuées simultanément si nécessaire (apprentissage *adaptatif*, utile notamment si les caractéristiques du processus dérivent dans le temps).

Ce mémoire présente la mise en oeuvre de plusieurs types de réseaux de fonctions paramétrées pour la modélisation dynamique de processus, et la comparaison de leurs performances respectives. Il s'agira donc exclusivement de modèles de type "boîte noire" qui peuvent être utilisés indifféremment comme simulateurs ou comme prédicteurs.

III. LES ÉTAPES DE LA CONCEPTION D'UN MODÈLE.

Lors de la conception d'un modèle de connaissance, la relation entre les entrées et la (ou les) sortie(s) du modèle découlent directement de la mise en équation des phénomènes physiques (chimiques, ou autres) qui régissent le fonctionnement du processus. Une fois le modèle obtenu sous forme analytique, des approximations peuvent être faites pour simplifier son expression (par exemple "linéariser" le modèle pour passer d'un modèle non linéaire à un modèle linéaire) si une telle approximation est justifiée.

Dans le cas d'une modélisation de type "boîte noire", la construction du modèle nécessite les trois élements suivants :

- Une hypothèse sur l'existence d'une relation déterministe liant les entrées à la (ou aux) sortie(s). Cette relation est caractérisée par une fonction appelée *fonction de régression* (ou plus simplement *régression)*. L'expression formelle supposée adéquate pour représenter cette relation est appelée *modèle-hypothèse*.
- Une séquence de mesures des entrées et de la sortie du processus.
- Un algorithme d'apprentissage.

Dans la suite de ce paragraphe, nous présentons les différents aspects qui doivent être pris en considération lors du choix d'un modèle-hypothèse.

III.1 Choix d'un modèle-hypothèse.

Les connaissances dont on dispose a priori sur le processus doivent guider le concepteur dans le choix de la modélisation la plus appropriée (statique ou dynamique, linéaire ou non linéaire, ...). L'élaboration du modèle-hypothèse nécessite d'effectuer les choix suivants :

• **Modèle statique ou dynamique** : lorsque l'on cherche à modéliser un processus physico-chimique ou biologique, il est généralement facile de savoir si l'application envisagée nécessite de modéliser la dynamique du processus (c'est-à-dire si l'on doit considérer une échelle de temps petite devant les constantes de temps du processus) ou si une modélisation statique suffit.

• **Modèle linéaire ou non linéaire :** il n'est pas douteux que la plupart des processus que l'on peut rencontrer nécessiteraient des modèles non linéaires s'il fallait les décrire de manière précise dans la totalité de leur domaine de fonctionnement : la plupart des modèles linéaires constituent des approximations valables dans un domaine plus ou moins restreint. Il est donc important de pouvoir élaborer un modèle non linéaire pour rendre compte du comportement d'un processus, non seulement autour de ses points de fonctionnement "habituels", mais également lors des passages d'un point de fonctionnement à un autre.

• **Modèle entrée-sortie ou modèle d'état** : dans le cas où l'on opte pour une modélisation dynamique, deux représentations sont possibles pour le modèle : il s'agit de la représentation d'état ou de la représentation entrée–sortie. L'état d'un processus est définit comme la quantité d'information minimale nécessaire pour prédire son comportement, étant données les entrées présentes et à venir. Il s'agit généralement d'un vecteur de grandeur égale à l'ordre du modèle. La représentation entrée–sortie est un cas particulier de la représentation d'état où le vecteur des états est constitué par la sortie et ses valeurs retardées dans le temps. Si le but de la modélisation est de prédire le comportement entrée–sortie du processus, il existe généralement une infinité de représentations d'état (au sens d'états ayant des trajectoires différentes) solutions du problèmes. En revanche, la représentation entrée–sortie est unique.

• **Présence de perturbations déterministes :** lorsque l'on cherche à réaliser un modèle dynamique, les perturbations déterministes peuvent être modélisées par une entrée supplémentaire (échelon, signal carré, sinusoïde). En particulier, si le modèle est construit pour la synthèse d'une loi de commande, la prise en considération de l'existence d'une perturbation pendant la phase de modélisation peut améliorer les performances de la commande pour le rejet de cette perturbation. Par exemple, il est proposé dans [Mukhopa93] une approche qui consiste à considérer la perturbation comme la sortie d'un processus. La modélisation de ce processus a pour effet d'introduire de nouvelles variables d'état, donc d'augmenter l'ordre du modèle.

• **Présence d'un bruit :** lorsque l'on cherche à réaliser un modèle dynamique, une perturbation de type "bruit" est modélisée par une séquence de variables aléatoires. Un bruit peut agir de différentes manières sur un processus. On distingue notamment le bruit de sortie (bruit additif qui affecte la mesure de la sortie du processus), et le bruit d'état (bruit additif qui affecte l'état du processus). Comme, en général, on ne connaît pas avec précision la nature du bruit qui

affecte le processus, on doit effectuer des hypothèses sur celle-ci ; on déduit de celles-ci la structure du modèle-hypothèse, et l'algorithme utilisé pour l'ajustement des paramètres. Une hypothèse erronée peut dégrader considérablement les performances du modèle. Ces problèmes ont été très largement étudiés dans le cas de la modélisation linéaire [Ljung87]. Dans le cadre de la modélisation non linéaire par réseaux de neurones, ces considérations sont développées dans [Nerrand94].

III.2 Du modèle-hypothèse au prédicteur ou au simulateur.

Un modèle-hypothèse ayant été choisi, l'étape suivante consiste à établir l'expression du prédicteur théorique, c'est-à-dire l'expression de la prédiction de la sortie du processus à l'instant $n+d$ en fonction des données disponibles à l'instant n (entrées et sorties du processus et/ou du prédicteur à l'instant n et aux instants antérieurs). Enfin, la dernière étape consiste à établir l'expression du prédicteur (ou du simulateur) proprement dit : dans le cas d'une modélisation "boîte noire", ce prédicteur utilise une fonction paramétrée, dont on estime les paramètres, à partir de mesures effectuées préalablement sur le processus, de telle manière qu'il constitue la meilleure approximation possible du prédicteur théorique. A l'issue de la procédure d'estimation des paramètres (apprentissage), il faut évaluer la performance du prédicteur (ou du simulateur).

Dans le cadre de ce mémoire nous nous intéressons plus particulièrement à l'étape d'apprentissage et donc aux caractéristiques du prédicteur (complexité, contraintes de mise en oeuvre) et aussi à l'algorithme d'apprentissage (efficacité, robustesse). La plupart des exemples étudiées étant des processus simulés, le problème du choix du modèle-hypothèse ne se pose pas. En revanche, la modélisation d'un processus réel (dans le dernier chapitre) sera l'occasion d'examiner ce problème.

III.3 Présentation de quelques modèles-hypothèses et de leurs prédicteurs associés.

Nous présentons dans ce paragraphe quelques exemples de modèles-hypothèses ainsi que les prédicteurs qui leurs sont associés, pour l'élaboration d'un modèle dynamique entrée-sortie. L'un des principaux paramètres qui interviennent dans le choix d'un modèle-hypothèse est la présence d'un bruit et la manière dont il agit sur le processus. Pour ceci, nous allons considérer deux classes de modèles-hypothèses : le modèle-hypothèse déterministe et des modèles-hypothèses non déterministe (faisant intervenir un bruit dans la modélisation du processus).

III.3.1 Modèle-hypothèse déterministe.

On considère qu'aucun bruit n'agit sur le processus. On propose un modèle-hypothèse déterministe ayant l'expression suivante :

$$y_p(n) = f\left(y_p(n\pm1), \ldots, y_p(n\pm N_s), u(n\pm1), \ldots, u(n\pm N_e)\right) \quad (1)$$

où $y_p(n)$ est la sortie mesurée du processus à l'instant n, N_s est l'ordre du modèle et N_e la mémoire sur l'entrée externe u. f est une fonction non linéaire dont on suppose qu'elle existe, et qu'elle constitue une représentation mathématique du comportement du processus.

La forme prédicteur théorique associée à ce modèle-hypothèse est la suivante :

$$y(n) = f\left(y_p(n\pm1), \ldots, y_p(n\pm N_s), u(n\pm1), \ldots, u(n\pm N_e)\right) \quad (2)$$

où $y(t)$ est la prédiction de la sortie du processus calculée par la forme prédicteur théorique. Étant donné que nous considérons que le processus n'est soumis à aucun bruit, la forme prédicteur théorique doit calculer à tout instant $y(t) = y_p(t)$.

Le prédicteur dont on effectuera l'apprentissage aura pour expression :

$$y(t) = \psi\left(y_p(t\pm1), \ldots, y_p(t\pm N_s), u(t\pm1), \ldots, u(t\pm N_e)\right) \quad (3)$$

où ψ est une fonction paramétrée, dont les paramètres doivent être estimés pour qu'elle approche au mieux la fonction f dans le domaine de fonctionnement considéré. Cette optimisation s'entend au sens de la minimisation de la fonction de coût empirique, que l'on appellera dorénavant fonction de coût et que l'on notera par J. Cette minimisation est réalisée à l'aide d'un algorithme d'apprentissage.

Si l'on est intéressé par la construction d'un modèle de simulation, un autre prédicteur peut être considéré :

$$y(n) = \psi\left(y(n\pm1), \ldots, y(n\pm N_s), u(n\pm1), \ldots, u(n\pm N_e)\right) \quad (4)$$

La seule différence avec la forme prédicteur de la relation (3) réside dans le fait que les entrées d'état du modèle sont les sorties retardées *du modèle*, non celles *du processus*.

III.3.2 Modèles-hypothèses non déterministes.

On désigne par "modèles-hypothèses non déterministes" des modèles-hypothèses qui supposent l'existence d'un bruit agissant sur le processus à modéliser. On peut envisager plusieurs hypothèses concernant la manière dont le bruit agit sur le processus. Nous en présentons deux, que nous considérerons lors de l'étude d'exemples dans ce mémoire.

III.3.2.1 L'hypothèse "Bruit de sortie".

L'hypothèse "Bruit de sortie" (Output Error en anglais) consiste à considérer qu'un bruit agit sur la sortie du processus. L'expression du modèle-hypothèse est :

$$\begin{cases} x(n) = f(x(n\pm1), \dots, x(n\pm N_s), u(n\pm1), \dots, u(n\pm N_e)) \\ y_p(n) = x(n) + w(n) \end{cases} \quad (5)$$

où $\{w(n)\}$ est une séquence de variables aléatoires indépendantes de moyenne nulle et de variance σ^2. La forme prédicteur théorique associée à ce modèle-hypothèse est donnée par l'expression suivante :

$$y(n) = f\left(y(n\pm1), y(n\pm2), \dots, y\left(n\pm N_s\right), u(n\pm1), \dots, u\left(n\pm N_e\right)\right) \quad (6)$$

Le prédicteur réel associé a pour expression :

$$y(n) = \psi\left(y(n\pm1), y(n\pm2), \dots, y\left(n\pm N_s\right), u(n\pm1), \dots, u\left(n\pm N_e\right)\right) \quad (7)$$

où ψ est une fonction réalisée à l'aide d'une fonction paramétrée, par exemple un réseau de neurones. C'est donc un modèle dont les entrées d'état sont ses propres sorties retardées, et non pas les sorties du processus. Si, après apprentissage, la fonction ψ était identique à la fonction f, l'erreur de prédiction commise par ce prédicteur serait une séquence aléatoire de mêmes caractéristiques que w. Lorsque la fonction parmétrée ψ est réalisée par un réseau de neurones, celui-ci est un réseau bouclé, que nous décrirons au paragraphe II.4.2 du chapitre suivant.

III.3.2.2 L'hypothèse "Bruit d'état".

L'hypothèse "Bruit d'état" (Equation Error en anglais) consiste à considérer qu'un bruit agit sur l'état du processus. Ce modèle-hypothèse a la forme suivante :

$$y_p(n) = f\left(y_p(n\pm1), y_p(n\pm2), \dots, y_p\left(n\pm N_s\right), u(n\pm1), \dots, u\left(n\pm N_e\right)\right) + w(n) \quad (8)$$

où $\{w(n)\}$ est une séquence de variables aléatoires indépendantes de moyenne nulle et de variance σ^2. La forme prédicteur théorique associée à ce modèle-hypothèse est donnée par l'expression suivante :

$$y(n) = f\left(y_p(n\pm1), y_p(n\pm2), \dots, y_p\left(n\pm N_s\right), u(n\pm1), \dots, u\left(n\pm N_e\right)\right) \quad (9)$$

Le prédicteur réel associé est de la forme :

$$y(n) = \psi\left(y_p(n\pm1), y_p(n\pm2), \dots, y_p\left(n\pm N_s\right), u(n\pm1), \dots, u\left(n\pm N_e\right)\right) \quad (10)$$

où ψ est une fonction paramétrée. Si ψ était identique à f, l'erreur de prédiction effectuée par ce prédicteur serait une séquence de variables aléatoires de mêmes caractéristiques que le bruit w. Lorsque la fonction parmétrée ψ est réalisée par un réseau de neurones, celui-ci est un réseau non bouclé, que nous décrirons au paragraphe II.4.1 du chapitre suivant.

IV. FONCTIONS PARAMÉTRÉES POUR LA MODÉLISATION "BOÎTE NOIRE".

Comme indiqué ci–dessus, une modélisation de type "boîte noire" est mise en œuvre dans le cas où l'on dispose de peu de connaissance sur le processus étudié, ou si le modèle de connaissance établi est trop compliqué pour être exploité. Dans les deux cas (et particulièrement dans le second) on a besoin d'un outil fournissant un modèle précis, aussi simple que possible en termes de nombre de paramètres ajustables et de nombre de calculs à effectuer, pour prédire la sortie du processus.

En général, un modèle "boîte noire" *statique* est une combinaison paramétrée de fonctions, qui peuvent être elles-mêmes paramétrées. Un modèle "boîte noire" *dynamique* est, comme nous l'avons vu ci-dessus, un ensemble d'équations différentielles (ou d'équations aux différences pour un modèle à temps discret) non linéaires, où la non-linéarité est réalisée, comme dans le cas d'un modèle statique, par une combinaison paramétrées de fonctions éventuellement paramétrées.

Des fonctions paramétrées constituent une famille *d'approximateurs universels* s'il est possible (sous certaines conditions de régularité) d'approcher toute fonction continue, avec la précision voulue, dans un domaine de l'espace des entrées, par une somme pondérée d'un nombre fini de ces fonctions.

Cette condition n'est néanmoins pas suffisante pour qu'une famille de fonctions soit utilisable de manière efficace pour la modélisation "boîte noire" efficace. En effet, parmi tous les modèles possibles, on recherche toujours celui qui possède le plus petit nombre de coefficients ajustables : c'est la propriété de *parcimonie,* dont nous verrons qu'elle n'est pas partagée par tous les types de fonctions paramétrées. A cet égard, il est important de distinguer les modèles *linéaires par rapport aux paramètres* des modèles *non linéaires par rapport aux paramètres.*

IV.1 Les fonctions paramétrées linéaires par rapport aux paramètres.

Une fonction paramétrée est linéaire par rapport aux paramètres si elle est de la forme :

$$\psi(X) = \sum_{i=1}^{N} \theta_i \, \Phi_i(X) \tag{11}$$

où les $\Phi_i(X)$ sont des fonctions non paramétrées d'une ou plusieurs variables groupées dans le vecteur X, et où les θ_i sont des paramètres.

Les fonctions $\Phi_i(X)$ peuvent être quelconques ; traditionnellement on utilise des monômes ; mais on peut également utiliser d'autres types de fonctions : fonctions splines, fonctions gaussiennes dont les centres et les écarts–types sont fixés,

fonctions ondelettes dont les translations et dilatations sont fixées (ces dernières seront présentées au chapitre IV de ce mémoire).

IV.2 Les fonctions paramétrées non linéaires par rapport aux paramètres.

Dans le présent travail, nous utiliserons essentiellement des fonctions non linéaires par rapport aux paramètres, qui sont de la forme

$$\psi(X) = \sum_{i=1}^{N} \theta_i \, \Phi_i(X, \Theta_i) \tag{12}$$

où Θ_i est un vecteur de paramètres de la fonction Φ_i. Ainsi, la fonction réalisée est linéaire par rapport aux θ_i, mais non linéaire par rapport aux paramètres constituant le vecteur Θ_i : c'est une combinaison linéaire de fonctions paramétrées.

Les réseaux de neurones à une couche cachée (présentés au chapitre II), les réseaux de fonctions gaussiennes radiales dont les centres et les écarts-types sont ajustables, les réseaux d'ondelettes (qui sont l'objet essentiel de ce travail) entrent dans cette catégorie de fonctions. Toutes ces fonctions sont des approximateurs universels [Hornik89] mais leur intérêt, par rapport aux fonctions linéaires par rapport aux paramètres, réside dans le caractère *parcimonieux* des modèles qu'ils permettent de réaliser [Hornik94]. Comme nous le verrons au paragraphe V.2, le prix à payer pour cela réside dans le fait que les méthodes habituelles d'estimation de paramètres (méthodes de moindres carrés) sont inutilisables, et que l'on doit avoir recours à des méthodes itératives (méthodes de gradient) dont la mise en œuvre est plus lourde.

Nous présentons brièvement ci-dessous ces trois types de réseaux, dont deux seront repris en détail dans les chapitres suivants.

IV.2.1 Les réseaux de neurones.

Dans ce travail, nous réserverons le terme de réseau de neurones aux réseaux de la forme (12) , où au moins une des fonctions $\Phi_i(X)$ est une fonction croissante bornée, notamment sigmoïde (tangente hyperbolique), d'une combinaison linéaire des entrées ; certaines de ces fonctions peuvent être l'identité. L'expression de ces réseaux est :

$$\psi(X) = \sum_{i=1}^{N} \theta_i \, \Phi_i(\Theta_i^T X) \tag{13}$$

Issus de travaux à connotation biologique dans les années 1940, ces réseaux sont maintenant considérés comme des outils mathématiques, indépendamment de toute référence à la biologie. Ils sont utilisés pour la modélisation et la commande

de processus non linéaires, ainsi que comme outils de classification, notamment pour la reconnaissance de formes.

Les principales étapes dans l'évolution de la théorie et de la pratique des réseaux de neurones ont été la mise au point d'un algorithme, économique en temps de calcul, pour l'évaluation du gradient de la fonction de coût (définie au paragraphe V), appelé *algorithme de rétropropagation* [Rumelhart86], et la preuve de ses propriétés d'approximateur universel [Hornik89] et de parcimonie [Barron93, Hornik94]. L'une des premières applications dans le domaine de la modélisation non linéaire de processus est présentée dans [Narendra90].

IV.2.2 Les réseaux de fonctions radiales (RBF pour Radial Basis Functions).

Les fonctions radiales ont été introduites par [Powell85] dans le cadre de *l'interpolation*, c'est-à-dire de la recherche de fonctions passant *exactement* par un nombre fini de points (dits points de collocation). Dans ce contexte, la fonction recherchée est une combinaison linéaire de fonctions de base, en nombre égal au nombre de points de collocation ; une fonction de base $\Phi_n(x)$, relative au point de collocation x_n, est dite radiale si elle ne dépend que de la distance du point courant x au point de collocation x_n. On peut utiliser diverses fonctions radiales, notamment des fonctions localisées (qui tendent vers zéro dans toutes les directions de l'espace des variables) telles que des gaussiennes centrées aux points de collocation. Bien entendu, la recherche d'une fonction passant exactement par les points n'a de sens que si ces points ne sont pas entachés de bruit.

La référence [Broom88] semble être parmi les premières à proposer l'idée d'utiliser des réseaux de RBF pour *l'approximation de fonctions* non linéaires. La fonction recherchée est toujours une combinaison linéaire de fonctions radiales, mais leur nombre est beaucoup plus petit que le nombre de points, et elles ne sont donc pas forcément centrées en ces points. Son expression est de la forme :

$$\psi(X) = \sum_{i=1}^{N} \theta_i \, \Phi_i\left(\left\| \left(X \pm M_i\right) \right\|, \sigma_i^2\right) \tag{14}$$

où M_ι est le vecteur des centres et σ_i^2 un scalaire (appelé variance dans le cas d'une RBF gaussienne).

La propriété d'approximateurs universels pour ces réseaux n'a été que récemment prouvée pour des gaussiennes radiales [Hartman90] et plus généralement pour des RBF [Park91].

Ces réseaux ont été utilisés comme outil de modélisation "boîte noire" dans le domaine de l'automatique. On les trouve à la base de modèles entrée–sortie [Chen90] et aussi de modèles d'état [Elanayar94]. Certaines spécificités de ces réseaux permettent de les utiliser pour la synthèse de lois de commande adaptatives stables [Behera95, Sanner92, Sanner95]. Le fait que ces réseaux

permettent de garantir la stabilité des correcteurs qu'ils réalisent les rend plus intéressants que les réseaux de neurones pour la résolution des problèmes de commande non linéaire. En revanche, cette propriété se fait au détriment de la parcimonie du réseau.

IV.2.3 Les réseaux d'ondelettes.

Les fonctions ondelettes trouvent leur origine dans des travaux de mathématiciens dès les années 1930. L'idée de départ était de construire une transformation, pour l'étude des signaux, plus commode que la transformation de Fourier, notamment pour des signaux de durée finie.

Les fonctions ondelettes ont subi une évolution au cours des années : celles dont nous disposons aujourd'hui sont plus complexes que leurs aînées, et possèdent des propriétés intéressantes pour l'approximation de fonctions. En particulier, elles possèdent la propriété d'approximateurs universels, ce qui suggère leur utilisation pour la construction de modèles "boîte noire".

La notion de réseaux d'ondelettes existe depuis peu [Pati 93] et l'étude de la propriété de parcimonie n'a pas été abordée. L'un des objectifs de ce mémoire est l'étude de la mise en oeuvre de cette classe de réseaux pour la modélisation entrée–sortie et d'état de processus, ainsi que la comparaison, sur des exemples, de la parcimonie et des performances de cette classe de réseaux par rapport à celle des réseaux de neurones (voir les chapitres III, IV et V).

V. ESTIMATION DES PARAMÈTRES D'UN MODÈLE.

V.1 Position du problème et notations.

Étant données les informations dont on dispose sur le processus (c'est à dire la séquence d'apprentissage) on détermine, dans une famille donnée de fonctions paramétrées $\psi(x, \theta)$ (où x est le vecteur regroupant toutes les entrées du modèle et θ le vecteur des paramètres inconnus de ψ) celle qui minimise une fonction de coût qui, le plus souvent, est la fonction de coût des moindres carrés.

Soit y_p^n la sortie du processus à l'instant n (dans le cas d'une modélisation dynamique), ou la valeur mesurée pour le $n^{\text{ème}}$ exemple de l'ensemble d'apprentissage (dans le cas d'une modélisation statique). De même, y^n est la sortie calculée par le modèle à l'instant n, *ou* pour le $n^{\text{ème}}$ exemple de l'ensemble d'apprentissage. On définit la fonction de coût des moindres carrés $J(\theta)$ par :

$$J(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left(y_p^n \pm y^n \right)^2 \qquad (15)$$

où N est le nombre de mesures (taille de la séquence). $J(\theta)$dépend du vecteur des paramètres, ainsi que de la séquence d'apprentissage. Pour alléger les notations, nous n'indiquerons pas explicitement cette dernière dépendance dans la suite.
On définit l'erreur quadratique moyenne d'apprentissage (EQMA) comme une la moyenne de la fonction de coût calculée sur la séquence d'apprentissage. Elle est donnée par : $\frac{2J(\theta)}{N}$.

Lors de son exploitation, le modèle reçoit des entrées différentes de celles de la séquence d'apprentissage. On peut estimer ses performances en calculant diverses fonctions ; celle que l'on utilise le plus fréquemment est l'erreur quadratique moyenne de performance *EQMP* dont la valeur est calculée sur une séquence différente de celle utilisée pour l'apprentissage.

V.2 Les algorithmes de minimisation de la fonction de coût.

Dans le cas où le modèle est linéaire par rapport aux paramètres à ajuster, la minimisation de la fonction de coût, et donc l'estimation du vecteur des paramètres θ, peut se faire à l'aide la méthode des moindes carrés, qui ramène le problème à la résolution d'un système d'équations linéaires. Nous présentons cette technique dans ce qui suit.

V.2.1 Méthode des moindres carrés ordinaires.

Cette méthode est applicable pour l'apprentissage de modèles statiques ou de prédicteurs non bouclés dont la sortie est linéaire par rapport aux paramètres inconnus. Si cette sortie est linéaire par rapport aux entrées, le prédicteur associé a pour expression :

$$y(n) = \sum_{i=1}^{N_i} \theta_i x_i(n) \tag{16}$$

Ce modèle prédictif peut se être mis sous forme d'une équation matricielle. En effet, on peut l'écrire $Y = X\,\theta$ avec :

$$Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix}, X = \begin{bmatrix} x_1(1) & x_2(1) & \cdots & x_{N_i}(1) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_1(N) & \cdots & \cdots & x_N(N) \end{bmatrix}, \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N_i} \end{bmatrix} \tag{17}$$

L'estimation des paramètres est fondée sur la minimisation de la fonction de coût des moindres carrés (relation (15)). En utilisant la notation matricielle présentée ci–dessus, l'expression de la fonctionde coût $J(\theta)$ devient :

$$J(\theta) = \frac{1}{2}\left(Y_P^T Y_P \pm 2\theta^T X^T Y_p + \theta^T X^T X\theta\right) \tag{18}$$

La fonction de coût étant quadratique (par rapport au vecteur des paramètres à estimer), il atteint son minimum pour la valeur du vecteur des paramètres annulant sa dérivée. Soit θ_{mc} cette valeur du vecteur des paramètres. Elle vérifie :

$$\left.\frac{\partial J}{\partial \theta}\right|_{\theta_{mc}} = 0 \tag{19}$$

Cette dernière équation fournit l'équation normale :

$$\left[X^T X\right]\theta_{mc} = \left[X^T Y_p\right] \tag{20}$$

dont la solution θ_{mc} donnée par :

$$\theta_{mc} = \left[X^T X\right]^{\pm 1}\left[X^T Y_p\right] \tag{21}$$

est l'estimation des moindres carrés du vecteur des paramètres θ_p . Cette solution existe à condition que la matrice $X^T X$ soit inversible. Cette condition est généralement vérifiée lorsque N (le nombres d'exemples) est très grand devant N_i (le nombre d'entrées du modèle).

La méthode des moindres carrés peut être utilisée plus généralement pour l'estimation des paramètres de tout modèle dont la sortie est linéaire par rapport aux paramètres à estimer ; c'est le cas, par exemple, pour l'estimation des paramètres du modèle suivant :

$$y(n) = \sum_{i=1}^{N_i} \theta_i \Phi_i(X) \tag{22}$$

où les Φ_i sont des fonctions non paramétrées du vecteur des entrées X. Plusieurs choix sont possibles pour les fonctions Φ_i (voir paragraphe IV.1).

Les sorties des modèles "boîte noire" que nous utilisons dans ce mémoire ne sont pas linéaires par rapport aux paramètres à ajuster. Une résolution directe du problème comme dans le cas de la solution des moindres carrés n'est donc pas possible : on a donc recours à des algorithmes d'apprentissage qui recherchent une solution suivant une procédure itérative. Ces algorithmes sont généralement applicables sauf dans le cas où des restrictions sur les valeurs possibles pour les paramètres du modèle sont imposées par la nature des fonctions paramétrées utilisées (voir le paragraphe III.1 du chapitre IV).

Dans ce qui suit, nous allons présenter les algorithmes que nous utilisons dans ce mémoire pour la minimisation de la fonction de coût.

V.2.2 Principe des algorithmes de gradient.

Les algorithmes d'apprentissage fondés sur l'évaluation du gradient de la fonction de coût $J(\theta)$ par rapport aux paramètres procèdent à la minimisation de

manière itérative. $J(\theta)$ est une fonction scalaire à variable vectorielle (le vecteur θ des paramètres à ajuster). Son gradient est donc un vecteur défini par :

$$\nabla J = \begin{pmatrix} \dfrac{\partial J}{\partial \theta_1} \\ \vdots \\ \dfrac{\partial J}{\partial \theta_M} \end{pmatrix} \tag{23}$$

où M est le nombre de paramètres inconnus.
Le principe des algorithmes de gradient repose sur le fait qu'un minimum de la fonction de coût est atteint si sa dérivée (son gradient) est nul. Il existe plusieurs types d'algorithmes ; nous présenterons ceux que nous utiliserons dans la suite. Leur déroulement suit le schéma suivant :

A l'itération 0 : Initialiser le vecteur des paramètres à θ^0. Cette initialisation de θ peut avoir une grande influence sur l'issue de l'apprentissage. Nous porterons une attention particulière à cette étape. Nous proposons une technique d'initialisation pour réseaux d'ondelettes au chapitre IV.

A la k^{ème} itération : Calculer la fonction de coût et la norme du gradient avec le vecteur des paramètres courant (obtenu à l'itération précédente).

Si $J(\theta^{k-1}) \le J_{\max}$ ou $\|\nabla J\| \le \varepsilon$ ou $k = k_{max}$ (où J_{max} est une valeur maximale recherchée pour l'EQMA, ou pour l'EQMP si les performances sont évaluées pendant l'apprentissage),

Alors arrêter l'algorithme ; le vecteur $\theta^{k\pm1}$ est une solution,

Sinon calculer θ^k à partir de $\theta^{k\pm1}$ par la formule de mise à jour des paramètres suivante :

$$\theta^k = \theta^{k-1} + \mu_k d_k \tag{24}$$

où μ_k est un scalaire positif appelé pas du gradient et d_k un vecteur calculé à partir du gradient, appelé direction de descente. Les différences entre les méthodes de gradient résident dans le choix de la direction de descente et dans le choix du pas.

V.2.3 La méthode du gradient simple.

V.2.3.1 Présentation de la méthode.

La méthode du gradient simple consiste à la mise en œuvre de la formule de mise à jour des paramètres suivante :

$$\theta^k = \theta^{k-1} \pm \mu_k \nabla J\left(\theta^{k-1}\right) \tag{25}$$

La direction de descente est donc simplement l'opposée de celle du gradient ; c'est en effet la direction suivant laquelle la fonction de coût diminue le plus rapidement.

En pratique, la méthode du gradient simple peut être efficace lorsque l'on est loin du minimum de J . Quand on s'en approche, la norme du gradient diminue et donc l'algorithme progresse plus lentement. A ce moment, on peut utiliser une méthode de gradient plus efficace.

Un "réglage" du pas de gradient μ_k est nécessaire : en effet, une petite valeur de ce paramètre ralentit la progression de l'algorithme ; en revanche une grande valeur aboutit généralement à un phénomène d'oscillation autour de la solution. Diverses heuristiques, plus ou moins efficaces, ont été proposées.

V.2.3.2 Techniques de réglage du pas.

• **Technique du pas constant :** elle consiste à adopter un pas constant $\mu_k = \mu$ tout au long de l'algorithme. Elle est très simple mais peu efficace puisqu'elle ne prend pas en considération la décroissance de la norme du gradient.

• **Technique du pas asservi :** on peut asservir le pas à l'aide de la norme du gradient de sorte que le pas évolue en sens inverse de celle–ci. A chaque étape, le pas peut être calculé par :

$$\mu_k = \frac{\mu}{1 + \| \nabla J \|} \tag{26}$$

où μ est un paramètre constant. Lors de l'utilisation de cette technique, nous avons adopté la valeur $\mu = 10^{-3}$ qui s'est révélée très souvent satisfaisante. Le numérateur est augmenté du nombre 1 afin d'éviter une instabilité numérique au moment de la division dans le cas où la norme du gradient devient très proche du zéro. Cette technique offre un bon compromis du point de vue de la simplicité et de l'efficacité. C'est celle que nous avons utilisée chaque fois que nous avons mis en œuvre la méthode du gradient simple.

V.2.4 Les méthodes de gradient du second ordre.

Les méthodes que nous venons de décrire sont simples mais en général très inefficaces. On a donc systématiquement recours à l'utilisation de méthodes plus

performantes (pour une comparaison numérique entre ces méthodes, voir [Battiti92]). Elles sont dites du second ordre parce qu'elles prennent en considération la dérivée seconde de la fonction de coût. Nous présentons ci-dessous celles que nous avons mises en œuvre dans notre travail, et dont nous comparons les performances lors de l'étude de nos exemples.

V.2.4.1 L'algorithme de BFGS.

L'algorithme de BFGS (du nom de ses inventeurs : Broyden, Fletcher, Goldfarb et Shanno) [Minoux83] fait partie des méthodes d'optimisation dites "quasi–newtoniennes". Ces méthodes sont une généralisation de la méthode de Newton.

La méthode de Newton consiste à l'application de la règle suivante :

$$\theta^k = \theta^{k-1} \pm \left[H\left(\theta^{k-1}\right)\right]^{\pm 1} \nabla J\left(\theta^{k-1}\right) \tag{27}$$

où $H(\theta)$ est le Hessien de la fonction J calculé avec le vecteur des paramètres disponible à l'étape courante. La direction de descente est dans ce cas :

$$d_k = \pm \left[H\left(\theta^{k-1}\right)\right]^{\pm 1} \nabla J\left(\theta^{k-1}\right) \tag{28}$$

Le pas μ_k est constant et égal à 1.

Pour que le déplacement soit en sens contraire du gradient, il est indispensable que la matrice du Hessien soit définie positive. Sous cette condition, et si la fonction de coût est quadratique par rapport aux paramètres, la méthode de Newton converge vers l'unique solution en une seule itération.

En général, et pour les problèmes d'optimisation auxquels nous sommes confrontés dans ce mémoire, la fonction de coût n'est généralement pas quadratique. Elle peut néanmoins l'être localement, à proximité d'un minimum de ses minima. Donc, la méthode de Newton ne peut converger en une seule itération. De plus, cette méthode nécessite l'inversion de la matrice du Hessien à chaque itération (puisqu'il apparaît que plusieurs sont nécessaires), ce qui conduit à des calculs lourds.

L'algorithme de BFGS, ainsi que l'algorithme de Levenberg-Marquardt présenté dans le paragraphe suivant, sont des méthodes "quasi-newtoniennes" qui permettent de pallier ces inconvénients.

L'algorithme de BFGS est une règle d'ajustement des paramètres qui a l'expression suivante :

$$\theta^k = \theta^{k-1} \pm \mu_k M_k \nabla J\left(\theta^{k-1}\right) \tag{29}$$

où M_k est une approximation, calculée itérativement, de l'inverse de la matrice Hessienne. L'approximation de l'inverse du Hessien est modifiée à chaque itération suivant la règle suivante :

$$M_k = M_{k\pm1} + \left[1 + \left(\frac{\gamma_{k-1}^T M_{k-1}\gamma_{k-1}}{\delta_{k-1}^T\gamma_{k-1}}\right)\right]\frac{\delta_{k-1}^T\delta_{k-1}}{\delta_{k-1}^T\gamma_{k-1}} \pm \frac{\delta_{k-1}\gamma_{k-1}^T M_{k-1} + M_{k-1}\gamma_{k-1}\delta_{k-1}^T}{\delta_{k-1}^T\gamma_{k-1}} \qquad (30)$$

avec $\gamma_{k-1} = \nabla J(\theta^k) \pm \nabla J(\theta^{k-1})$ et $\delta_{k-1} = \theta^k \pm \theta^{k-}$. Nous prenons pour valeur initiale de M la matrice identité. Si, à une itération, la matrice calculée n'est pas définie positive, elle est réinitialisée à la matrice identité.

Reste la question du choix du pas μ_k. A cet effet, nous avons opté pour une méthode économique en calculs, la technique de Nash [Nash80]. Cette technique recherche un pas qui vérifie la condition de descente :

$$J(\theta^{k-1} + \mu_k d_k) \le J(\theta^{k-1}) + m_1 \mu_k d_k^T \nabla J(\theta^{k-1}) \qquad (31)$$

où m_1 est un facteur choisi très inférieur à 1 (par exemple $m_1 = 10^{-3}$).

En pratique, la recherche du pas se fait de manière itérative. On initialise μ_k à une valeur positive arbitraire. On teste la condition (31). Si elle est vérifiée, on accepte l'ajustement des paramètres. Sinon, on multiplie le pas par un facteur inférieur à 1 (par exemple 0.2) et on teste à nouveau la condition de descente. On répète cette procédure jusqu'à ce qu'une valeur satisfaisante du pas soit trouvée. Au bout de 22 essais, le pas atteint une valeur de l'ordre de 10^{-16}. On peut considérer alors qu'il n'est pas possible de trouver un pas satisfaisant.

Une méthode "quasi–newtonienne", n'est efficace que si elle est appliquée au voisinage d'un minimum. D'autre part, la règle du gradient simple est efficace lorsqu'on est loin du minimum et sa convergence ralentit considérablement lorsque la norme du gradient diminue (c'est à dire lorsqu'on s'approche du minimum). Ces deux techniques sont donc complémentaires. De ce fait, l'optimisation s'effectue en deux étapes : utilisation de la règle du gradient simple pour approcher un minimum, et de l'algorithme de BFGS pour l'atteindre. Le critère d'arrêt est alors un des critères décrits au paragraphe V.2.2.

V.2.4.2 L'algorithme de Levenberg–Marquardt.

L'algorithme de Levenberg–Marquardt [Levenberg44, Marquardt63] repose sur l'application de la formule de mise à jour des paramètres suivante :

$$\theta^k = \theta^{k-1} \pm \left[H(\theta^{k-1}) + \mu_k I\right]^{\pm1} \nabla J(\theta^{k-1}) \qquad (32)$$

où $H(\theta^{k-1})$ est le Hessien de la fonction de coût et μ_k est le pas. Pour de petites valeurs du pas, la méthode de Levenberg–Marquardt s'approche de celle de Newton. Inversement, pour de grandes valeurs de μ_k, l'algorithme Levenberg–Marquardt est équivalent à l'application de la règle du gradient simple avec un pas de $\frac{1}{\mu_k}$.

La première question relative à cet algorithme est celle de l'inversion de la matrice $\left[H\left(\theta^{k-1}\right)+\mu_k I\right]$. L'expression exacte du Hessien de la fonction J est :

$$H\left(\theta^k\right)=\sum_{n=1}^{N}\left(\frac{\partial e^n}{\partial \theta^k}\right)\left(\frac{\partial e^n}{\partial \theta^k}\right)^T+\sum_{n=1}^{N}\frac{\partial^2 e^n}{\partial \theta^k\,\partial\left(\theta^k\right)^T}e^n \tag{33}$$

avec $e^n=y_p^n\pm y^n$.

Le second terme de l'expression étant proportionnel à l'erreur, il est permis de le négliger en première approximation, ce qui fournit une expression approchée :

$$\tilde{H}\left(\theta^k\right)=\sum_{n=1}^{N}\left(\frac{\partial e^n}{\partial \theta^k}\right)\left(\frac{\partial e^n}{\partial \theta^k}\right)^T=\sum_{n=1}^{N}\left(\frac{\partial y^n}{\partial \theta^k}\right)\left(\frac{\partial y^n}{\partial \theta^k}\right)^T \tag{34}$$

Dans le cas d'un modèle linéaire par rapport aux paramètres, c'est à dire si y est une fonction linéaire de θ, le second terme de l'expression de H est nul est l'approximation devient exacte.

Plusieurs techniques sont envisageables pour l'inversion de la matrice $\left[\tilde{H}+\mu_k I\right]$.

• **Inversion indirecte.**

Un lemme d'inversion permet de calculer la matrice inverse suivant une loi récurrente. En effet, soient *A*, *B*, *C* et *D* quatre matrices. On a la relation suivante : $\left(A+B\,C\,D\right)^{-1}=A^{-1}\pm A^{-1}B\left(C^{-1}+D\,A^{-1}\,B\right)^{-1}D\,A^{-1}$.

D'autre part, en posant $X^n=\dfrac{\partial y^n}{\partial \theta^k}$, l'approximation de la matrice H peut être calculée à partir de la loi de récurrence suivante :

$$\tilde{H}^n=\tilde{H}^{n-1}+X^n\left(X^n\right)^t \quad \text{avec} \quad n=1,\ldots,N$$

De ce fait, on a $\tilde{H}=\tilde{H}^N$.

Si l'on applique le lemme d'inversion à la relation précédente en choisissant $A=\tilde{H}$, $B=X^n$, $C=I$ et $D=\left(X^n\right)^T$, on obtient la relation suivante :

$$\left(\tilde{H}^n\right)^{-1}=\left(\tilde{H}^{n-1}\right)^{-1}\pm\frac{\left(\tilde{H}^{n-1}\right)^{-1}X^n\left(X^n\right)^T\left(\tilde{H}^{n-1}\right)^{-1}}{1+\left(X^n\right)^T\left(\tilde{H}^{n-1}\right)^{-1}X^n} \tag{35}$$

En prenant, à la première étape (n = 1), $\tilde{H}^0=\mu_k I$, on obtient, à l'étape N : $\left(\tilde{H}^N\right)^{-1}=\left[\tilde{H}+\mu_k I\right]^{-1}$.

• **Inversion directe.**

Plusieurs méthodes d'inversion directes existent. Étant donné que l'algorithme est itératif et que la procédure de recherche du pas nécessite souvent plusieurs inversions de matrice, on a intérêt à utiliser une méthode économique en nombre de calculs.

Le fait que l'approximation du Hessien augmentée de μ_k reste une matrice symétrique définie positive nous permet d'utiliser la méthode de Cholesky.

De la même façon que dans le cas de l'algorithme de BFGS, une recherche unidimensionnelle doit être appliquée pour la recherche d'un pas de descente et ceci à chaque itération de l'algorithme. Une stratégie communément utilisée [Bishop95, Walter94] consiste à appliquer la procédure suivante : soit $r > 1$ (généralement égal à 10) un facteur d'échelle pour μ_k. Au début de l'algorithme, on initialise μ_0 à une grande valeur ([Bishop95] propose 0.1). A l'étape k de l'algorithme :

- Calculer $J(\theta^k)$ avec μ_k déterminé à l'étape précédente.
- **Si** $J(\theta^k) < J(\theta^{k-1})$, **alors** accepter le changement de paramètres et diviser μ_k par r.
- **Sinon**, récupérer θ^{k-1} et multiplier μ_k par r. Répéter cette dernière étape jusqu'à ce qu'une valeur de μ_k correspondant à une décroissance de J soit trouvée.

Cet exemple de procédure présente l'avantage de nécessiter peu d'inversions de matrice à chaque itération de l'algorithme. En revanche, le choix du pas initial possède une influence sur la vitesse de convergence de l'algorithme.
Ces observations nous mènent à proposer la procédure suivante :

Au début de l'algorithme, initialiser μ_0 à une valeur positive quelconque. En effet ce choix n'a pas d'influence sur le déroulement de l'algorithme. A l'étape k de l'algorithme :

1. Calculer $J(\theta^k)$ avec le μ_k disponible (le dernier calculé).
2. Si $J(\theta^k) < J(\theta^{k-1})$, **alors** récupérer θ^{k-1}, diviser μ_k par r et aller à l'étape 1.
3. Sinon récupérer θ^{k-1} et multiplier μ_k par r. Répéter cette dernière étape jusqu'à ce qu'une valeur de μ_k correspondant à une décroissance de J soit trouvée.

Cette procédure permet de s'approcher de la méthode de Newton plus rapidement que la méthode précédente. En revanche, étant donné que plusieurs ajustements de paramètres sont testés, elle nécessite unn plus grand nombre d'inversions de matrice.

V.3 Commentaire.

Nous avons présenté dans cette partie les algorithmes du second ordre que nous utilisons dans ce mémoire (c'est à dire l'algorithme de BFGS et celui de Levenberg–Marquardt). La difficulté essentielle lors de l'application de l'algorithme de BFGS réside dans le choix de la condition de passage du gradient simple à la méthode de BFGS. Ce problème ne se pose pas pour l'algorithme de Levenberg–Marquardt, mais le volume de calculs nécessaires à chaque itération de cet algorithme croît rapidement avec le nombre de paramètres.

VI. CONCLUSION

Dans ce chapitre, nous avons présenté les principes de la modélisation "boîte noire", les étapes de la conception d'un tel modèles, ainsi que les fonctions paramétrées utilisables, et les algorithmes qu'il convient de mettre en œuvre pour l'ajustement des paramètres. Les deux chapitres suivants seront consacrés à la présentation et à la mise en œuvre des deux catégories de fonctions paramétrées que nous avons utilisées : les réseaux de neurones et les réseaux d'ondelettes.

CHAPITRE II

Réseaux de fonctions dorsales

I. INTRODUCTION.

Le présent chapitre est consacré à une catégorie de réseaux utilisés pour la modélisation non linéaire "boîte noire", à temps discret : les réseaux de fonctions dorsales (ridge function networks). Cette appellation provient de la forme géométrique des fonctions constituant ces réseaux. Dans ce mémoire, nous utilisons indifféremment les deux appellations “réseaux de neurones” ou “réseaux de fonctions dorsales” que nous considérons comme synonymes, par opposition aux réseaux d'ondelettes ou de fonctions radiales.

Nous présentons ici les neurones constituant ces réseaux, leurs propriétés ainsi que l’apprentissage des réseaux.

II. NEURONES FORMELS À FONCTIONS DORSALES ET RÉSEAUX.

II.1 Qu’est ce qu’un neurone formel ?

Un neurone formel est une fonction paramétrée, non linéaire, de plusieurs variables appelées *entrées* du neurone ; la valeur de la fonction est disponible en *sortie* du neurone. Par abus de langage, nous utiliserons parfois, le terme de "neurone linéaire" pour désigner une fonction linéaire ou affine.

II.2 Qu'est-ce qu'un neurone formel à fonction dorsale ?

Pour un neurone formel à fonction dorsale, le calcul de la fonction est effectué en deux étapes :

1. Calcul d'une somme pondérée des entrées. Le résultat obtenu est appelé *potentiel* du neurone. Il est donné par la relation suivante :

$$v_i = \sum_{j \in P_i} c_{ij}\, x_j \qquad (1)$$

où P_i est l’ensemble des indices $\{j\}$ des entrées x_j du neurone i. Les coefficients c_{ij} sont des coefficients de pondération des entrées du neurones appelés (pour des raisons historiques) *poids synaptiques* ou plus simplement *poids*.

2. Calcul d'une fonction non linéaire du potentiel, souvent appelée *fonction d’activation*. La sortie du neurone est alors la valeur de cette fonction, appelée parfois *activité du neurone* :

$$x_i = f(v_i) \qquad (2)$$

Ainsi, en tout point d'un hyperplan de l'espace des entrées défini par v_i = constante, la sortie du neurone est constante. Si le neurone est une fonction

de deux variables, les lignes de niveau de la sortie sont des droites parallèles, d'où le terme de *fonction dorsale*.

La sortie d'un neurone à fonction dorsale est non linéaire par rapport aux entrées *et par rapport aux coefficients* c_{ij}. Cette caractéristique est importante puisque, comme nous l'avons vu dans le chapitre précédent, elle est à l'origine de la propriété de parcimonie.

Il est commode de représenter graphiquement un neurone à fonction dorsale comme indiqué sur la Figure 1, où apparaissent les deux phases du calcul de la sortie.

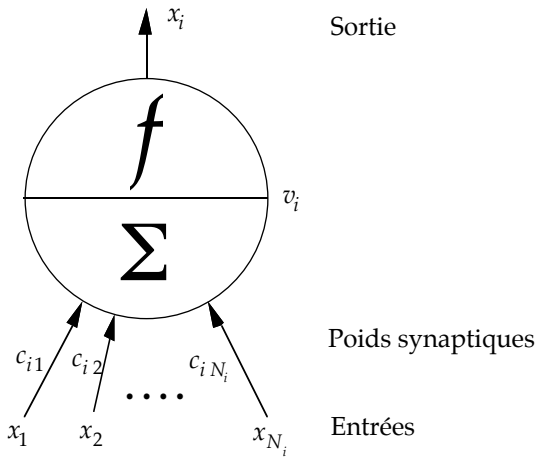


Figure 1. Représentation graphique d'un neurone.

Différentes fonctions d'activation sont envisageables. Cette question est discutée plus amplement, et des exemples sont présentés, dans le paragraphe III de ce chapitre.

II.3 Qu'est ce qu'un réseau de neurones ?

Un neurone étant une fonction non linéaire paramétrée, un réseau de neurones réalise une combinaison, elle-même paramétrée, de telles fonctions. On a coutume de représenter cette combinaison sous la forme de neurones, comme représentés sur la Figure 1, reliés entre eux par des connexions qui représentent les poids. On distingue conventionnellement deux types de neurones dans un réseau :

- *les neurones cachés*, caractérisés en ce que leurs sorties ne constituent pas les sorties du réseau, mais sont combinées par le (ou les) neurone(s) de sortie pour constituer celle(s)-ci : ils sont dits cachés parce que leur sortie n'est pas une sortie du réseau.

- *les neurones de sortie* combinent les sorties des neurones cachés pour constituer les sorties du réseau. Pour des réseaux destinés à la modélisation, les neurones de sortie sont généralement des "neurones linéaires" (leur fonction d'activation est l'identité) : ainsi, la sortie d'un réseau de neurones destiné à la modélisation statique de processus est *une combinaison linéaire paramétrée de fonctions non linéaires paramétrées des variables.*

II.4 Réseaux non bouclés et réseaux bouclés.

II.4.1 Les réseaux non bouclés.

Un réseau de neurones non bouclé, appelé aussi réseau *statique*, est un réseau dont le graphe des connexions est acyclique; il réalise une fonction algébrique non linéaire de ses entrées.
Comme nous l'avons indiqué au paragraphe précédent, on utilise généralement, pour la modélisation de processus, un réseau comprenant un neurone de sortie linéaire ; un tel réseau réalise donc *une combinaison linéaire paramétrée de fonctions non linéaires paramétrées des variables.*
Si ces dernières sont les valeurs, décalées d'une période d'échantillonnage, d'un même signal, un tel réseau constitue un filtre non linéaire transverse à temps discret.

II.4.2 Les réseaux bouclés.

Un réseau de neurones bouclé, appelé aussi réseau *dynamique*, est un réseau dont le graphe des connexions peut contenir des cycles. Dans un réseau à temps discret, un retard (entier positif ou nul) est associé à chaque connexion; pour que le réseau soit causal, tout cycle du graphe des connexions doit être tel que la somme des retards associés à chacune des connexions du cycle soit non nul. Un réseau bouclé à temps discret est régi par une équation aux différences récursive. Il constitue un filtre transverse non linéaire.

Il a été montré dans [Nerrand92, Dreyfus98] que tout réseau (statique ou dynamique) peut être mis sous une forme particulière, appelée *forme canonique*, qui est une représentation d'état minimale. Elle est constituée d'un réseau non bouclé, et de connexions de retard unité ramenant les sorties de ce réseau non bouclé vers les entrées de celui-ci. La forme canonique permet de mettre clairement en évidence un ensemble minimum de variables d'état, et, de plus, son utilisation facilite la mise en oeuvre de l'apprentissage du réseau.

Étant donné que, dans le cadre de ce mémoire, nous nous intéressons à la modélisation dynamique de processus, nous utiliserons, le plus souvent des réseaux bouclés.

II.5 Réseaux non bouclés complètement connectés et réseaux à couches.

On distingues deux familles de réseaux non bouclés en fonction de la topologie des connexions entre les neurones.

II.5.1 Les réseaux non bouclés complètement connectés.

Dans un réseau complètement connecté, chaque neurone reçoit les entrées du réseau et les sorties des neurones de numéro inférieur. La figure 2 illustre un réseau complètement connecté ayant N_i entrées, N_c neurones cachés et une sortie.

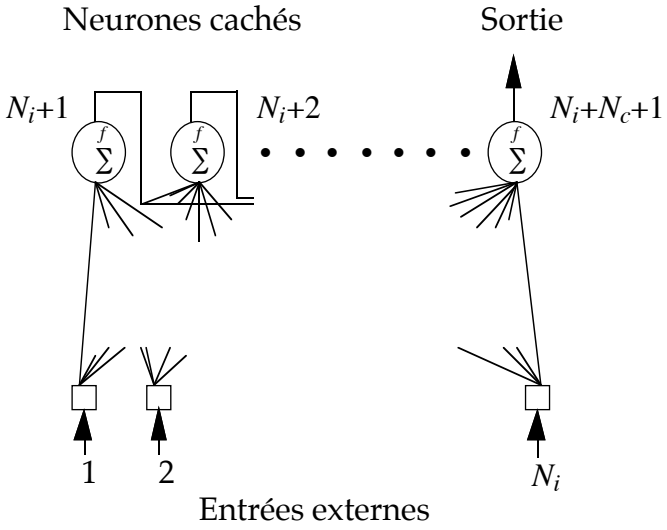


Figure 2. Réseau de neurones complètement connecté.

II.5.2 Les réseaux non bouclés à couches.

C'est sans doute l'architecture de réseau de neurones la plus répandue. De tels réseaux sont appelés perceptrons multi–couches (ou MLP pour Multi-Layer Perceptrons). Les neurones cachés sont organisés en une (ou parfois plusieurs) couches. Ils reçoivent leurs entrées des neurones de la couche précédente (ou des entrées du réseau s'il s'agit de la première couche de neurones cachés) et transmettent leur sortie à ceux de la couche suivante. La figure 3 illustre un réseau ayant N_i entrées, constitué d'une couche de N_c neurones cachés et d'une sortie.

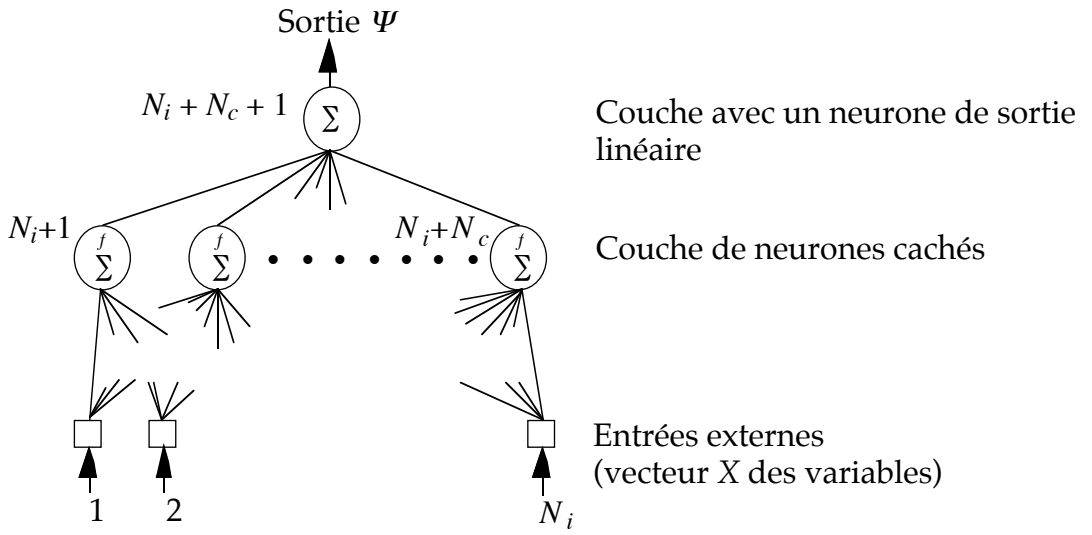


Figure 3. Réseau de neurones avec une couche de neurones cachés et un neurone de sortie linéaire.

Dans le cadre de ce mémoire, nous nous intéresserons uniquement aux réseaux de neurones possédant une seule couche cachée et un neurone de sortie linéaire. En effet, cette classe de réseaux possède la propriété d'approximation universelle que nous présenterons dans le paragraphe III de ce chapitre. La fonction réalisée par un tel réseau est bien du type défini par la relation (11) du chapitre précédent :

$$\psi(X) = \sum_{i=1}^{N_C} \theta_i \, \Phi_i(X, \Theta_i) \tag{3}$$

où chaque fonction Φ_i est réalisée par un neurone, où les θ_i sont les poids de la couche de connexions entre les sorties des neurones cachés et le neurone de sortie, et où les Θ_i sont les vecteurs des poids des connexions entre les entrées et le neurone caché i.

II.5.3 Les réseaux mis en œuvre dans ce travail.

Dans ce mémoire, et comme précisé plus haut dans ce même chapitre, nous utiliserons uniquement

- *pour la modélisation statique,* des réseaux possédant une seule couche de neurones cachés. Ce choix est motivé par le fait qu'une telle architecture possède la propriété d'approximation universelle.
- *pour la modélisation dynamique,* des réseaux tels que la partie statique de leur forme canonique est constituée d'un réseau à une couche cachée.

Le neurone de sortie est choisi avec une fonction d'activation identité. Ce choix permet de ne pas limiter les valeurs de la sortie à celle des bornes de la fonction d'activation.

Les réseaux que nous utiliserons possèdent une entrée constante (appelée aussi biais), reliée à tous les neurones (y compris le neurone de sortie).

D'autre part, nous utiliserons des connexions directes entre les entrées du réseau et le neurone de sortie (ces réseaux ne sont donc pas de purs réseaux à couches). L'expression analytique d'un tel réseau est donnée par la relation suivante :

$$\psi(x) = \sum_{j=1}^{N_c} c_j f\left(\sum_{k=0}^{N_i} c_{jk}\, x_k\right) + \sum_{k=0}^{N_i} a_k\, x_k \qquad (4)$$

Cette relation est un cas particulier de la relation (11) du chapitre précédent, où N_c fonctions Φ_i sont des fonctions dorsales, et N_i fonctions Φ_i sont les fonctions identité.

L'entrée d'indice k=0 correspond à l'entrée constante (x_0=0). La figure 4 illustre ce réseau.

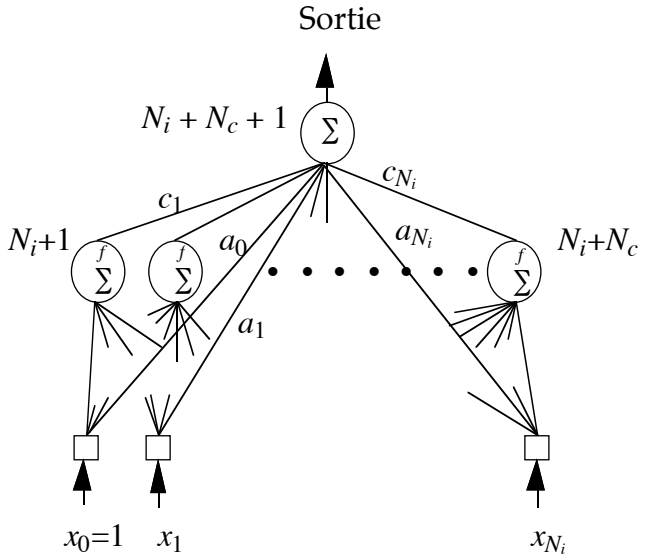


Figure 4. Réseau de fonctions dorsales non bouclé que nous utilisons comme modèle statique de processus.

III. CHOIX DE LA FONCTION D'ACTIVATION ET PROPRIÉTÉ D'APPROXIMATION UNIVERSELLE.

Comme indiqué dans le chapitre précédent, la propriété d'approximation universelle est une caractéristique très désirable pour une fonction paramétrée destinée à réaliser des modèles "boîte noire". Dans le cas des réseaux de fonctions dorsales, cette propriété dépend de la fonction d'activation choisie. Dans le cadre

de ce mémoire, nous nous intéressons à deux fonctions : la fonction tangente hyperbolique (sigmoïde) et la fonction gaussienne.

III.1 La fonction sigmoïde.

Il a été montré dans [Cybenko89] qu'un réseau de fonctions dorsales constitué d'une seule couche de neurones cachés et dont la fonction d'activation tend vers 0 en $-\infty$ et vers 1 en $+\infty$ possède la propriété d'approximateur universel. Par exemple, la fonction (à valeurs dans {0, 1}) définie par $\Phi(v)=\frac{1}{1+e^{\pm v}}$ remplit ces deux conditions. Dans [Funahashi89], la propriété est prouvée pour des fonctions strictement croissantes et bornées. A cet effet, la fonction la plus utilisée est la fonction tangente hyperbolique : $\Phi(v)=\frac{e^{v}\pm e^{\pm v}}{e^{v}+e^{\pm v}}$ (à valeurs dans {–1,1}).

Ces deux fonctions sont dérivables, ce qui, comme nous l'avons vu au chapitre précédent, est important pour l'apprentissage. Notons que cette fonction peut être vue comme une forme dérivable de la fonction signe(x) qui a été utilisée comme fonction d'activation des premiers neurones formels.

D'autre part, nous avons signalé plus haut que les réseaux de fonctions dorsales sont non linéaires par rapport à leurs paramètres ; ceci confère à ces réseaux la propriété de *parcimonie* [Barron93, Hornik94] : l'erreur d'approximation décroît comme l'inverse du nombre de fonctions sigmoïdes que contient le réseau et ceci quelque soit le nombre d'entrées. De ce fait, pour une précision désirée, le nombre de paramètres du réseau est proportionnel au nombre d'entrées. Par opposition, le nombre de paramètres est une fonction exponentielle du nombre d'entrées pour des réseaux linéaires par rapport aux paramètres (polynômes par exemple).

III.2 La fonction gaussienne.

La fonction d'activation gaussienne a été proposée dans [Girosi95] dans un contexte de généralisation des réseaux préalablement proposés par ces auteurs. Elle est définie par $\Phi(v)=e^{\pm v^2}$.

Du point de vue de la propriété d'approximation universelle, les réseaux de fonctions dorsales gaussiennes possèdent des propriétés équivalentes à celles des réseaux de fonctions sigmoïdes [Girosi95].

IV. APPRENTISSAGE DES RÉSEAUX DE FONCTIONS DORSALES.

IV.1 Apprentissage de réseaux non bouclés.

Étant donnée une fonction de régression à approcher ou un processus statique à modéliser à l'aide d'un réseau non bouclé de fonctions dorsales, la première étape consiste à choisir une architecture pour ce réseau. Puisque nous nous intéressons uniquement aux réseaux constitués d'une seule couche de neurones cachés, ce choix d'architecture revient au choix (ou à la détermination) du nombre de neurones cachés à considérer.

Une fois que le nombre de neurones cachés a été fixé, le réseau constitue une famille de fonctions paramétrées. La phase d'apprentissage du réseau consiste à trouver, parmi toutes ces fonctions, celle qui minimise la fonction de coût $J(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left(y_p^n \pm y^n(\Theta) \right)^2$; comme nous l'avons indiqué au chapitre I, cette fonction permet de mesurer l'écart entre les sorties du processus et celles du réseau aux points de la séquence d'apprentissage.

Comme nous l'avons indiqué plus haut, la sortie d'un réseau de fonctions dorsales est non linéaire par rapport aux poids synaptiques ; l'apprentissage doit donc être effectué en utilisant un algorithme de gradient comme ceux présentés au chapitre I.

Le vecteur gradient de la fonction de coût est calculé en utilisant l'algorithme de rétropropagation [Rumelhart86]. Cette implémentation judicieuse du calcul du gradient est intéressante dans la mesure où le nombre d'opérations à effectuer est moins important que dans le cas du calcul du gradient dans le sens direct. En effet, le calcul du gradient par rétropropagation consiste à décomposer la $(i,j)^{\text{ème}}$ composante du gradient partiel $\frac{\partial J^n}{\partial c_{ij}}$ en : $\frac{\partial J^n}{\partial c_{ij}} = \frac{\partial J^n}{\partial v_i} \frac{\partial v_i}{\partial c_{ij}} = \pm q_i x_j$ avec $\Theta_i = \left(\begin{array}{cccc} c_{i0} & c_{i1} & .... & c_{iN_i} \end{array} \right)^T$. Pour calculer le vecteur gradient de la fonction de coût, il suffit de connaître les N_C dérivées q_i. En revanche, le calcul du gradient de la fonction de coût dans le sens direct est basé sur la décomposition suivante : $\frac{\partial J^n}{\partial c_{ij}} = \frac{\partial J^n}{\partial e} \frac{\partial e}{\partial c_{ij}} = \pm (y_p \pm y) \frac{\partial y}{\partial c_{ij}}$. Le calcul de $\frac{\partial y}{\partial c_{ij}}$ se fait à partir des dérivées des sorties de tous les neurones qui influent sur le neurone de sortie, en partant de celles des entrées : $\frac{\partial x_j}{\partial c_{ij}}$ avec $j = 1$, ... , N_i. Dans cette dernière phrase apparaît un avantage de cette technique par rapport à la rétropropagation. En effet, la rétropropagation considère implicitement que les dérivées partielles par rapport aux entrées sont nulles. Si ce n'est pas le cas (en particulier pour les réseaux bouclés), il est

nécessaire d'avoir recours au calcul dans le sens direct. Nous discutons dans le chapitre III, une autre situation où le calcul dans le sens direct est plus approprié que celui par rétropropagation.

IV.2 Apprentissage de réseaux bouclés.

Nous avons indiqué plus haut que tout réseau de neurones bouclé à temps discret peut être mis sous une forme canonique constituée d'un réseau non bouclé dont les entrées sont les entrées externes et les variables d'état à l'instant n, et dont les sorties sont les sorties du réseau et les variables d'état à l'instant $n+1$. Cette mise en forme d'un réseau bouclé facilite le calcul du gradient de la fonction de coût, et ramène l'apprentissage d'un réseau bouclé à celui d'un réseau non bouclé, comme nous le verrons dans le chapitre III. Ainsi, qu'il s'agisse d'effectuer l'apprentissage de réseaux non bouclés ou celui de réseaux bouclés, on est toujours amené à minimiser une fonction de coût par l'une des méthodes de gradient décrites dans le chapitre I.

IV.3 Initialisation du réseau et minima locaux.

Pour que l'algorithme de rétropropagation "démarre", les paramètres du réseau doivent être initialisés à des valeurs non nulles. Cette étape d'initialisation est très importante car elle est susceptible de déterminer en partie le résultat obtenu en fin d'apprentissage, donc les performances du modèle ainsi conçu. En effet, des initialisations différentes peuvent conduire à trouver, dans l'espace de paramètres, des minima différents, donc des valeurs de paramètres différentes.

Dans les exemples étudiés dans ce mémoire, nous avons adopté une technique d'initialisation classique, qui consiste à initialiser les paramètres de telle sorte que, en début d'apprentissage, les valeurs des sorties des neurones cachés se situent dans les parties linéaires des sigmoïdes pour l'ensemble des séquences d'apprentissage. Pour chaque architecture de réseau, on effectue plusieurs apprentissages avec des initialisations différentes, et l'on retient le réseau qui correspond à la plus petite valeur de la fonction de coût.
Au demeurant, *pour les réseaux de fonctions dorsales,* des expériences numériques antérieures menées au laboratoire [Stoppi97] ont montré que le problème des minima locaux n'est pas dramatique : lorsque la fonction génératrice des données d'apprentissage est un réseau de neurones (le réseau "maître") ayant plus de cinq entrées, et que l'on effectue l'apprentissage d'un réseau de neurones de même architecture (le réseau "élève"), celui-ci, en fin d'apprentissage, est identique au réseau maître dans la majorité des cas. Nous verrons dans le chapitre III que la situation est très différente pour les réseaux d'ondelettes.

IV.4 Autres schémas d'apprentissage pour les réseaux de fonctions dorsales.

Le schéma d'apprentissage classique des réseaux de fonctions dorsales que nous utilisons dans ce mémoire consiste à choisir un nombre de neurones cachés et à effectuer ensuite l'ajustement de tous les poids synaptiques de tous les neurones simultanément.

Dans des tentatives d'apporter des réponses au problème des minima locaux, on trouve dans la littérature des propositions d'autres procédures d'apprentissage. Citons la procédure de l'apprentissage incrémental [Hirose91, Jutten95, Mohraz96] qui consiste à démarrer l'apprentissage avec un seul neurone et à en introduire d'autres au fur et à mesure que l'apprentissage progresse. L'inconvénient général de ces procédures réside dans le fait qu'elles aboutissent généralement à des réseaux largement sur-dimensionnés, donc non parcimonieux, car les erreurs, forcément importantes, commises au début avec un petit nombre de neurones, nécessitent, pour être corrigées, l'introduction d'un grand nombre de neurones.

Dans le même esprit, d'autres auteurs [Chentouf96] proposent une procédure d'apprentissage incrémental de réseaux où fonctions dorsales et fonctions radiales cohabitent.

La "*projection pursuit regression*" [Friedman81, Huber85, Hwang94] peut être considérée comme une procédure d'apprentissage particulière de réseaux de fonctions dorsales, dont l'originalité réside dans le fait que les fonctions d'activation des neurones ne sont pas prédéterminées (sigmoïde ou gaussienne) mais reconstruites à chaque itération de l'algorithme comme une somme de polynômes ou de fonctions splines.

V. ANALYSE D'UN RÉSEAU DE FONCTIONS DORSALES.

V.1 Principe.

A la fin de l'apprentissage d'un réseau, on peut se poser la question suivante : tous les poids synaptiques ou tous les neurones participent-ils effectivement à la fonction réalisée par le réseau ? Autrement dit, sont-ils tous utiles ? Suivant que l'on s'intéresse aux poids synaptiques ou aux neurones, la façon d'étudier la question est sensiblement différente. Les techniques permettant de répondre à cette question sont dites des *techniques d'élagage.*

V.2 Élagage de poids synaptiques.

On trouve dans la littérature consacrée aux réseaux de fonctions dorsales plusieurs procédures d'élagages. Des techniques telles que OBD (Optimal Brain Damage) [LeCun90] ou OBS (Optimal Brain Surgeon) [Hassibi93] commencent par

effectuer l'apprentissage d'un grand réseau. L'élagage des poids synaptiques est alors fondé sur la sensibilité de la fonction de coût à la variation des poids synaptiques : si la fonction de coût est peu sensible vis à vis d'un poids synaptique, celui-ci est supprimé et un nouvel apprentissage du réseau est effectué. En particulier, si un poids correspondant à la pondération de la sortie d'un neurone est supprimé, ce neurone est alors supprimé.

L'un des principaux inconvénients de ces méthodes et qu'un apprentissage est nécessaire à chaque suppression d'un poids. Ces techniques peuvent être considérées comme un moyen de déterminer l'architecture "minimale" pour obtenir une performance désirée.

D'autres procédures d'élagage de poids synaptiques existent et sont résumées dans [Reed93].

Une méthode de suppression de neurones cachés a été proposée dans [Stoppi97], pour des réseaux à couches avec un neurone linéaire de sortie. La méthode consiste à ajouter un neurone caché dont la sortie est aléatoire. On classe les entrées de ce modèle (sorties des neurones cachés) par ordre de pertinence décroissant par la technique classique d'orthogonalisation de Gram-Schmidt, et l'on supprime tout neurone caché dont la sortie est classée après le neurone aléatoire.

V.3 Une procédure pour la détection de neurones à fonctions gaussiennes "*mal utilisés*".

Lors de l'apprentissage de réseaux de fonctions dorsales gaussiennes, il arrive que les coefficients synaptiques d'un neurone deviennent très grands. Le potentiel v étant très grand, cela a pour effet de rendre la sortie du neurone nulle presque partout ; ce neurone est alors mal utilisé, car il ne participe pas (ou très peu) à la fonction réalisée par le réseau.

Dans [Girosi95], les réseaux de fonctions dorsales gaussiennes sont utilisés avec un algorithme d'apprentissage non déterministe [Caprile90]. Cet algorithme d'apprentissage (appelé "random step" algorithm) effectue un ajustement aléatoire des coefficients suivant la procédure suivante : on effectue un tirage aléatoire des adaptations à apporter à chacun des coefficients dans un intervalle $[\alpha, \beta]$ de longueur préalablement choisie. Si cet ajustement aboutit à une décroissance du coût, on le retient et on double la longueur de l'intervalle. Sinon, il est rejeté et on diminue de moitié la longueur de $[\alpha, \beta]$. Le fait que l'on contrôle la longueur de l'intervalle permet de contrôler l'ajustement qu'on apporte aux coefficients.

En ce qui nous concerne, nous sommes confrontés à ce problème de neurones mal utilisés parce que nous utilisons des algorithmes d'optimisation non linéaire

qui n'imposent aucune contrainte sur les valeurs que peuvent prendre les coefficients du réseau.

Ce phénomène se manifeste très souvent par une erreur quadratique moyenne calculée sur l'ensemble de performance très supérieure à celle calculée sur l'ensemble d'apprentissage sans qu'il y ait un réel surajustement. Ceci est dû au fait que, si un exemple de la séquence de test appartient à la partie non nulle de la gaussienne mal utilisée, il introduit un terme important dans l'EQMP. La figure 5 illustre un exemple à une dimension.

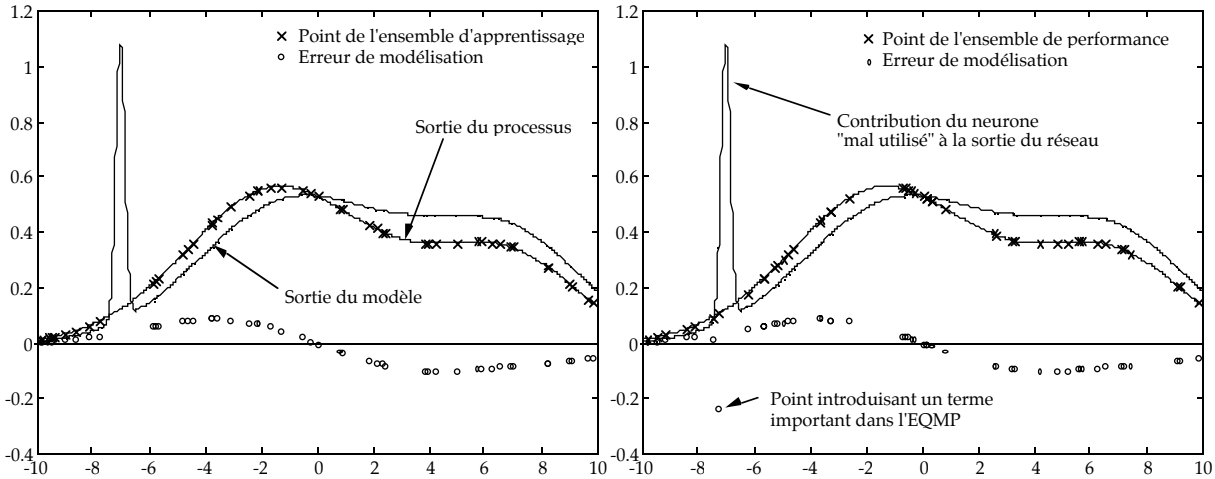


Figure 5. Illustration d'un exemple de neurone mal utilisé par le réseau : une des gaussiennes est très "pointue", et un exemple de l'ensemble d'estimation de la performance se trouve précisément dans le pic de cette gaussienne.

Nous nous sommes donc intéressés plus particulièrement au traitement de ces neurones plutôt qu'à celui de poids synaptiques considérés individuellement. Cet intérêt ne s'inscrit pas dans la perspective de déterminer le nombre "optimal" de neurones pour obtenir une performance désirée mais plutôt dans l'objectif de "rentabiliser" tous les neurones présents dans un réseau afin d'obtenir la meilleure performance possible avec une architecture donnée.

Une première approche consisterait à mettre au point une procédure qui, comme les techniques d'élagage de poids synaptiques, interviendrait en fin d'apprentissage et permettrait de détecter ces neurones et de les supprimer. Une telle approche n'est pas une réponse à notre problème étant donné qu'elle ne permet pas de mettre à profit ces neurones.

Nous proposons de détecter automatiquement ces neurones pendant la phase d'apprentissage, et à les "réinitialiser" afin de les réintégrer dans le processus d'apprentissage.

Voici la procédure proposée :

• Tester les valeurs des poids synaptiques de chacun des neurones.
• **Si** aucun neurone mal utilisé n'est détecté : ne rien faire et aller à l'itération suivante de l'algorithme d'apprentissage.
• **Sinon**, réinitialiser ces neurones de manière adéquate et aller à l'itération suivante.

Les étapes de détection et de réinitialisation des neurones se déroulent de la manière suivante :

1ère étape : Détection automatique des neurones "mal utilisés".

Étant donné que l'on considère qu'un neurone est "mal utilisé" si la fonction qu'il réalise a un écart-type trop petit, on propose de fonder la détection de ces neurones sur l'analyse de leurs poids synaptiques qui sont équivalents à l'inverse d'un écart-type. En effet, l'expression de la sortie d'un neurone j étant $c_j \exp \pm\left(\sum_{i=1}^{N_i} c_{ji} x_i \right)^2$, peut être écrite de la manière suivante : $c_j \exp \pm\left(\sum_{i=1}^{N_i} \left(\frac{x_i}{1/c_{ji}} \right) \right)^2$ où les $\frac{1}{c_{ji}}$ sont les largeurs de la gaussienne dans chacune des directions i définies par les entrées.

Le potentiel du neurone caché j est donné par :

$$v_j = \sum_{k=1}^{N_i} c_{jk} x_k \tag{6}$$

Nous désignons par S_j la somme des valeurs absolues de tous les poids du neurone j :

$$S_j = \sum_{k=1}^{N_i} \mid c_{jk} \mid \tag{7}$$

On compare cette quantité à un seuil S préalablement choisi :

Si $S_j < S$ on considère que le neurone est bien utilisé par le réseau et on intervient donc pas. L'apprentissage continue.
Sinon on considère que le neurone est mal utilisé et on le réinitialise.

A chaque itération de l'algorithme d'apprentissage, ce test est appliqué à tous les neurones cachés du réseau.

À l'heure actuelle, le choix de la valeur du seuil S est effectué de manière empirique. Une grande valeur du seuil rend la procédure peu efficace puisque des

neurones mal utilisés peuvent échapper au test de réinitialisation. Inversement, une petite valeur de S aboutit à une réinitialisation de plusieurs neurones à chaque test, de sorte que l'apprentissage ne s'arrête pas.

2ème étape : Réinitialisation d'un neurone "mal utilisé".

Les neurones "mal utilisés" ayant été détectés sont comme en début d'apprentissage : leurs poids synaptiques sont initialisés à de petites valeurs aléatoires, suivant une distribution uniforme (voir paragraphe IV.3).

Une attention particulière doit être portée au cas des coefficients de pondération des sorties de ces neurones. En effet, étant donné que ces neurones participaient peu au réseau, leur réinitialisation peut introduire une augmentation du coût en cours d'apprentissage. Pour éviter cette situation, on initialise ces pondérations à de petites valeurs de telle sorte qu'à la reprise de l'apprentissage leur contribution soit très faible.

V.4 Étude d'un exemple.

Afin de mettre en œuvre la procédure décrite plus haut sur un exemple, on se propose d'approcher la fonction sinus cardinal sur l'intervalle $[-7,7]$ à l'aide d'un réseau de quatre fonctions dorsales gaussiennes comme celui de la figure 4. L'ensemble d'apprentissage est constituée de 50 exemples tirés suivant une distribution uniforme sur l'intervalle considéré. Afin d'estimer correctement la performance, on choisit 500 exemples régulièrement espacés sur l'intervalle considéré.

L'apprentissage débute par une phase de gradient simple sur 500 itérations puis une phase de gradient de second ordre (algorithme de BFGS). L'apprentissage s'arrête lorsque au moins un des critères d'arrêt est satisfait (ces critères sont énoncés dans le paragraphe V.2.2 du chapitre I).

Afin de mettre en évidence l'apport de la procédure proposée, on effectue dans une première étape dix apprentissages du réseau avec autant d'initialisations différentes. Dans une seconde étape, on fait autant d'apprentissages en détectant cette fois les neurones "mal utilisés" et en les réinitialisant.

Le tableau 1 illustre les résultats des apprentissages sans utilisation de la procédure.

Apprentissage	EQMA	EQMP
1er	$1.6\ 10^{-6}$	$2.3\ 10^{-5}$
2ème	$1.8\ 10^{-6}$	$1.6\ 10^{-5}$
3ème	$3.0\ 10^{-3}$	$7.2\ 10^{-3}$
4ème	$4.0\ 10^{-7}$	$2.1\ 10^{-6}$
5ème	$1.1\ 10^{-6}$	$5.7\ 10^{-6}$
6ème	$1.8\ 10^{-7}$	$1.7\ 10^{-5}$
7ème	$3.5\ 10^{-7}$	$9.7\ 10^{-6}$
8ème	$1.9\ 10^{-7}$	$9.7\ 10^{-6}$
9ème	$5.0\ 10^{-3}$	$5.8\ 10^{-3}$
10ème	$4.1\ 10^{-8}$	$4.7\ 10^{-5}$

Tableau 1. Résultats des apprentissages sans application de la procédure.

Hormis deux apprentissages, la performance est de l'ordre de 10^{-5} ou de 10^{-6}. D'autre part, on constate souvent un rapport de l'ordre de 10 ou plus entre le critère de l'apprentissage et celui d'évaluation de la performance.

Le tableau 2 présente les résultats des apprentissages (utilisant les même initialisations des coefficients du réseau) au cours desquels la procédure a été utilisée avec un seuil $S = 1$.

La quatrième colonne du tableau indique le nombre de fois où la procédure a détecté un ou plusieurs neurones "mal utilisés". Après chaque intervention de la procédure, l'algorithme d'apprentissage repart avec une phase de gradient simple avant celle de gradient de second ordre.

Apprentissage	EQMA	EQMP	Procédure (s)
1er	$1.6\ 10^{-7}$	$3.2\ 10^{-7}$	5
2ème	$1.6\ 10^{-7}$	$3.0\ 10^{-7}$	4
3ème	$1.1\ 10^{-7}$	$6.8\ 10^{-7}$	7
4ème	$1.6\ 10^{-9}$	$1.8\ 10^{-8}$	3
5ème	$2.4\ 10^{-9}$	$9.3\ 10^{-9}$	4
6ème	$3.6\ 10^{-9}$	$1.2\ 10^{-8}$	2
7ème	$6.1\ 10^{-9}$	$2.3\ 10^{-8}$	5
8ème	$1.2\ 10^{-7}$	$2.3\ 10^{-7}$	7
9ème	$5.8\ 10^{-3}$	$9.1\ 10^{-3}$	1
10ème	$1.3\ 10^{-7}$	$2.6\ 10^{-7}$	3

Tableau 2. Résultats des apprentissages avec application de la procédure.

Lors de la mise en oeuvre de cette procédure, on constate qu'en début d'apprentissage il arrive très souvent qu'un ou plusieurs neurones soient réinitialisés. Au fur et à mesure que l'algorithme progresse, le nombre de réinitialisations diminue et en fin d'apprentissage la procédure n'intervient plus.

Une comparaison des deux tableaux montre que les performances obtenues avec des apprentissages utilisant la procédure sont très souvent meilleures que celle obtenues avec un apprentissage classique. D'autre part, la différence entre l'EQMA et l'EQMP est souvent plus petite. Cela montre que la procédure apporte effectivement une meilleure utilisation des neurones et évite l'apparition de surapprentissage. On voit néanmoins sur l'exemple du 9ème apprentissage que, l'utilisation de la procédure n'améliore pas toujours la performance du réseau.

VI. MODÉLISATION DYNAMIQUE DE PROCESSUS À L'AIDE DE RÉSEAUX DE FONCTIONS DORSALES.

Les réseaux de fonctions dorsales et particulièrement de fonctions sigmoïdes ont été très étudiés comme outils de modélisation dynamique boîte noire. On trouve dans la littérature les deux approches classique de la modélisation de processus abordées : modélisation entrée–sortie [Narendra90, Nerrand92] et modélisation d'état [Levin92, Rivals96].

VI.1 Modélisation entrée–sortie.

Dans un contexte de modélisation non linéaire dynamique de processus à l'aide de réseaux de fonctions dorsales, on peut être amené à effectuer l'apprentissage d'un réseau bouclé ou non bouclé. Le choix de l'architecture dépend de l'hypothèse faite sur l'existence ou non d'un bruit qui agit sur le processus, et s'il existe, de la manière avec laquelle il agit.

VI.1.1 Prédicteurs non bouclé.

Un prédicteur entrée-sortie non bouclé aura pour expression :

$$y(n) = \psi\left(y_p(n\pm1), \ldots, y_p(n\pm N_s), u(n\pm1), \ldots, u(n\pm N_e)\right) \tag{8}$$

où ψ est une fonction réalisée à l'aide d'un réseau de fonctions dorsales comme celui de la figure 4. Le vecteur des entrées est constitué par les N_s valeurs de la sortie et les N_e de l'entrée externe. Nous avons donc N_i=N_e+N_s. Ce réseau est représenté sur la figure 6.

L'apprentissage d'un prédicteur non bouclé s'appuie sur une approche identique à celle adoptée pour des réseaux pour modélisation statique.

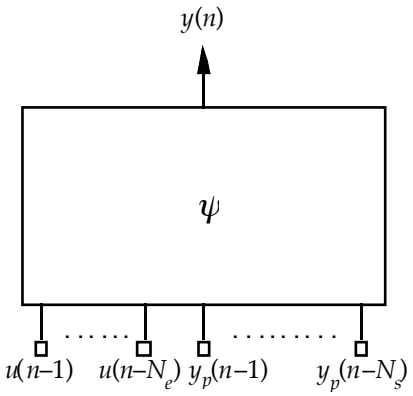


Figure 6. Réseau entrée-sortie non bouclé.

VI.1.2 Prédicteur bouclé.

Un prédicteur entrée-sortie bouclé aura pour expression :

$$y(n) = \psi\left(y(n\pm1), y(n\pm2), \ldots, y\left(n\pm N_s\right), u(n\pm1), \ldots, u\left(n\pm N_e\right)\right) \tag{9}$$

où ψ est une fonction réalisée à l'aide d'un réseau de fonctions dorsales comme celui de la figure 4. Ce réseau est illustré par la figure suivante :

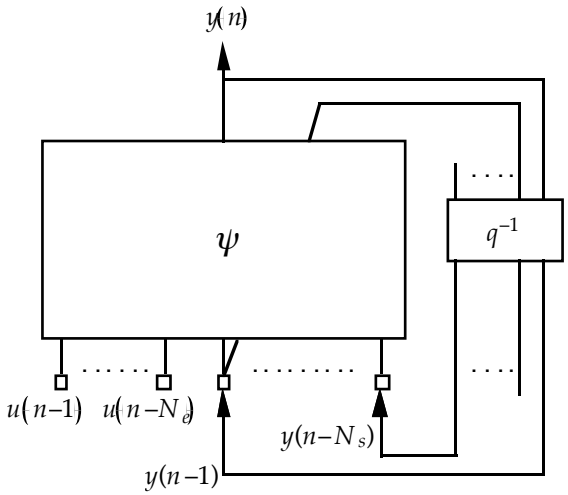


Figure 7. Réseau entrée-sortie bouclé.

L'apprentissage d'un prédicteur bouclé diffère du cas précédent par le fait que le calcul du gradient de la fonction de coût est rendu plus compliqué étant donné que les entrées d'état dépendent des poids synaptiques du réseau. Ce calcul est présenté dans [Nerrand93].

VI.2 Modélisation d'état.

Un prédicteur d'état neuronal a pour expression :

$$\begin{cases} x(n) = \psi_1(x(n\pm1), u(n\pm1)) \\ y(n) = \psi_2(x(n\pm1), u(n\pm1)) \end{cases} \tag{10}$$

où $x(n)$ et le vecteur d'état du modèle à l'instant n. ψ_1 et ψ_2 sont deux fonctions, la première vectorielle et la seconde scalaire réalisée à l'aide d'un réseau de fonctions dorsales. Celui-ci est illustré par la figure suivante :

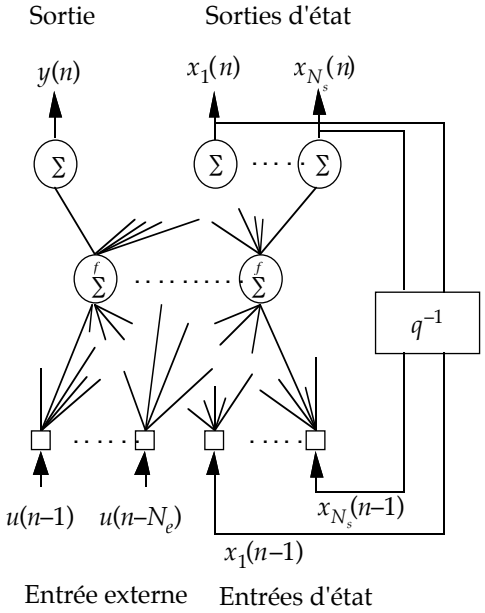


Figure 8. Réseau d'état bouclé.

Le calcul du gradient de la fonction de coût pour l'apprentissage des réseaux d'état fondés sur des fonctions dorsales est présenté dans [Rivals95a].

Nous considérons également des réseaux d'état pour lesquels la sortie est l'un des états du modèle. Un exemple est présenté dans le paragraphe III.4 du chapitre V.

VII. CONCLUSION.

En raison de leurs propriétés d'approximation universelle et de parcimonie, les réseaux de fonctions dorsales sont bien adaptés à la modélisation non linéaire de processus, aussi bien entrée–sortie que d'état. Ils peuvent constituer des outils de modélisation non linéaire, statique ou dynamique, très efficaces.

Le prix à payer réside dans le fait que, la sortie n'étant pas linéaire par rapport aux paramètres, l'estimation de ceux-ci exige la minimisation itérative de fonctions de coût qui possèdent des minima locaux. Cependant, cet inconvénient peut être aisément surmonté en effectuant plusieurs apprentissages, ce qui est rendu possible par l'existence d'algorithmes de minimisation très efficaces.

CHAPITRE III

Réseaux d'ondelettes (approche fondée sur la transformée continue)

I. INTRODUCTION.

Le terme *ondelette* désigne une fonction qui oscille pendant un " temps donné" (si la variable est le temps) ou sur un intervalle de longueur finie (si la variable est de type spatial). Au delà, la fonction décroît très vite vers zéro.

Historiquement, les premières ondelettes (introduites par Haar dans les années 1930) constituaient une base de fonctions orthogonales. Les ondelettes de Haar présentent la particularité de ne pas être dérivables.

Plus récemment, de nouvelles fonctions ondelettes ont été introduites [Meyer85, Meyer90], qui constituent également une base de fonctions orthogonales, et qui, de plus, sont dérivables. Elles ont été notamment mises en œuvre dans le cadre de l'analyse multirésolution de signaux [Mallat89]. Ces ondelettes ne peuvent s'exprimer sous une forme analytique simple. Pour cette raison, elles sont peu adaptées pour l'approximation de fonctions. Nous n'utiliserons donc pas les ondelettes orthogonales dans ce mémoire.

Les structures obliques (*frames* en anglais) ont été introduites par J. Morlet dans le but de trouver des bases de fonctions (non nécessairement orthogonales) pour représenter des signaux. Ces structures obliques ont été ensuite l'objet des travaux de I. Daubechies [Daubechies90] qui a développé un support théorique aux résultats de J. Morlet. Les structures obliques ont des expressions analytiques simples, et toute fonction de carré sommable peut être approchée, avec la précision voulue, par une somme finie d'ondelettes issues d'une structure oblique. Cette propriété est équivalente à celle de l'approximation universelle pour les réseaux de fonctions dorsales. Pour toutes ces raisons, nous nous sommes intéressés uniquement, dans notre travail, à des structures obliques d'ondelettes.

Dans ce chapitre, nous présentons tout d'abord les fonctions ondelettes et la transformée en ondelettes. Deux approches sont à considérer : la transformée en ondelettes continue et la transformée en ondelettes discrète, comme illustré par la Figure 1.

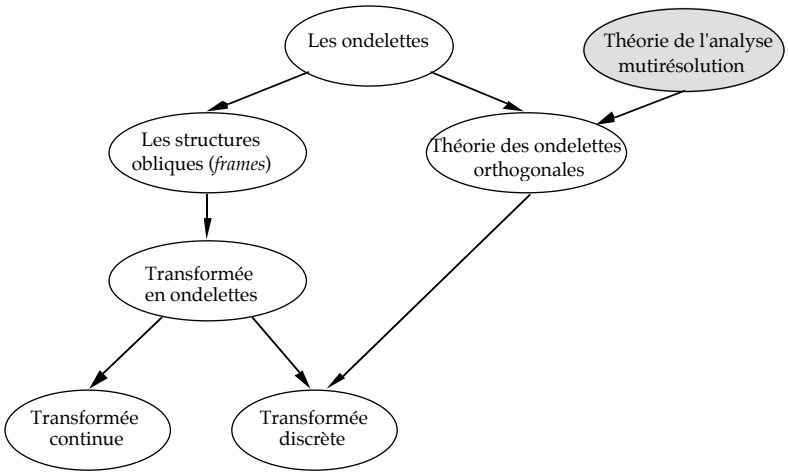


Figure 1.

Le présent chapitre est consacré aux ondelettes utilisées pour la transformée continue et aux réseaux de telles ondelettes. Nous décrivons en détail la technique de modélisation statique par réseaux d'ondelettes, et nous introduisons la modélisation dynamique par ces réseaux ; nous montrons qu'il est possible de considérer soit des réseaux entrée-sortie, soit des réseaux d'état.

II. RÉSEAUX ISSUS DE LA TRANSFORMÉE EN ONDELETTES CONTINUE.

De manière analogue à la théorie des séries de Fourier, les ondelettes sont principalement utilisées pour la décomposition de fonctions. La décomposition d'une fonction en ondelettes consiste à l'écrire comme une somme pondérée de fonctions obtenues à partir d'opérations simples effectuées sur une fonction principale appelée *ondelette–mère*. Ces opérations consistent en des translations et dilatations de la variable. Selon que ces translations et dilatations sont choisies de manière continue ou discrète, on parlera d'une transformée en ondelettes continue ou discrète.

II.1 La transformée en ondelettes continue.

Une transformée en ondelettes est dite continue lorsque les paramètres structurels des fonctions utilisées (c'est-à-dire les translations et les dilatations) peuvent prendre n'importe quelle valeur de l'ensemble des réels $\boldsymbol{R}$ (les dilatations doivent néanmoins être positives).

Soit ϕ une ondelette–mère, x la variable, m_j le paramètre de translation et d_j le paramètre de dilatation. L'ondelette ϕ_j de la famille de ϕ ayant pour paramètres m_j et d_j a pour expression :

$$\phi_j(x) = \frac{1}{\sqrt{d_j}}\,\phi\left(\frac{x \pm m_j}{d_j}\right) \qquad (1)$$

avec $m_j \in \boldsymbol{R}$ et $d_j \in \boldsymbol{R}^*_+$.

On constitue ainsi une famille d'ondelettes engendrée à partir de l'ondelette-mère. On la note Ω. On a alors la définition suivante :

$$\boldsymbol{\Omega} = \left\{ \frac{1}{\sqrt{d_j}} \phi\left(\frac{x \pm m_j}{d_j}\right), m_j \in R \text{ et } d_j \in R^*_+ \right\} \tag{2}$$

Comme les réseaux d'ondelettes auxquels nous allons nous intéresser sont issus de la transformée en ondelettes continue, nous allons présenter brièvement celle–ci.

Soient f et g deux fonctions ; on définit leur produit scalaire par l'intégrale suivante :

$$<f, g> = \int_R f(x)\, g(x)\, dx \tag{3}$$

Pour que la transformée en ondelettes d'une fonction existe, il faut que cette fonction appartienne à l'ensemble des fonctions de carré sommable que l'on note par $L^2(\boldsymbol{R})$. Autrement dit, il faut que son carré soit fini. Cette condition se traduit par :

$$\int_R f^2(x)\, dx < \infty \tag{4}$$

Dans ces conditions, la transformée en ondelette continue de la fonction f est définie comme le produit scalaire de f et de ϕ [Cohen96] :

$$W(m, d) = \frac{1}{\sqrt{d}} \int_R f(x)\, \phi\left(\frac{x \pm m}{d}\right) dx \tag{5}$$

La famille Ω doit constituer une *structure oblique* de l'ensemble $L^2(\boldsymbol{R})$ et y être dense. Cette propriété est assurée par l'existence de deux constantes $c > 0$ et $C < \infty$ telles que, pour toute fonction f pour laquelle il existe une transformée en ondelettes, on ait l'inégalité suivante :

$$c\,|f|^2 \le \sum_{\phi_j \in \Omega} \left| < \phi_j, f > \right|^2 \le C\,|f|^2 \tag{6}$$

De ce fait, toute combinaison linéaire d'un nombre fini d'éléments de la famille Ω est dense dans $L^2(\boldsymbol{R})$. Ceci garantit également que cette famille de fonctions possède la propriété d'approximation universelle définie dans le chapitre I du présent mémoire [Zhang92].

La reconstruction de la fonction f à partir de sa transformée est possible dans le cas où l'intégrale suivante est convergente :

$$C_\phi = \int_0^\infty \frac{|\hat{\phi}(\omega)|^2}{\omega} d\omega \tag{7}$$

où $\hat{\phi}$ est la transformée de Fourier de ϕ. Cette dernière condition est également appelée critère d'admissibilité pour une ondelette. Dans ce cas, f peut être reconstruite à partir de la relation suivante :

$$f(x) = \frac{1}{C_\phi} \int_{\mathbb{R}} \int_{\mathbb{R}_+} W(m, d) \frac{1}{\sqrt{d}} \phi\left(\frac{x \pm m}{d}\right) \mathrm{d}d \, \mathrm{d}m \tag{8}$$

La condition (7) est très intéressante dans la mesure où elle donne des informations sur les propriétés que doit vérifier une ondelette mère (si l'on souhaite que la reconstruction de la fonction transformée soit possible). En particulier, on doit avoir $\phi(0)= 0$. En remplaçant ω par 0 dans la définition de la transformée de Fourier de ϕ, on voit que cette condition est équivalente à :

$$\int_R \phi(x)\, dx = 0 \tag{9}$$

Donc, une ondelette est une fonction à support de longueur finie et d'intégrale nulle. Ainsi, les gaussiennes radiales ne peuvent pas être considérées comme des ondelettes.

II.2 De la transformée inverse aux réseaux d'ondelettes.

La relation (8) donne l'expression d'une fonction f de carré sommable sous la forme d'une intégrale sur toutes les dilatations et toutes les translations possibles de l'ondelette mère. Supposons que l'on ne dispose que d'un nombre fini N_w d'ondelettes ϕ_j obtenues à partie de l'ondelette mère ϕ. On peut alors considérer la relation

$$f(x) \approx \sum_{j=1}^{N_w} c_j \, \phi_j(x) \tag{10}$$

comme une approximation de la relation (8). La somme finie de la relation (10) est donc une approximation d'une transformée inverse. Elle peut être vue aussi comme la décomposition d'une fonction en une somme pondérée d'ondelettes, où chaque poids c_j est proportionnel à $W(m_j, d_j)$. Si l'on cherche à réaliser une approximation d'une fonction définie sur un domaine fini (donc de carré sommable), la transformée en ondelette de cette fonction existe, et sa reconstruction est possible.

Dans le cadre de la modélisation boîte noire, la fonction que l'on veut approcher (la fonction de régression de la grandeur à modéliser) n'est pas connue : on ne dispose que des points de mesure, en nombre fini. On peut alors chercher à obtenir une approximation de la fonction de régression sous la forme (10), où les coefficients c_j, ainsi que les paramètres m_j et d_j des ondelettes, doivent être estimés à partir des données disponibles.

C'est dans cette perspective qu'a été proposée l'idée de réseaux d'ondelettes. Ces réseaux ont été introduits pour la première fois à la même époque dans [Zhang92, Pati93].

III. DÉFINITION DES ONDELETTES MULTIDIMENSIONNELLES ET DES RÉSEAUX D'ONDELETTES.

III.1 Ondelettes multidimensionnelles.

Dans le paragraphe précédent, nous avons présenté les ondelettes à une dimension. Dans le cadre de la modélisation, il est fréquent d'avoir affaire à des processus multivariables ; il est donc utile d'introduire la notion d'ondelette multidimensionnelle.

On peut définir une ondelette multidimensionnelle comme le produit d'ondelettes monodimensionnelles : on dit alors que les ondelettes sont séparables. Dans ce cas, l'expression d'une ondelette multidimensionnelle est :

$$\Phi_j(x) = \prod_{k=1}^{N_i} \phi(z_{jk}) \quad \text{avec} \quad z_{jk} = \frac{x_k \pm m_{jk}}{d_{jk}} \tag{11}$$

où x_k est la k-ième composante du vecteur d'entrée x, et z_{jk} la composante centrée par m_{jk} et dilatée d'un facteur d_{jk}. Il a été montré dans [Kuga95] que ces ondelettes multidimensionnelles sont des structures obliques de $L^2(\boldsymbol{R}^{Ni})$.

III.2 Réseaux d'ondelettes.

Dans le présent travail, nous considérons des réseaux d'ondelettes de la forme suivante :

$$y = \psi(x) = \sum_{j=1}^{N_w} c_j \Phi_j(x) + \sum_{k=0}^{N_i} a_k x_k \quad \text{avec} \quad x_0=1 \tag{12}$$

où y est la sortie du réseau et $x = \{x_1, x_2, ..., x_{N_i}\}$ le vecteur des entrées ; il est souvent utile de considérer, outre la décomposition en ondelettes proprement dite, que la sortie peut avoir une composante affine par rapport aux variables, de coefficients a_k (k = 0, 1, ... , N_i). Pour la simplicité de l'exposé, nous ne considérerons que des réseaux à une sortie ; la généralisation à des réseaux à plusieurs sorties ne présente pas de difficulté.

Par analogie avec les réseaux de fonctions dorsales discutés dans le chapitre II, on peut représenter une ondelette de manière analogue à un neurone, comme indiqué sur la figure 2.

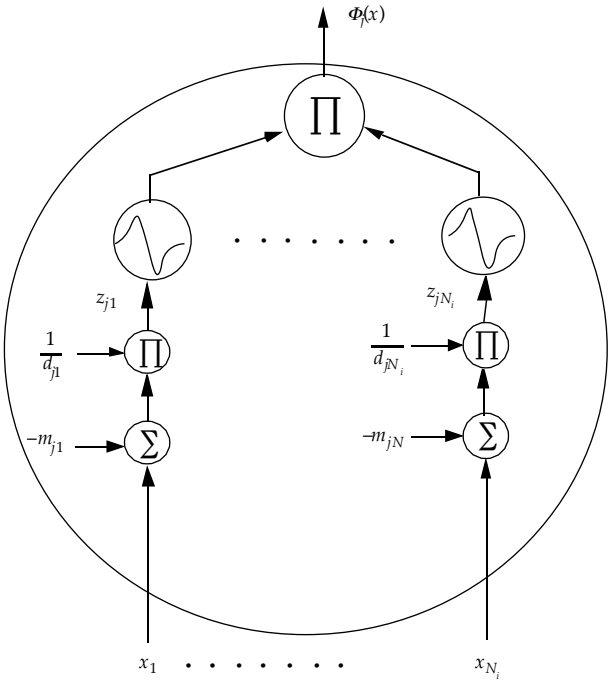


Figure 2. Représentation graphique d'une ondelette multidimensionnelle séparable.

Le réseau peut être considéré comme constitué de trois couches. Une première couche avec N_i entrée(s), une couche cachée constituée par N_w ondelettes et un sommateur (ou neurone linéaire) de sortie recevant les sorties pondérées des ondelettes multidimensionnelles et la partie affine. Ce réseau est illustré par la figure 3.

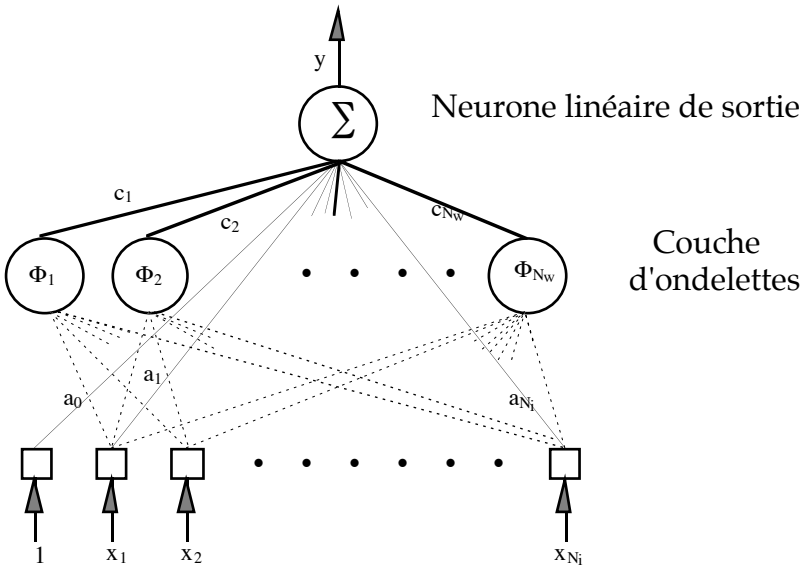


Figure 3. Représentation graphique d'un réseau d'ondelettes.

Ainsi, en se référant à la classification faite dans le paragraphe IV du chapitre I, les réseaux d'ondelettes sont des réseaux de fonctions non linéaires paramétrées, où le vecteur Θ_i est constitué par les translations et les dilatations de l'ondelette multidimensionnelle.

Plusieurs choix d'ondelettes sont possibles. En effet, plusieurs familles d'ondelettes existent. Les ondelettes les plus connues (et aussi les plus anciennes) sont certainement celles qui constituent le système de Haar que l'on présentera dans le chapitre suivant, dans le contexte d'ondelettes orthogonales. Les fonctions du système de Haar n'étant pas dérivables, il n'est pas possible d'appliquer aux réseaux de telles ondelettes les algorithmes d'estimation des paramètres présentés dans le chapitre I.

Les ondelettes que nous allons utiliser pour la construction de réseaux sont celles issues des travaux de I. Daubechies. On parle dans ce cas d'ondelettes de la famille de Daubechies. Ces fonctions sont dérivables et possèdent la propriété d'approximation universelle en vertu de la relation (6).

Une ondelette-mère que nous utilisons dans ce mémoire et que l'on retrouve dans [Zhang92] est la dérivée première de la fonction gaussienne. C'est l'une des ondelettes les plus utilisées [Torré95]. Elle est définie par :

$$\phi(x) = \pm x\, e^{-\frac{1}{2}x^2} \quad \text{et} \quad \phi_j(x) = \pm \frac{1}{\sqrt{d_j}} \left(\frac{x \pm m_j}{d_j} \right) \exp\left(\pm \frac{1}{2} \left(\frac{x \pm m_j}{d_j} \right)^2 \right) \tag{13}$$

Le graphe de cette fonction est représenté sur la figure suivante :

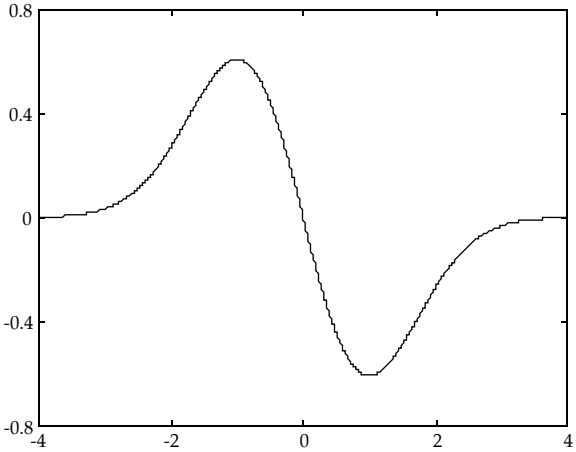


Figure 4. Graphe d'une ondelette.

Cette ondelette peut être considérée comme une forme dérivable des ondelettes du système de Haar, comme la tangente hyperbolique utilisée comme fonction d'activation des réseaux de neurones (présentés au chapitre II de ce mémoire) est une forme dérivable de la fonction signe.

Une autre ondelette-mère que l'on rencontre souvent dans la bibliographie (par exemple dans [Cannon95, Baron97]) est la dérivée seconde de la fonction gaussienne. Son expression est :

$$\phi(x) = \left(x^2 \pm 1\right) e^{-\frac{1}{2}x^2} \tag{14}$$

Elle est appelée "ondelette chapeau mexicain". Son graphe est le suivant :

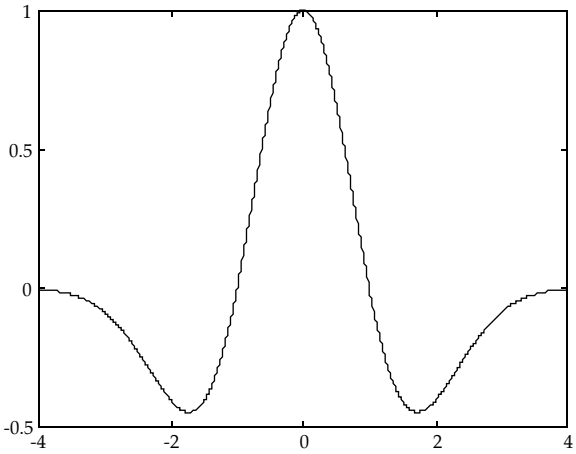


Figure 5. Graphe de l'ondelette "chapeau mexicain".

III.3 Réseaux d'ondelettes et réseaux de neurones.

La principale ressemblance entre les réseaux de neurones à fonctions dorsales, étudiés au chapitre II du présent mémoire, et les réseaux d'ondelettes, réside dans le fait que les deux réseaux calculent une combinaison linéaire, à paramètres ajustables, de fonctions non linéaires dont la forme dépend de paramètres ajustables (translations et dilatations).

Les différences essentielles entre ces deux types de réseaux sont les suivantes :

- contrairement aux fonctions dorsales, les ondelettes sont des fonctions qui décroissent rapidement , et tendent vers zéro dans toutes les directions de l'espace. Elles sont donc locales si d_j est petit ;
- contrairement aux fonctions dorsales, la forme de chaque ondelette monodimensionnelle est déterminée par deux paramètres ajustables (translation et dilatation) qui sont des paramètres structurels de l'ondelette ;
- chaque ondelette monodimensionnelle possède deux paramètres structurels, donc, pour chaque ondelette multidimensionnelle, le nombre de paramètres ajustables est le double du nombre de variables.

Pour comparer la complexité des réseaux, deux éléments sont importants : le nombre de paramètres ajustables et le nombre d'opérations élémentaires à effectuer (tableau 1).

Nous utiliserons les notations suivantes :

Nombre d'entrées : N_i (entrée constante non comprise).
Nombre de fonctions : N_w (pour les ondelettes), N_c (pour les fonctions dorsales).
Nombre de sorties : Une.

	Réseaux de fonctions dorsales	Réseaux de fonctions ondelettes
Nombre de fonctions.	N_c	N_w
Nombre de paramètres.	$\left(N_i + 2\right)\left(N_c + 1\right) \pm 1$	$2\,N_i\,N_w + N_w + N_i + 1$
Nombre d'opérations pour le calcul de la sortie.	$N_c\left(2\,N_i + 3\right) + N_i + 1$	$3\,N_w\left(N_i + 2\right) + N_i + 1$

Tableau 1. Une comparaison entre réseaux d'ondelettes et de fonctions dorsales.

On entend par "opération" les opérations mathématiques élémentaires, c'est-à-dire une addition, une multiplication ou une division. Étant données les propriétés de la fonction exponentielle, et pour chacun des deux types de réseaux, il y a autant de fonctions exponentielle à calculer que de neurones cachés ou d'ondelettes multidimensionnelles dans le réseau.
Lequel des deux types de réseaux est plus économique en termes de nombre d'opérations nécessaires pour le calcul de la sortie ? La réponse peut être obtenue en faisant la différence entre les deux résultats de la dernière ligne du tableau ci-dessus. En effet, à nombre de fonctions égales (c'est-à-dire $N_w = N_c$), la différence entre les nombre d'opérations pour les deux types de réseaux est égale à N_w (N_i + 3).
Le nombre d'opérations effectuées lors du calcul de la sortie avec un réseau d'ondelettes est donc toujours supérieur à celui effectué par un réseau de fonctions dorsales ayant le même nombre d'entrées et de fonctions.

IV. APPRENTISSAGE DES RÉSEAUX D'ONDELETTES NON BOUCLÉS.

IV.1 Calcul du gradient de la fonction de coût.

Les coefficients du réseau peuvent être divisés en deux classes :

– les paramètres structurels des fonctions, c'est-à-dire les translations et les dilatations ;

– les coefficients de pondérations c_j et les coefficients a_k de la partie affine.

Deux possibilités s'offrent à nous pour la construction du réseau :

– choisir les paramètres structurels dans un ensemble de valeurs discrètes ;

– considérer ces paramètres comme ceux d'un réseau de neurones classique et utiliser une technique d'optimisation pour en faire une estimation.

Discrétiser le domaine des translations et des dilatations signifie qu'on effectue la construction de réseaux d'ondelettes suivant une approche fondée sur la transformée en ondelettes discrète. Cette question sera étudiée en détail dans le chapitre suivant.

Dans ce qui suit, nous allons adopter la seconde possibilité, et faire appel aux techniques d'optimisation non linéaire décrite dans le chapitre I de ce document.

Rappelons que l'apprentissage consiste en la minimisation de la fonction de coût suivante :

$$J(\theta) = \frac{1}{2}\sum_{n=1}^{N}\left(y_p^n \pm y^n\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(e^n\right)^2 \tag{15}$$

avec

$$y^n = \psi\left(x^n, \theta\right) = \sum_{j=1}^{N_w} c_j \Phi_j\left(x^n, m_j, d_j\right) + \sum_{k=0}^{N_i} a_k x_k^n \quad \text{avec} \quad x_0^n = 1 \tag{16}$$

où y_p^n est la sortie désirée correspondant à l'exemple n, y^n est la sortie du réseau d'ondelettes pour l'exemple n, et

$$x^n = \left\{x_1^n, \ldots, x_{N_i}^n\right\} \tag{17}$$

est le vecteur des entrées.

θ est le vecteur regroupant l'ensemble des paramètres ajustables :

$$\theta = \left\{m_{jk}, d_{jk}, c_j, a_k, a_0\right\} \quad j = 1, \ldots, N_w \quad \text{et} \quad k = 1, \ldots, N_i \tag{18}$$

Les techniques d'optimisation utilisées nécessitent le calcul du vecteur gradient de la fonction de coût par rapport au vecteur des paramètres ajustables. Son expression est :

$$\frac{\partial J}{\partial \theta} = \pm \sum_{n=1}^{N} e^n \frac{\partial y^n}{\partial \theta} \tag{19}$$

où $\frac{\partial y^n}{\partial \theta}$ est la valeur du gradient de la sortie du réseau par rapport aux paramètres θ au point $x=x^n$:

$$\frac{\partial y^n}{\partial \theta} = \left.\frac{\partial y}{\partial \theta}\right|_{x = x^n} \tag{20}$$

Calculons à présent la dérivée de la sortie par rapport à chacun des paramètres du réseau.

• Pour les coefficients directs $\{a_k\}$:

$$\frac{\partial y^n}{\partial a_k} = x_k^n \quad k = 1\ ,\ \ldots\ ,\ N_i \tag{21}$$

• Pour les pondérations des ondelettes $\{c_j\}$:

$$\frac{\partial y^n}{\partial c_j} = \Phi_j(x^n) \quad k = 1\ ,\ \ldots\ ,\ N_i \ \text{ et } \ j = 1\ ,\ \ldots\ ,\ N_w \tag{22}$$

• Pour les translations $\{m_{jk}\}$:

$$\frac{\partial y^n}{\partial m_{jk}} = \pm \frac{c_j}{d_{jk}} \left.\frac{\partial \Phi_j}{\partial z_{jk}}\right|_{x = x^n} \quad k = 1\ ,\ \ldots\ ,\ N_i \ \text{ et } \ j = 1\ ,\ \ldots\ ,\ N_w \tag{23}$$

• Pour les dilatations $\{d_{jk}\}$:

$$\frac{\partial y^n}{\partial d_{jk}} = \pm \frac{c_j}{d_{jk}} z_{jk}^n \left.\frac{\partial \Phi_j}{\partial z_{jk}}\right|_{x = x^n} \quad k = 1\ ,\ \ldots\ ,\ N_i \ \text{ et } \ j = 1\ ,\ \ldots\ ,\ N_w \tag{24}$$

$\left.\frac{\partial \Phi_j}{\partial z_{jk}}\right|_{x = x^n}$ est la valeur de la dérivée partielle de l'ondelette multidimensionnelle par rapport à la variable z_{jk} au point $x=x^n$. Étant donné la relation (11), cette dérivée partielle vaut :

$$\left.\frac{\partial \Phi_j}{\partial z_{jk}}\right|_{x = x^n} = \phi\left(z_{j1}^n\right)\phi\left(z_{j2}^n\right) \ldots \phi'\left(z_{jk}^n\right) \ldots \phi\left(z_{jN_i}^n\right) \tag{25}$$

avec $\phi'\left(z_{jk}^n\right)$ la dérivée au point $x=x^n$ de l'ondelette scalaire, c'est-à-dire :

$$\phi'_{z_{jk}^n} = \left.\frac{d\phi(z)}{dz}\right|_{z = z_{jk}^n} \tag{26}$$

IV.2 Initialisation des paramètres du réseau.

Une fonction ondelette monodimensionnelle est définie sur tout l'ensemble $\boldsymbol{R}$, mais l'essentiel de sa contribution s'étend sur un intervalle centré autour de la valeur de la translation et dont la longueur dépend du paramètre de dilatation.

Dans le cas de réseaux de neurones à fonctions dorsales, l'initialisation des paramètres du réseau est généralement effectuée de manière aléatoire, de telle manière que le potentiel de chaque neurone caché soit suffisamment petit pour que les sorties des neurones se trouvent dans la partie linéaire de la sigmoïde. Les ondelettes étant des fonctions à décroissance rapide, une initialisation aléatoire des paramètres de translation et de dilatation serait très inefficace : en effet, si les translations sont initialisées à l'extérieur du domaine contenant les exemples, ou si les dilatations choisies sont trop petites, la sortie de l'ondelette

est pratiquement nulle, de même que sa dérivée. L'algorithme d'adaptation des paramètres étant fondé sur une technique de gradient, il est inopérant. Une attention particulière doit donc être portée à cette phase d'initialisation des paramètres.

Nous proposons ici une procédure d'initialisation simple, qui prend en considération le domaine où sont répartis les exemples de l'ensemble d'apprentissage.

Soit $[\alpha_k, \beta_k]$ l'intervalle contenant les $k^{\text{ème}}$ composantes des vecteurs d'entrée des exemples. On initialise les translations m_{jk} ($j = 1, \ldots, N_w$) au centre de l'intervalle $[\alpha_k, \beta_k]$:

$$m_{jk} = \frac{\alpha_k + \beta_k}{2} \quad \text{avec} \quad j = 1, \ldots, N_u \tag{27}$$

Les paramètres de dilatation sont choisis de telle manière que les variations de l'ondelette s'étendent sur tout l'intervalle $[\alpha_k, \beta_k]$. Cette condition est remplie avec le choix suivant :

$$d_{jk} = 0{,}2(\alpha_k \pm \beta_k) \quad \text{avec} \quad j = 1, \ldots, N_w \tag{28}$$

Cette procédure est valable notamment pour l'ondelette mère illustrée par la figure 4 que nous allons utiliser dans nos exemples.

Reste la question de l'initialisation des coefficients de pondération des ondelettes (c_j avec $j = 1, \ldots, N_w$) et ceux de la partie affine a_k avec $k = 1, \ldots, N_i$. L'initialisation de ces coefficients est moins importante, pour le déroulement de l'apprentissage, que celle des paramètres structurels ; ils sont initialisés de manière aléatoire, uniformément répartis dans l'intervalle$[-10^{-2} ; +10^{-2}]$.

Cette procédure ne nécessite pratiquement pas de calcul ; elle est très simple à mettre en œuvre. Le fait que, pour $j = 1, \ldots, N_w$ toutes les translations soient initialisées à la même valeur (ainsi que les dilatations) peut laisser penser qu'elles vont évoluer de manière identique si l'on effectue plusieurs apprentissages successifs. Une telle situation est évitée par le fait que les pondérations de chacune des ondelettes du réseau sont initialisées différemment.

Néanmoins, cette procédure d'initialisation présente un inconvénient : elle utilise peu les propriétés des ondelettes. En effet, on peut imaginer que l'on puisse mettre au point une technique d'initialisation qui utilise plus l'information apportée par les paramètres structurels, afin que la fonction de coût soit au voisinage d'un minimum avant d'effectuer l'apprentissage proprement dit. Une telle procédure est proposée dans le chapitre suivant ; elle est utilisable pour des réseaux d'ondelettes issus de la transformée en ondelettes discrète.

IV.3 Exemple de modélisation statique.

IV.3.1 Présentation du processus simulé.

Pour mettre en pratique les réseaux d'ondelettes non bouclés que nous venons de présenter, nous nous proposons d'étudier la modélisation statique d'un processus à une entrée.

Le processus est simulé à partir de la fonction définie sur l'intervalle [–10, +10] par :

$$f(x) = \begin{cases} \pm 2{,}186\, x \pm 12{,}864 \quad \text{si} \quad x \in [\pm 10, \pm 2[\\ 4{,}246\, x \quad \text{si} \quad x \in [\pm 2, 0[\\ 10 \exp\left(\pm 0{,}05\, x \pm 0{,}5\right) \sin\left(x\left(0{,}03\, x + 0{,}7\right)\right) \quad \text{si} \quad x \in [0, 10] \end{cases} \tag{29}$$

Le graphe de cette fonction est représenté sur la figure 6 :

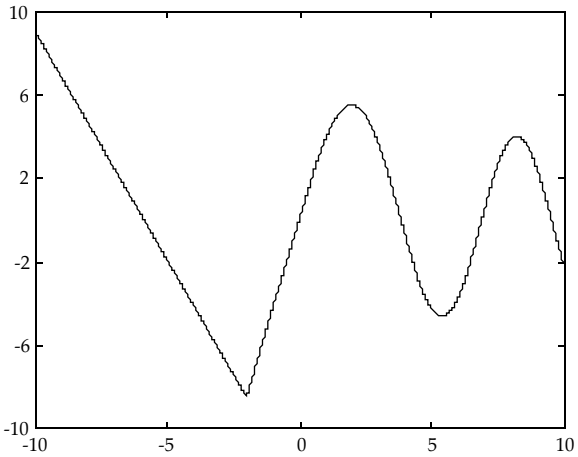


Figure 6. Sortie du processus pour l'intervalle de l'entrée considéré.

IV.3.2 Modélisation avec 100 exemples.

La séquence d'apprentissage est constituée de 100 exemples choisis de manière aléatoire, uniformément répartis, dans l'intervalle considéré. La séquence d'estimation de la performance du modèle est formée de 1000 exemples régulièrement répartis. On utilise les deux algorithmes de BFGS et de Levenberg–Marquardt (présentés dans le chapitre I de ce document). Dans le cas de l'utilisation de la procédure BFGS, une phase de gradient simple avec pas asservi est préalablement appliquée. Pour chaque réseau, on effectue cent apprentissages en modifiant à chaque fois le germe de l'initialisation aléatoire des pondérations $\{c_j\}$ des ondelettes et des coefficients $\{a_k\}$ de la partie affine du réseau. Rappelons que l'initialisation des translations et des dilatations est déterministe (suivant la procédure exposée au paragraphe précédent) : elle est donc identique pour tous les apprentissages. Nous avons testé quatre architectures, à 4, 6, 8 et 10 ondelettes.

Le tableau 2 présente, pour chacune de ces quatre architectures :

- le meilleur EQMP obtenu à l'issue de cent apprentissages avec l'algorithme de BFGS,
- l'EQMA correspondant.

Nombre d'ondelettes.	EQMA	EQMP
4	$7{,}9\ 10^{-3}$	$8{,}3\ 10^{-3}$
6	$1{,}3\ 10^{-3}$	$1{,}4\ 10^{-3}$
8	$5{,}7\ 10^{-4}$	$9{,}1\ 10^{-4}$
10	$1{,}0\ 10^{-4}$	$2{,}4\ 10^{-4}$

Tableau 2. Résultats obtenus avec l'algorithme de BFGS.

La Figure 7 présente les histogrammes des EQMA et des EQMP, pour les 100 apprentissages d'un réseau de 10 ondelettes effectués avec l'algorithme de BFGS. On observe une dispersion des EQMA et des EQMP, due à l'existence de minima locaux de la fonction de coût. Nous montrerons dans le paragraphe suivant que ce problème est très atténué si l'on utilise un plus grand nombre d'exemples pour l'apprentissage.

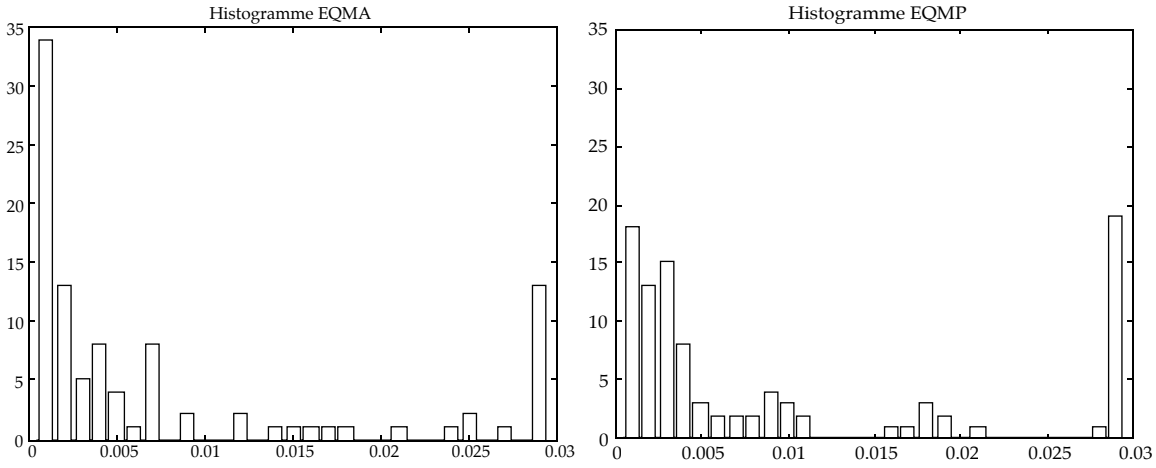


Figure 7. Histogrammes des EQMA et EQMP pour 100 apprentissages.

Les résultats obtenus dans les mêmes conditions en utilisant l'algorithme de Levenberg–Marquardt sont portés sur le tableau 3.

Nombre d'ondelettes.	EQMA	EQMP
4	$8{,}1\ 10^{-3}$	$7{,}8\ 10^{-3}$
6	$2{,}0\ 10^{-3}$	$2{,}3\ 10^{-3}$
8	$1{,}4\ 10^{-4}$	$3{,}2\ 10^{-4}$
10	$3{,}9\ 10^{-5}$	$1{,}9\ 10^{-4}$

Tableau 3. Résultats obtenus avec l'algorithme de Levenberg–Marquardt.

Les meilleurs résultats fournis par les deux algorithmes sont équivalents.

IV.3.3 Modélisation avec 300 exemples.

Nous utilisons cette fois un ensemble d'apprentissage comprenant 300 exemples uniformément répartis dans l'intervalle [–10, +10] et nous effectuons de nouveau 100 apprentissages comme précédemment.

Le tableau 4 présente, pour chacune des quatre architectures considérées :

- le meilleur EQMP obtenu à l'issue de cent apprentissages avec l'algorithme de BFGS,
- l'EQMA correspondant.

Nombre d'ondelettes.	EQMA	EQMP
4	$6{,}8\ 10^{-3}$	$6{,}6\ 10^{-3}$
6	$9{,}3\ 10^{-4}$	$1{,}2\ 10^{-3}$
8	$5{,}1\ 10^{-4}$	$6{,}4\ 10^{-4}$
10	$7{,}5\ 10^{-5}$	$1{,}1\ 10^{-4}$

Tableau 4. Résultats obtenus avec l'algorithme de BFGS.

La Figure 8 présente les histogrammes des EQMA et des EQMP, pour les 100 apprentissages d'un réseau de 10 ondelettes effectués avec l'algorithme de BFGS.

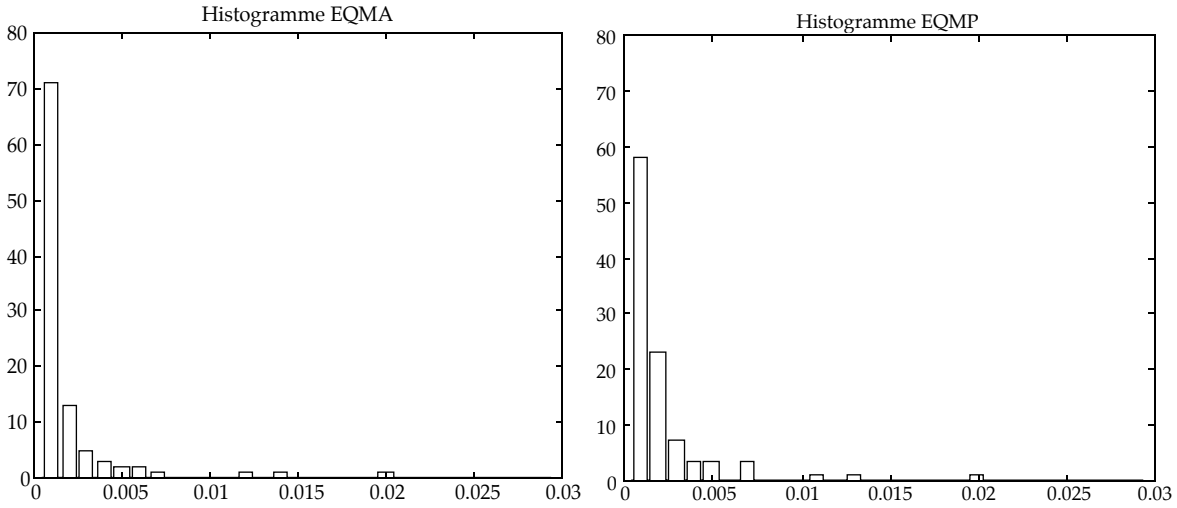


Figure 8. Histogrammes des EQMA et EQMP pour 100 apprentissages.

On constate que les résultats sont beaucoup moins dispersés que ceux qui sont présentés sur la Figure 7. Le fait que la distribution des minima locaux est d'autant plus large que le nombre d'exemples est petit n'est pas spécifique des ondelettes ; il a fait l'objet d'une étude dans [Stoppi97].

Les résultats obtenus dans les mêmes conditions en utilisant l'algorithme de Levenberg–Marquardt sont portés sur le tableau 5.

Nombre d'ondelettes.	EQMA	EQMP
4	$7{,}0\ 10^{-3}$	$6{,}5\ 10^{-3}$
6	$1{,}2\ 10^{-3}$	$1{,}3\ 10^{-3}$
8	$1{,}5\ 10^{-4}$	$2{,}6\ 10^{-4}$
10	$3{,}4\ 10^{-5}$	$5{,}0\ 10^{-5}$

Tableau 5. Résultats obtenus avec l'algorithme de Levenberg–Marquardt.

Là encore, les meilleurs résultats sont analogues à ceux qui ont été obtenus avec l'algorithme de BFGS. Les fréquences d'obtention des meilleurs minima sont voisines.
Cet exemple sera repris dans le chapitre IV, où nous illustrerons la mise en œuvre d'une procédure de sélection pour l'initialisation des translations et dilatations des ondelettes.

IV.3.4 Influence des termes directs

Les résultats présentés dans les deux paragraphes précédents étaient relatifs à des réseaux décrits par la relation (12), dans laquelle apparaissent des "termes directs" (coefficients $\{a_k, k\neq 0\}$) qui réalisent une fonction linéaire des entrées du réseau. Pour évaluer l'influence de ces termes, nous présentons ici les résultats obtenus par apprentissage de réseaux sans termes directs ($a_k = 0$, $k = 1, \ldots N_i$). Nous considérerons uniquement l'apprentissage avec 300 exemples.
Le tableau 6 présente les résultats obtenus après apprentissage par l'algorithme de BFGS, et le tableau 7 ceux obtenus par l'algorithme de Levenberg-Marquardt.

Nombre d'ondelettes.	EQMA	EQMP
4	$4{,}0\ 10^{-2}$	$3{,}6\ 10^{-2}$
6	$5{,}3\ 10^{-3}$	$5{,}4\ 10^{-3}$
8	$8{,}5\ 10^{-4}$	$1{,}2\ 10^{-3}$
10	$3{,}9\ 10^{-4}$	$4{,}7\ 10^{-4}$

Tableau 6. Résultats obtenus avec l'algorithme de BFGS.

Nombre d'ondelettes.	EQMA	EQMP
4	$2{,}6\ 10^{-2}$	$2{,}4\ 10^{-2}$
6	$1{,}4\ 10^{-3}$	$1{,}9\ 10^{-3}$
8	$3{,}7\ 10^{-4}$	$4{,}8\ 10^{-4}$
10	$3{,}3\ 10^{-4}$	$4{,}0\ 10^{-4}$

Tableau 7. Résultats obtenus avec l'algorithme de Levenberg–Marquardt.

On observe que les EQM sont systématiquement supérieures à celles que l'on obtient avec des réseaux comportant des termes directs.

IV.3.5 Quelques figures.

La figure suivante illustre la disposition des ondelettes en fin d'apprentissage pour le réseau de 10 ondelettes optimisé avec l'algorithme de Levenberg–Marquardt.

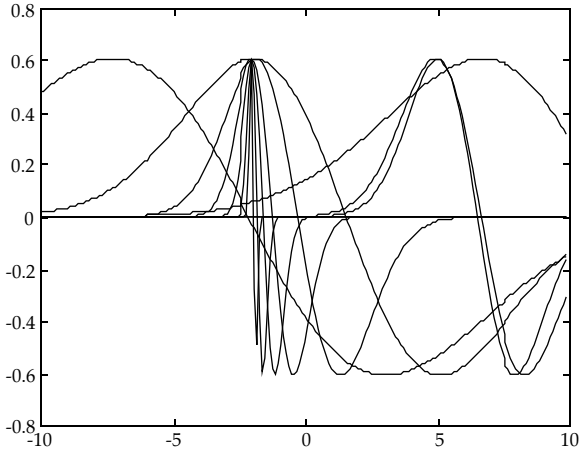
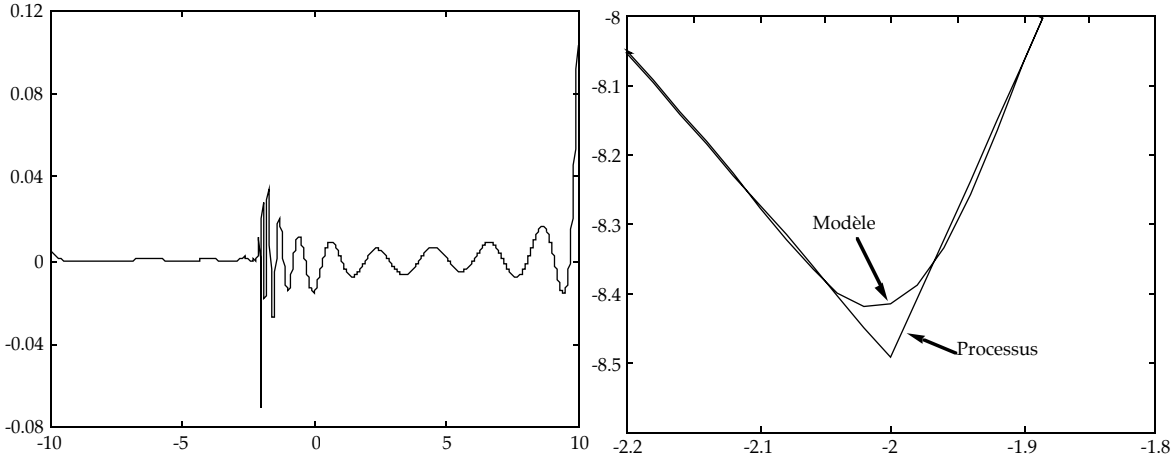


Figure 9. Ondelettes en fin d'apprentissage.

On peut observer que tous les centres ne sont pas à l'intérieur du domaine où est définie la fonction *f*, mais l'intersection du support de chacune des ondelettes avec le domaine est non nulle. Afin d'utiliser une seule échelle, les ondelettes sont représentées avec leurs sorties non pondérées par les coefficients c_j.



(a) (b)

Figure 10. Erreur de modélisation (a) et détail de la sortie du modèle et du processus autour du point anguleux (b).

La figure 10 illustre l'erreur de modélisation (a) commise par le réseau de 10 ondelettes. L'erreur est principalement commise au niveau du point anguleux (b) qui est certainement la seule difficulté pour l'approximation de cette fonction.

V. MODÉLISATION DYNAMIQUE ENTRÉE–SORTIE ET RÉSEAUX D'ONDELETTES.

Comme nous venons de le voir, la construction de réseaux d'ondelettes non bouclés pour la modélisation statique de processus tire son origine de la transformée en ondelettes inverse. On se propose d'étendre l'utilisation des réseaux d'ondelettes à la modélisation dynamique de processus.

Considérons un modèle-hypothèse de la forme :

$$y_p(n) = f\left(y_p(n\text{-}1)\ ,\ \dots\ ,\ y_p(n\text{-}N_s),\ u(n\text{-}1)\ ,\ \dots\ ,\ u(n\text{-}N_e),\ w(n)\ ,\ \dots\ ,\ w(n\text{-}N_n)\right) \qquad (30)$$

où u est une entrée externe appliquée au processus et y_p sa sortie. N_s est l'ordre du modèle. $\{w(n)\}$ est une séquence de variables aléatoires de moyenne nulle et de variance σ^2. f est une fonction paramétrée inconnue dont il s'agit d'estimer les paramètres à l'aide d'une séquence d'apprentissage. Chaque exemple correspond à un instant de mesure. Pour une séquence d'apprentissage de N exemples, nous avons $n = 1, \dots, N$.

Des hypothèses supplémentaires sont généralement faites sur la façon dont le bruit agit. Un choix adéquat du prédicteur associé peut alors être effectué. Différents exemples de modèles-hypothèses ainsi que les prédicteurs optimaux qui leur sont associés sont présentés dans le paragraphe III.3 du chapitre I.

Rappelons que :

- si l'hypothèse de l'existence d'un bruit additif de sortie a été retenue, ou si, en l'absence de bruit, on désire obtenir un modèle de simulation du processus, les entrées d'état du prédicteur, durant son apprentissage, sont les sorties passées du prédicteur ; si le prédicteur est réalisé par un réseau, celui-ci est bouclé pendant l'apprentissage ;
- si l'hypothèse de l'existence d'un bruit d'état additif a été retenue, ou si, en l'absence de bruit, on envisage d'utiliser le prédicteur pour prédire la sortie une seule période d'échantillonnage plus tard, les entrées d'état du prédicteur, durant son apprentissage, sont les sorties du processus ; si le prédicteur est une réseau de fonctions, celui-ci est non bouclé pendant l'apprentissage.

A notre connaissance, les réseaux d'ondelettes bouclés n'ont jamais été étudiés auparavant. On trouve dans la conclusion de la référence [Zhang92] (qui traite des réseaux d'ondelettes fondés sur la transformée en ondelette continue)

un commentaire à ce propos. Les auteurs soulignent qu'une investigation des performances de tels réseaux est une voie à explorer.

V.1 Apprentissage de réseaux de type entrée-sortie.

Le schéma d'apprentissage que nous adoptons est semblable à celui qui est utilisé dans le cas de réseaux de neurones à fonctions sigmoïdes. Nous utilisons la notion de copie du réseau. On désigne par "copie numéro n" la partie statique du réseau canonique qui calcule $y(n)$.

V.1.1 Apprentissage de prédicteurs non bouclés.

Pour paramétrer un réseau constituant un prédicteur non bouclé pendant l'apprentissage, on se ramène à la notation utilisée pour les réseaux statiques.

Soit $x^n \in R^{N_i}$ le vecteur d'entrée de la copie n. Ses différentes composantes sont les suivantes :

Pour $k = 1$, ... , N_e : $x_k^n = u(n \pm k)$ sont les entrées externes. N_e est le nombre de ces entrées.

Pour $k = N_e$+1, ... , N_e+ N_s : $x_k^n = y_p(n \pm k + N_e)$ sont les entrées d'état, qui sont les sorties mesurées sur le processus (entrées d'état).

Comme nous l'avons rappelé dans le paragraphe précédent, les valeurs des entrées d'état pour chaque copie sont forcées aux sorties correspondantes du processus. Le prédicteur est dirigé par le processus, d'où le nom d'apprentissage "dirigé" [Nerrand92, Nerrand93] ou *"teacher–forced"* [Jordan85].

Nous avons ainsi $N_e + N_s = N_i$, le nombre d'entrées d'un réseau d'ondelettes pour la modélisation statique.

L'apprentissage par la méthode du gradient s'effectue de la même manière que dans le cas de la modélisation statique.

V.1.2 Apprentissage de prédicteurs bouclés.

Pour un prédicteur bouclé pour l'apprentissage, le calcul du gradient de la fonction de coût doit tenir compte du fait que le réseau est bouclé.

Pour $k = 1, \ldots, N_e : x_k^n = u(n \pm k)$ sont les entrées externes.

Pour $k = N_e$+1, ... , N_e+ N_s : $x_k^n = y(n \pm k + N_e)$ sont les N_s valeurs passées des sorties de la copie n–1.

Pour $k = N_e$+N_s+1 , ... , N_e+2N_s : $x_k^n = y(n \pm k + N_e + N_s + 1)$ sont les sorties de la copie n.

La figure 11 illustre la configuration du réseau pour l'exemple de l'instant n (c'est-à-dire la copie numéro n).

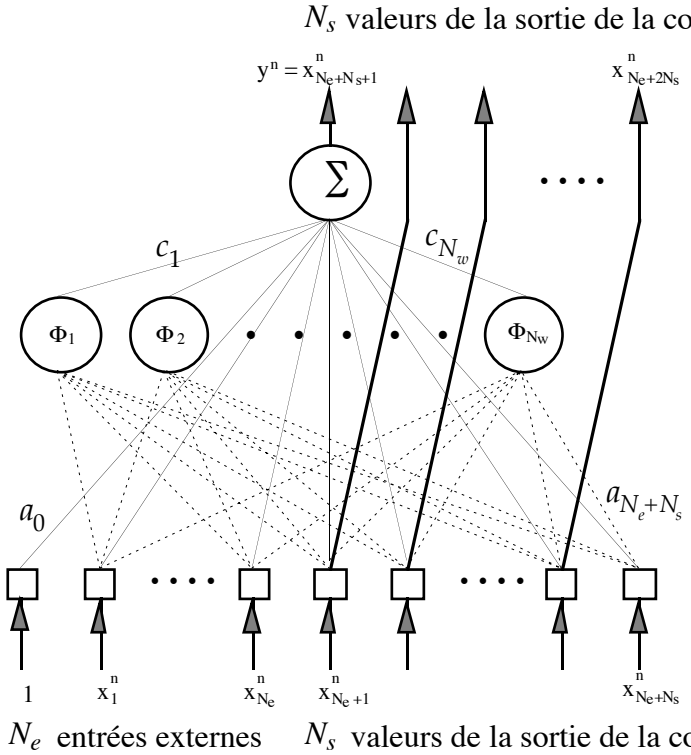


Figure 11. La copie numéro n du réseau prédicteur entrée-sortie bouclé.

Ici, seules les valeurs des entrées d'état de la première copie sont prises égales aux valeurs correspondantes de la sortie du processus. Pour les copies suivantes, ces entrées prennent les valeurs des variables d'état en sortie de la copie précédente. Le prédicteur est "*semi-dirigé*" par le processus. Pour cette raison, l'algorithme est dit *"semi-dirigé"* [Nerrand92] (*"backpropagation through time"* [Rumelhart86]).

Rappelons que la fonction de coût à minimiser au cours de l'apprentissage est la même que dans le cas de la modélisation statique, c'est-à-dire $J(\theta) = \frac{1}{2}\sum_{n=1}^{N}\left(y_p^n \pm y^n(\theta)\right)^2$, où θ est le vecteur des paramètres ajustables.

On désigne par θ^n le vecteur des paramètres de la copie n du réseau :

$$\theta^n = \left\{ m_{jk}^n,\, d_{jk}^n,\, c_j^n,\, a_k^n,\, a_0^n \right\} \quad avec \quad j = 1, \ldots, N_w \text{ et } k = 1, \ldots, N_e+N_s \tag{31}$$

Il est nécessaire de distinguer les paramètres θ_i^n et $\theta_i^{n'}$ de deux copies différentes n et n' bien qu'ils aient les mêmes valeurs : en effet, les composantes du gradient $\frac{\partial J}{\partial \theta_i^n}$ et $\frac{\partial J}{\partial \theta_i^{n'}}$ sont différentes. Rappelons que $\frac{\partial J}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial J}{\partial \theta^n}$.

Ces notations étant définies, nous abordons le calcul du gradient $\frac{\partial J}{\partial \theta}$ pour les réseaux d'ondelettes bouclés. Deux approches sont possibles :

- calcul par rétropropagation,
- calcul dans le sens direct.

V.1.3 Calcul du gradient par rétropropagation.

Le vecteur gradient est décomposé de la manière suivante :

$$\frac{\partial J}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial J}{\partial \theta^n} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial \theta^n} \tag{32}$$

La quantité $\frac{\partial y^n}{\partial \theta^n}$ est la dérivée de la sortie de la copie n par rapport aux coefficients de la même copie. Les expressions de cette dérivée pour chacune des composantes du vecteur θ sont données par les relations (20) à (24).

Reste donc à calculer $\frac{\partial J}{\partial y^n}$ pour $n = 1, \ldots, N$.

Afin de présenter les expressions de façon plus claire, on introduit des variables intermédiaires que l'on note par q_k^n et que l'on définit par :

$$q_k^n = \pm \frac{\partial J}{\partial x_k^n} \quad \text{avec} \quad k = N_e+1 , \ \ldots \ , N_e+2N_s \tag{33}$$

Ce sont les dérivées partielles de la fonction $-J$ par rapport aux variables d'état en entrée et en sortie de la copie numéro n.

Pour la présentation du calcul du gradient par rétropropagation, on considère séparément la dernière copie, de numéro N, celles dont le numéro est compris entre $N-1$ et 2, et enfin la première copie.

<u>Pour la copie N</u>, nous avons :

Pour la sortie :

$$\pm \frac{\partial J}{\partial y^N} = q_{sortie}^N = q_{N_e+N_s+1}^N = e^N \tag{34}$$

Pour les autres variables d'état (en sortie du réseau) :

$$q_k^N = 0 \quad \text{avec} \quad k = N_e+N_s+2 , \ \ldots \ , N_e+2N_s \tag{35}$$

Pour les variables d'état (en entrée du réseau) :

$$\pm \frac{\partial J}{\partial x_k^N} = q_k^N = \left(a_k^N + \sum_{j=1}^{N_w} \frac{c_j^N}{d_{jk}^N} \frac{\partial \Phi_j^N}{\partial z_{jk}^N} \right) q_{sortie}^N \quad \text{avec} \quad k = N_e+1 , \ \ldots \ , N_e+N_s \tag{36}$$

<u>Pour les copies de $n=N-1$ à 2</u>, nous avons :

Pour la sortie :

$$\pm \frac{\partial J}{\partial y^n} = q^n_{sortie} = e^n + q^{n+1}_{N_e+1} \tag{37}$$

Pour les autres variables d'état (en sortie du réseau) :

$$q^n_k = q^{n+1}_{k \pm N_s} \quad \text{avec} \quad k = N_e+N_s+2 \, , \, \ldots \, , \, N_e+2N_s \tag{38}$$

Pour les variables d'état en entrée du réseau :

$$\pm \frac{\partial J}{\partial x^n_k} = q^n_k = q^n_{k+N_s+1} + \left(a^n_k + \sum_{j=1}^{N_w} \frac{c^n_j}{d^n_{jk}} \frac{\partial \Phi^n_j}{\partial z^n_{jk}} \right) q^n_{sortie} \quad \text{avec} \quad k = N_e+1 \, , \, \ldots \, , \, N_e+N_s \tag{39}$$

Pour la copie n=1, nous avons :

Pour la sortie :

$$\pm \frac{\partial J}{\partial y^1} = q^1_{sortie} = e^1 + q^2_{N_e+1} \tag{40}$$

V.1.4 Calcul du gradient dans le sens direct.

L'utilisation de l'algorithme de Levenberg–Marquardt pour l'apprentissage de réseaux bouclés nécessite la mise en oeuvre du calcul du gradient dans le sens direct. En effet, cet algorithme demande le calcul du Hessien de la fonction de coût J (ou d'une approximation de celui-ci). Rappelons que cette approximation s'exprime de la façon suivante :

$$\tilde{H} = \sum_{n=1}^{N} \frac{\partial e^n}{\partial \theta} \left(\frac{\partial e^n}{\partial \theta} \right)^T = \sum_{n=1}^{N} \frac{\partial y^n}{\partial \theta} \left(\frac{\partial y^n}{\partial \theta} \right)^T \tag{41}$$

Les dérivées partielles sont calculées à partir de :

$$\frac{\partial y^n}{\partial \theta} = \sum_{m=1}^{n} \frac{\partial y^n}{\partial \theta^m} \tag{42}$$

Afin d'obtenir les quantités de la relation (42) avec un calcul du gradient par rétropropagation, il est nécessaire d'effectuer N rétropropagations. De ce fait, le calcul dans le sens direct est plus économique.

La relation précédente exprime que le calcul du gradient de la sortie de la copie numéro n du réseau par rapport au vecteur des paramètres θ doit prendre en considération les dérivées de cette sortie par rapport à chacune des copies du vecteur des paramètres d'indices inférieurs ou égaux. Décomposons l'expression (42) :

$$\sum_{m=1}^{n} \frac{\partial y^n}{\partial \theta^m} = \sum_{m=1}^{n\pm 1} \frac{\partial y^n}{\partial \theta^m} + \frac{\partial y^n}{\partial \theta^n} \tag{43}$$

Le deuxième terme est donné par les relations (21) à (24) ; il suffit donc de calculer le premier.

La quantité $\frac{\partial y^n}{\partial \theta^m}$ peut s'écrire de la manière suivante, où y^{n-l} est la variable d'état numéro l en entrée de la $n^{ème}$ copie :

$$\frac{\partial y^n}{\partial \theta^m} = \sum_{l=1}^{N_s} \frac{\partial y^n}{\partial y^{n\pm l}} \frac{\partial y^{n\pm l}}{\partial \theta^m} \quad \text{avec } m = 1, \ldots, n\pm 1 \text{ et } n \pm l \geq m \tag{44}$$

On obtient ainsi :

$$\sum_{m=1}^{n\pm 1} \frac{\partial y^n}{\partial \theta^m} = \sum_{l=1}^{N_s} \frac{\partial y^n}{\partial y^{n\pm l}} \left(\sum_{m=1}^{n\pm 1} \frac{\partial y^{n\pm l}}{\partial \theta^m} \right) \quad \text{avec} \quad n \pm l \geq m \tag{45}$$

Remarquons que le second facteur peut s'écrire :

$$\sum_{m=1}^{n\pm 1} \frac{\partial y^{n\pm l}}{\partial \theta^m} = \sum_{m=1}^{n\pm l} \frac{\partial y^{n\pm l}}{\partial \theta^m} = \frac{\partial y^{n\pm l}}{\partial \theta} \tag{46}$$

En introduisant ce dernier résultat dans la relation (45), on obtient :

$$\sum_{m=1}^{n\pm 1} \frac{\partial y^n}{\partial \theta^m} = \sum_{l=1}^{N_s} \frac{\partial y^n}{\partial y^{n\pm l}} \frac{\partial y^{n\pm l}}{\partial \theta} \tag{47}$$

En reprenant la relation (43), on aboutit à :

$$\frac{\partial y^n}{\partial \theta} = \sum_{l=1}^{N_s} \frac{\partial y^n}{\partial y^{n\pm l}} \frac{\partial y^{n\pm l}}{\partial \theta} + \frac{\partial y^n}{\partial \theta^n} \tag{48}$$

La quantité $\frac{\partial y^n}{\partial y^{n\pm l}}$ est la dérivée de la sortie de la copie n par rapport à la sortie calculée de la copie $n-l$, qui est donc la $l^{\text{ème}}$ variable d'état en entrée de la copie n. Elle est donnée par :

$$\frac{\partial y^n}{\partial y^{n\pm l}} = a^n_{N_e+l} + \sum_{j=1}^{N_w} \frac{c^n_j}{d^n_{j,N_e+l}} \frac{\partial \Phi^n_j}{\partial z^n_{j,N_e+l}} \quad \text{avec} \quad l=1, \ldots, N_s \tag{49}$$

La relation (48) permet de calculer la dérivée de la sortie y^n de la copie n par rapport à θ en fonction de celles calculées aux N_s copies précédentes.

V.2 Exemple.

V.2.1 Présentation du processus.

On se propose d'étudier un exemple de modélisation dynamique à l'aide d'un réseau d'ondelettes bouclé afin de mettre en œuvre les algorithmes et procédures présentés au paragraphe précédent.
Le processus est simulé à partir d'une équation aux différences ayant pour expression :

$$y_p(k) = \frac{y_p(k\pm1)\, y_p(k\pm2)\, y_p(k\pm3)\, u(k\pm2)\left(y_p(k\pm3) \pm 1\right) + u(k\pm1)}{1 + y_p^2(k \pm 2) + y_p^2(k \pm 3)} \tag{50}$$

où $u(.)$ est l'entrée externe et $y_p(.)$ la sortie du processus. Afin de simuler le processus, il est indispensable de choisir une séquence pour l'entrée externe u. La séquence des entrées externes est une séquence pseudo-aléatoire de distribution uniforme entre –1 et 1. Les séquences d'apprentissage et d'estimation de la performance sont constituées chacune de 1000 points. Étant donné que le processus n'est pas bruité, on peut effectuer indifféremment une modélisation avec un réseau bouclé ou non. On choisit la première possibilité.

Dans le domaine des entrées choisi, c'est-à-dire [–1, +1], les sorties sont comprises dans le même intervalle.

V.2.2 Étude du gain statique.

Un régime statique est atteint si pour $u(k-1)=u(k-2)=\alpha$ constante, on a $y_p(k)=y_p(k-1)=y_p(k-2)=y_p(k-2)=y_p(k-3)=\beta$ constante. Le gain statique est le rapport de la sortie à l'entrée :

$$G_{\text{statique}} = \frac{\beta}{\alpha} \tag{51}$$

En utilisant les égalités précédentes et le modèle du processus donné par la relation (50), on obtient pour expression du gain statique en fonction de la sortie :

$$G_{\text{statique}}(\beta) = \frac{\beta^3\left(\beta \pm 1\right) + 1}{1 + 2\,\beta^2} \tag{52}$$

Dans le domaine des entrées que nous avons choisi pour la construction de nos deux séquences, le graphe du gain statique est le suivant :

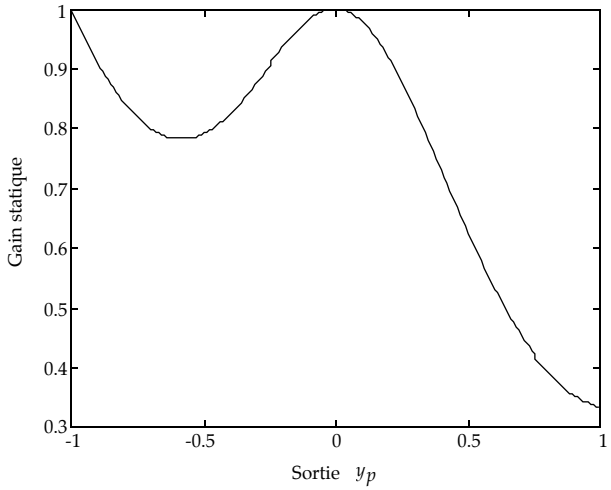


Figure 12. Gain statique dans le domaine de sortie [-1 ; +1]

On constate que pour de faibles amplitudes de la sortie (proches de zéro) le gain statique est proche de l'unité. Précisons que cette étude ne nous donne pas d'information sur la stabilité du modèle. En fait, des essais de simulation montrent que le modèle est instable si des entrées d'amplitudes supérieures à 1 sont appliquées.

V.2.3 Modélisation du processus.

On se propose d'utiliser quatre architectures de réseaux formés respectivement de 5, 10 et 15 ondelettes. Une modélisation linéaire est également effectuée (réseau ne contenant aucune ondelette). Pour chaque architecture, on effectue 50 apprentissages en modifiant à chaque fois le germe de l'initialisation aléatoire des paramètres. Le réseau retenu est celui dont performance estimée est la meilleure. Les résultats obtenus en utilisant l'algorithme de BFGS sont représentés dans le tableau suivant :

Nb. d'ondelettes.	EQMA	EQMP
0	$8{,}1\ 10^{-3}$	$9{,}3\ 10^{-3}$
5	$8{,}7\ 10^{-6}$	$1{,}3\ 10^{-5}$
10	$1{,}5\ 10^{-6}$	$2{,}3\ 10^{-6}$
15	$1{,}3\ 10^{-7}$	$2{,}9\ 10^{-7}$

Tableau 8. Résultats obtenus avec l'algorithme de BFGS.

On effectue également des apprentissages dans les mêmes conditions en utilisant cette fois l'algorithme de Levenberg–Marquardt avec le calcul du gradient dans le sens direct (comme présenté plus haut dans ce chapitre). Les résultats obtenus sont reportés sur le tableau 9.

Nb. d'ondelettes.	EQMA	EQMP
0	$8{,}1\ 10^{-3}$	$9{,}3\ 10^{-3}$
5	$6{,}6\ 10^{-6}$	$9{,}4\ 10^{-6}$
10	$1{,}1\ 10^{-7}$	$3{,}1\ 10^{-7}$
15	$1{,}2\ 10^{-8}$	$4{,}8\ 10^{-8}$

Tableau 9. Résultats obtenus avec l'algorithme de Levenberg–Marquardt.

Les EQM obtenues ici avec les grands réseaux (10 et 15 ondelettes) sont plus faibles que celles obtenues par l'algorithme de BFGS. Sur cet exemple, nous avons donc un meilleur comportement de l'algorithme de Levenberg–Marquardt au prix d'un temps de calcul beaucoup plus important.

VI. MODÉLISATION D'ÉTAT ET RÉSEAUX D'ONDELETTES.

L'état d'un modèle est l'ensemble minimal des N_s valeurs nécessaires à l'instant k pour calculer sa sortie à l'instant k+1, les valeurs des entrées étant données jusqu'à k. N_s est l'ordre du modèle. La représentation d'état est la représentation la plus générale du comportement dynamique d'un processus. La représentation entrée-sortie en est un cas particulier.

Un modèle d'état à temps discret est constitué

- d'un système de N_s équations récurrentes du 1er ordre exprimant l'état à l'instant k+1 en fonction de l'état et des entrées à l'instant k,
- d'une équation d'observation, qui exprime la sortie en fonction de l'état, ou plus généralement, de l'état et des entrées.

Un modèle-hypothèse prend la forme suivante :

$$\begin{cases} x_p(k+1) = f\left(x_p(k), u(k)\right) \\ y_p(k) = g\left(x_p(k)\right) \end{cases} \tag{53}$$

où $x_p(k) \in R^{N_s}$ est le vecteur d'état à l'instant k, $u(k) \in R^{N_e}$ est le vecteur des entrées et $y_p(k)$ la sortie du processus à l'instant k. f et g sont des fonctions à variable vectorielle.

Dans le cadre de cette étude, on s'intéressera aux cas où $u(k)$ est une entrée scalaire et où le modèle ne possède qu'une sortie (modèles mono-entrée mono-sortie ou SISO pour Single Input, Single Output).

Dans le cas d'un modèle entrée-sortie, les variables d'état sont les sorties, donc elles sont nécessairement mesurées, ce qui n'est pas le cas pour un modèle d'état.

Dans la suite de la présente étude de modélisation par réseaux d'état, nous ne considérons que des modèles sans bruit ou des modèles avec un bruit additif de sortie.

VI.1 Modèles d'état sans bruit, avec états non mesurables.

Si les états d'un modèle boîte noire ne sont pas mesurables, seul le comportement entrée–sortie peut être modélisé. Dans ce cas, les états obtenus n'ont pas forcément une signification physique, contrairement au cas où ils sont mesurables. En changeant le nombre de fonctions dans le réseau, ou son initialisation, les modèles obtenus peuvent posséder une performance équivalente du point de vue du comportement entrée–sortie, bien que les séquences des variables d'état soient différentes. Le prédicteur associé est obligatoirement bouclé et son expression est la suivante :

$$\begin{cases} x(k+1) = \psi_1(x(k), u(k)) \\ y(k+1) = \psi_2(x(k), u(k)) \end{cases} \tag{54}$$

Ce prédicteur est illustré par la figure suivante :

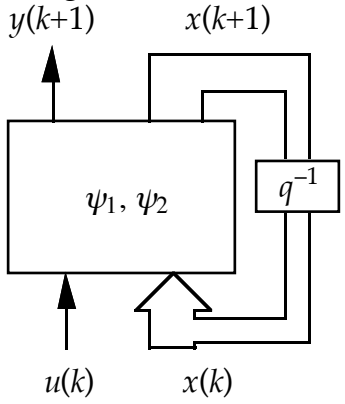


Figure 13. Prédicteur d'état bouclé.

Dans le cas où la sortie *y(k)* n'est pas fonction de *u(k)*, un réseau associé comme celui de la figure 13 est envisageable en supprimant la connexion correspondante.

Dans les exemples que nous étudierons dans la suite de ce mémoire, les états sont généralement non mesurables et les séquences d'apprentissage et d'estimation de la performance dont on dispose sont formées uniquement des mesures de l'entrée externe et de la sortie du processus : seul l'apprentissage d'un prédicteur d'état bouclé est possible. Dans la suite de ce mémoire, on ne considérera donc que des prédicteurs d'état bouclés comme celui illustré par la figure 13.

VI.2 Apprentissage de réseaux d'état bouclés.

VI.2.1 Structure du réseau d'état.

Le réseau non bouclé de la forme canonique d'un réseau d'ondelettes bouclé comprend :

• Une couche d'entrée possédant N_e entrées externes et N_s variables d'état. Le nombre total des entrées est alors $N_i = N_e + N_s$.

• Une couche cachée constituée par N_w ondelettes multidimensionnelles.

• Une couche de sortie comportant un neurone linéaire donnant la sortie du réseau $y(n) = y^n$ et N_s neurones linéaires d'état donnant chacun la valeur de l'état correspondant pour l'instant considéré.

La notion de copie de réseau utilisée dans le cadre de la modélisation entrée–sortie est généralisée. Ici, comme expliqué plus haut, seuls les réseaux d'état bouclés seront utilisés et l'apprentissage met en jeu un grand réseau constitué par N copies en cascade (N est toujours la taille de la séquence d'apprentissage). Le vecteur d'état en entrée de la copie numéro n est le vecteur de sortie de la copie précédente. Pour la première copie :

- si les états sont mesurables le vecteur est identique au vecteur des entrées du processus ;
- si les états ne sont pas mesurables, et en l'absence de toute information sur l'état initial du processus, on force les entrées d'état de la première copie à zéro.

Dans les exemples que nous avons traités, nous nous trouvons dans cette dernière situation.

Suivant la description donnée plus haut sur la structure du réseau, le vecteur θ des paramètres est composé des éléments suivants :

$$\theta = \{m_{jk}, d_{jk}, c_{kj}, c_0\} \tag{55}$$

• Les translations m_{jk} et les dilatations d_{jk} avec k=1 , ... , $N_e + N_s$ et j=1 , ... , N_w.

• Les pondérations et les coefficients directs que l'on note par c_{kj}. Ce choix d'indices signifie qu'il s'agit du coefficient de la liaison entre la fonction (ou le neurone d'entrée) numéro j et le neurone de sortie (ou le neurone d'état) numéro k. Pour les pondérations nous avons :
$j= N_e+N_s+1$, ... , $N_e+N_s+N_w$ et $k=N_e+N_s+N_w+1$, ... , $N_e+2N_s+N_w+1$.
Pour les coefficients directs, nous avons :
j=1, ... , N_e+N_s et $k=N_e+N_s+N_w+1$, ... , $N_e+2N_s+N_w+1$.

• Un terme constant sur le neurone linéaire de la sortie que l'on note par c_0.

• Le nombre de composantes du vecteur θ est alors :

$$2N_w(N_e+N_s)+(N_s+1)(N_e+N_s+N_w)+1.$$

On note par x_k^n la variable d'état numéro k en entrée de la copie numéro n du réseau si $k=N_e+1$, ... , N_e+N_s et en sortie de cette copie si $k=N_e+N_s+N_w+2$, ... ,

N_e+2N_s+N_w+1. La figure 14 montre l'architecture de la copie n. Notons que les noeuds sont numérotés dans l'ordre de 1 à N_e+2N_s+N_w+1 alors que les ondelettes le sont de 1 à N_w.

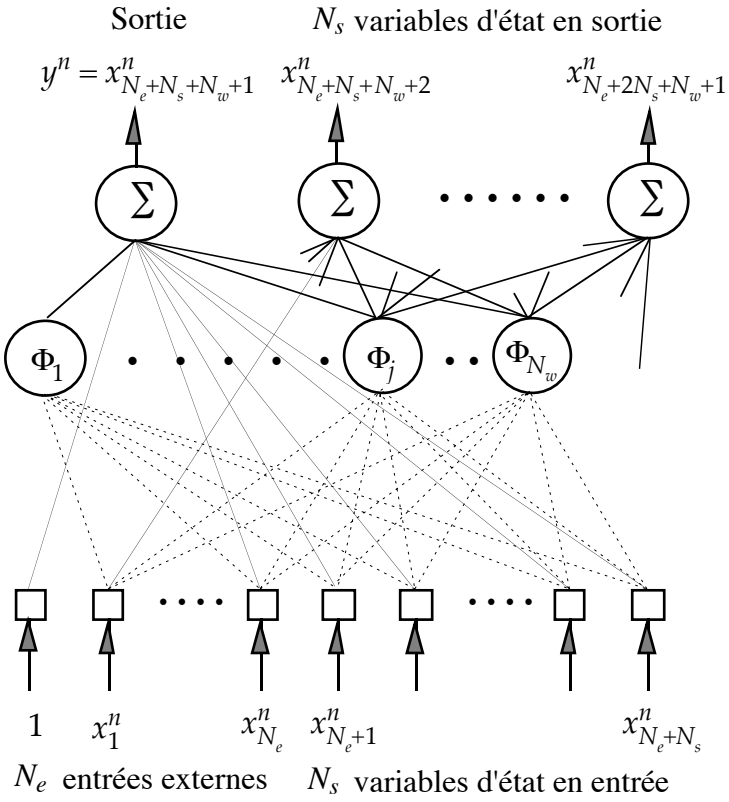


Figure 14. Illustration de la copie numéro n du réseau d'état.

La sortie de la copie n du réseau a pour expression :

$$y^n = \sum_{j=1}^{N_w} c_{\alpha,\, j+N_e+N_s}\, \Phi_j\left(x^n\right) + \sum_{k=0}^{N_e+N_s} c_{\alpha,k}\, x^n_k \quad \text{avec} \quad \alpha = N_e+N_s+N_w+1 \quad \text{et} \quad x^n_0=1 \qquad (56)$$

avec $x^n = \left\{x^n_1, x^n_2, \ldots, x^?_{N_e+N_s}\right\}$.

La variable d'état numéro k en sortie du réseau est calculée à partir de la relation suivante :

$$x^n_k = \sum_{j=N_e+N_s+1}^{N_e+N_s+N_w} c_{k,\,j}\, \Phi_j\left(x^n\right) + \sum_{j=1}^{N_e+N_s} c_{k,\,j}\, x^n_j \quad \text{avec} \quad k = N_e+N_s+N_w+2\,,\ \ldots\,,\ N_e+2N_s+N_w+1 \qquad (57)$$

Notons que nous avons omis les termes directs sur les neurones linéaires d'état. En effet, ces termes ont ici moins d'importance que pour le neurone linéaire de la sortie du réseau.

La variable d'état en entrée est prise égale à celle de la variable d'état correspondante en sortie de la copie précédente. On peut écrire pour la copie n :

$$x^n_k = \begin{cases} x^{n\pm1}_{k\,+N_s+N_w+1} & \text{si } n \geq 2 \\ 0 & \text{si } n = 1 \end{cases} \qquad \text{avec} \quad k = N_e+1\,,\ \ldots\,,\ N_e+N_s \qquad (58)$$

VI.2.2 Calcul du gradient par rétropropagation.

De la même façon que pour les réseaux de type entrée–sortie bouclés et non bouclés, nous devons calculer le gradient de la fonction de coût par rapport au vecteur des paramètres. L'apprentissage est également fondé sur des méthodes de gradient telles que celles présentées au chapitre I de ce mémoire.

Les deux approches déjà mentionnées (calcul du gradient par rétropropagation à travers les copies ou dans le sens direct) sont envisageables. Nous les présenterons toutes les deux. Seule la première sera mise en œuvre. En effet, dans le cas des réseaux d'état, le calcul du gradient dans le sens direct implique un volume de calculs plus important que le calcul par rétropropagation. Ce dernier se divise en deux étapes :

- calcul du gradient de la fonction de coût J par rapport à la sortie et aux variables d'état en entrée et en sortie de chacune des N copies,
- calcul du gradient de J par rapport au vecteur θ des paramètres.

VI.2.2.1 Calcul du gradient de J par rapport à la sortie et aux variables d'état.

La fonction de coût utilisée pour l'apprentissage est toujours : $J(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left(y_p^n \pm y^n(\theta) \right)^2$.

On distingue trois calculs différents suivant qu'il s'agit de la première copie, de la dernière ou des autres.

<u>Pour la copie N,</u> nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^N} = \pm e^N \tag{59}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... , $N_e+2N_s+N_w+1$:

$$\frac{\partial J}{\partial x_k^N} = 0 \tag{60}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... , N_e+N_s :

$$\frac{\partial J}{\partial x_k^N} = \frac{\partial J}{\partial y^N} \frac{\partial y^N}{\partial x_k^N} = \pm e^N \left(c_{\alpha,k} + \sum_{j=1}^{N_w} \frac{c_{\alpha, N_e+N_s+j}}{d_{jk}} \frac{\partial \Phi_j(x^N)}{\partial z_{jk}} \right) \tag{61}$$

avec $\alpha = N_e+N_s+N_w+1$.

<u>Pour les copies de $n=N-1$ à 2</u>, nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^n} = \pm e^n \tag{62}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... , $N_e+2N_s+N_w+1$:

$$\frac{\partial J}{\partial x_k^n} = \frac{\partial J}{\partial x_{k\pm N_s\pm N_w\pm 1}^{n+1}} \tag{63}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... , N_e+N_s :

$$\frac{\partial J}{\partial x_k^n} = \frac{\partial J}{\partial y^n}\frac{\partial y^n}{\partial x_k^n} = \pm e^n \left(c_{\alpha,k} + \sum_{j=1}^{N_w} \frac{c_{\alpha,\,N_e+N_s+j}}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \right) + \sum_{j=N_e+N_s+N_w+2}^{N_e+2N_s+N_w+1} c_{j,k} \frac{\partial J}{\partial x_j^n} \tag{64}$$

Pour la copie n=1, nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^1} = \pm e^1 \tag{65}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... , $N_e+2N_s+N_w+1$:

$$\frac{\partial J}{\partial x_k^1} = \frac{\partial J}{\partial x_{k\pm N_s\pm N_w\pm 1}^{2}} \tag{66}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... , N_e+N_s :

le calcul des $\dfrac{\partial J}{\partial x_k^1}$ n'est pas utile.

VI.2.2.2 Calcul du gradient de J par rapport aux paramètres du réseau.

Disposant des dérivées calculées précédemment, il est à présent possible de déterminer le gradient de la fonction de coût par rapport à chacune des composantes du vecteur des paramètres ajustables.

Pour les coefficients directs sur la sortie :

$$\frac{\partial J}{\partial c_{\alpha j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n}\frac{\partial y^n}{\partial c_{\alpha j}^n} = \pm \sum_{n=1}^{N} e^n x_j^n \quad \text{avec } j = 1 , \dots , N_e+N_s \text{ et } \alpha = N_e+N_s+N_w+1 \tag{67}$$

Pour les coefficients directs sur les états :

$$\frac{\partial J}{\partial c_{k,j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x_k^n}\frac{\partial x_k^n}{\partial c_{k,j}^n} = \sum_{n=1}^{N} \frac{\partial J}{\partial x_k^n} x_j^n \tag{68}$$

avec $j = 1 , \dots , N_e+N_s$ et $k = N_e+N_s+N_w+2 , \dots , N_e+2N_s+N_w+1$

Pour les pondérations sur la sortie :

$$\frac{\partial J}{\partial c_{\alpha,j+N_e+N_s}} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial c^n_{\alpha,j+N_e+N_s}} = \pm \sum_{n=1}^{N} e^n \, \Phi_j(x^n) \tag{69}$$

avec $j = 1, \ldots, N_w$ et $\alpha = N_e+N_s+N_w+1$

Pour les pondérations sur les états :

$$\frac{\partial J}{\partial c_{k,N_e+N_s+j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} \frac{\partial x^n_k}{\partial c^n_{k,N_e+N_s+j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} \Phi_j(x^n) \tag{70}$$

avec $j = 1, \ldots, N_w$ et $k = N_e+N_s+N_w+2, \ldots, N_e+2N_s+N_w+1$

Pour le terme constant sur le neurone de sortie :

$$\frac{\partial J}{\partial c_0} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial c^n_0} = \pm \sum_{n=1}^{N} e^n \tag{71}$$

Pour les translations, j= 1, ... N_w et k=*1* , ... , N_e+N_s :

$$\frac{\partial J}{\partial m_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial m^n_{jk}} = \sum_{n=1}^{N} \left(\frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial m^n_{jk}} + \sum_{l=N_e+N_s+N_w+2}^{l=N_e+2N_s+N_w+1} \frac{\partial J}{\partial x^n_l} \frac{\partial x^n_l}{\partial m^n_{jk}} \right) \tag{72}$$

En remplaçant les dérivées par leurs expressions (déjà calculées), on obtient :

$$\frac{\partial J}{\partial m_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial m^n_{jk}} = \sum_{n=1}^{N} \left(e^n \frac{c_{\alpha,N_e+N_s+j}}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \pm \sum_{l=N_e+N_s+N_w+2}^{N_e+2N_s+N_w+1} \frac{\partial J}{\partial x^n_l} \frac{c_{l,N_e+N_s+j}}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \right) \tag{73}$$

Enfin, une factorisation permet d'alléger cette expression :

$$\frac{\partial J}{\partial m_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial m^n_{jk}} = \sum_{n=1}^{N} \frac{1}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \left(c_{\alpha,N_e+N_s+j} \, e^n \pm \sum_{l=N_e+N_s+N_w+2}^{N_e+2N_s+N_w+1} c_{l,N_e+N_s+j} \frac{\partial J}{\partial x^n_l} \right) \tag{74}$$

Pour les dilatations, j=1 , ... N_w et k=1 , ... , N_e+N_s :

$$\frac{\partial J}{\partial d_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial d^n_{jk}} = \sum_{n=1}^{N} \left(\frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial d^n_{jk}} + \sum_{l=N_e+N_s+N_w+2}^{N_e+2N_s+N_w+1} \frac{\partial J}{\partial x^n_l} \frac{\partial x^n_l}{\partial d^n_{jk}} \right) \tag{75}$$

En remplaçant les dérivées par leurs expressions (déjà calculées), on obtient :

$$\frac{\partial J}{\partial d_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial d^n_{jk}} = \sum_{n=1}^{N} \frac{z^n_{jk}}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \left(c_{\alpha,N_e+N_s+j} \, e^n \pm \sum_{l=N_e+N_s+N_w+2}^{N_e+2N_s+N_w+1} c_{l,N_e+N_s+j} \frac{\partial J}{\partial x^n_l} \right) \tag{76}$$

VI.2.2.3 *Commentaire sur le choix des variables d'état.*

Lors de la conception d'un modèle d'état d'un processus, il arrive que le cahier de charges exige qu'une des composantes du vecteur d'état soit la sortie du processus. Dans un tel cas, le calcul du gradient de *J* présenté ci-dessus se trouve légèrement modifié. La principale modification à apporter se situe au niveau du calcul du gradient de *J* par rapport à la sortie du réseau. Suivant la notation adoptée pour la présentation des relations ci–dessus, si la sortie est considérée comme une variable d'état du modèle, on la notera de la manière suivante :

$$y^n = x^n_{N_e+N_s+N_w+1} \tag{77}$$

La relation (64) devient dans ce cas :

$$\frac{\partial J}{\partial y^n} = \frac{\partial J}{\partial x^n_{N_e+N_s+N_w+1}} = \pm e^n + \frac{\partial J}{\partial x^{n+1}_{N_e+1}} \tag{78}$$

Remarquons que cette relation est très semblable à la relation (37) qui concerne le calcul du gradient dans le cas d'un réseau entrée–sortie.

Nous présentons en annexe de ce mémoire les modifications à apporter aux équations précédentes (de 59 à 76). Le calcul du gradient exposé dans cette annexe permet aisément le passage d'un modèle où la sortie est une variable d'état au cas où tous les états sont indépendants de la sortie.

VI.2.3 Calcul du gradient dans le sens direct.

La motivation pour le calcul du gradient de la fonction coût dans le sens direct est la même que dans le cas de la modélisation entrée-sortie : le calcul du Hessien approché nécessite la connaissance de grandeurs qui ne sont pas fournies par le calcul par rétropropagation.

Le principe du calcul de $\frac{\partial y^n}{\partial \theta}$ en prenant en considération toutes les copies est donné par la relation (42). On considère le problème de l'évaluation du second membre de la relation (43) mais cette fois dans le cadre d'un réseau d'état. On a :

$$\frac{\partial y^n}{\partial \theta^m} = \sum_{k=N_e+1}^{N_e+N_s} \frac{\partial y^n}{\partial x^n_k} \frac{\partial x^n_k}{\partial \theta^m} \text{ avec } m < n \tag{79}$$

En faisant la sommation de toutes les équations de la forme de (79) pour m allant de 1 à n–1, on obtient :

$$\sum_{m=1}^{n-1} \frac{\partial y^n}{\partial \theta^m} = \sum_{k=N_e+1}^{N_e+N_s} \frac{\partial y^n}{\partial x^n_k} \left(\sum_{m=1}^{n-1} \frac{\partial x^n_k}{\partial \theta^m} \right) \tag{80}$$

Étant donné que nous avons $x^n_k = x^{n\pm1}_{k+N_s+N_w+1}$, nous pouvons écrire les égalités suivantes :

$$\sum_{m=1}^{n\pm1} \frac{\partial x_k^n}{\partial \theta^m} = \sum_{m=1}^{n\pm1} \frac{\partial x_{k+N_s+N_w+1}^{n\pm1}}{\partial \theta^m} = \frac{\partial x_{k+N_s+N_w+1}^{n\pm1}}{\partial \theta} \tag{81}$$

En injectant ce résultat dans la relation (80), on aboutit à :

$$\sum_{m=1}^{n\pm1} \frac{\partial y^n}{\partial \theta^m} = \sum_{k=N_e+1}^{N_e+N_s} \frac{\partial y^n}{\partial x_k^n} \frac{\partial x_{k+N_s+N_w+1}^{n\pm1}}{\partial \theta} \tag{82}$$

Ce dernier résultat aboutit à l'écriture de la relation qui nous permet de calculer la dérivée de la sortie par rapport au vecteur θ :

$$\frac{\partial y^n}{\partial \theta} = \sum_{k=N_e+1}^{N_e+N_s} \frac{\partial y^n}{\partial x_k^n} \frac{\partial x_{k+N_s+N_w+1}^{n\pm1}}{\partial \theta} + \frac{\partial y^n}{\partial \theta^n} \tag{83}$$

Nous allons analyser sommairement la complexité de ce calcul.

$\frac{\partial y^n}{\partial \theta^n}$ est la dérivée de la sortie par rapport au vecteur des paramètres de la même copie. Les différentes composantes de ce vecteur sont données par les relations (20) à (24).

$\frac{\partial y^n}{\partial x_k^n}$ est la dérivée de la sortie par rapport à une variable d'état en entrée de la même copie. Le calcul est immédiat et est donné par la relation suivante :

$$\frac{\partial y^n}{\partial x_k^n} = \sum_{j=1}^{N_w} \frac{c_{\alpha,N_e+N_s+j}}{d_{jk}} \frac{\partial \Phi_j\left(x^n\right)}{\partial z_{jk}} + c_{\alpha,k} \tag{84}$$

où $\alpha = N_e+N_s+N_w+1$ est l'indice du neurone de sortie.

$\frac{\partial x_{k+N_s+N_w+1}^{n\pm1}}{\partial \theta}$ est plus délicat à évaluer puisqu'il s'agit d'une quantité qui implique le calcul des dérivées par rapport à chacune des copies d'indice inférieur ou égal. Étant donné que les variables d'états sont indépendantes de la sortie, il faut trouver une relation récurrente du type de (48) pour calculer à chaque copie les dérivées des états par rapport aux paramètres de manière économique. Par analogie avec la relation (48), nous proposons la relation suivante :

$$\frac{\partial x_l^{n\pm1}}{\partial \theta} = \sum_{k=N_e+1}^{N_e+N_s} \frac{\partial x_l^{n\pm1}}{\partial x_k^{n\pm1}} \frac{\partial x_{k+N_s+N_e+1}^{n\pm2}}{\partial \theta} + \frac{\partial x_l^{n\pm1}}{\partial \theta^{n\pm1}} \tag{85}$$

avec $n \geq 2$ et $l = N_e+N_s+N_w+2, \ldots, N_e+2N_s+N_w+1$

En résumé, l'algorithme de calcul du Hessien approché à l'aide des relations présentées ci-dessus est le suivant :

Pour n allant de 1 à N faire :

Pour m allant de 1 à N_s faire :

Exécution de la relation (85) en utilisant les résultats de cette même relation à l'étape précédente.

Fin de la boucle sur m.

Exécution de la relation (83).

Fin de la boucle sur n.

Construction du Hessien approché à partir de la relation (41).

Commentaire.

Les relations que nous venons d'établir pour l'apprentissage de réseaux d'ondelettes nécessitent un volume de calcul plus important que les relations équivalentes relatives aux réseaux de neurones à sigmoïdes.

Pour l'étude des exemples présentés dans le chapitre V, nous avons mis en œuvre le calcul du gradient par rétropropagation. De ce fait, l'apprentissage de nos réseaux d'état ne peut se faire qu'à l'aide de l'algorithme de BFGS.

V I.2.4 Initialisation des paramètres du réseau.

Un problème intéressant qui se pose est celui de l'initialisation des paramètres du réseau d'ondelettes dans le cas d'une modélisation d'état.

Deux cas peuvent se présenter :
- les états sont mesurables,
- les états ne sont pas mesurables.

Dans le cas où les états sont mesurables, la question peut être résolue de la même façon que dans le cas des réseaux pour la modélisation statique. En effet, les domaines de toutes les entrées étant connus (entrées externes et états), le calcul des translations et des dilatations initiales est immédiat (suivant la procédure proposée).

Dans le cas où les états ne sont pas mesurables, situation que nous avons choisi d'étudier dans nos exemples, le domaine des entrées du réseau (plus particulièrement les états) ne sont pas connus avant l'apprentissage. Le calcul des translations et des dilatations initiales est donc rendu délicat.

Nous avons initialisé ces paramètres en faisant l'hypothèse que les états sont dans un domaine de longueur comparable à celui de la sortie et centré en zéro. En fait, le but de cette procédure et d'éviter une initialisation aléatoire et de placer les ondelettes initialement dans le domaine de la variable d'entrée. Étant donné que les coefficients des neurones d'état sont initialisés à de petites valeurs, cette condition est vérifiée : en début d'apprentissage, les valeurs des états sont à l'intérieur du support des ondelettes.

VII. LE PROBLÈME MAÎTRE–ÉLÈVE ET LES RÉSEAUX D'ONDELETTES.

Le problème dit "maître–élève" consiste à engendrer des données à l'aide d'un réseau de fonctions "maître" dont les poids sont fixés, puis à retrouver ce réseau par apprentissage d'un réseau "élève" qui possède la même architecture. Ainsi, on est assuré que la fonction de régression recherchée (le réseau "maître") fait partie de la famille de fonctions du modèle (le réseau "élève"). L'intérêt de ce problème est qu'il permet de tester l'efficacité des algorithmes d'apprentissage, et notamment d'évaluer l'influence des minima locaux : en effet, si l'algorithme d'apprentissage converge en un temps raisonnable, et s'il ne trouve pas un minimum local, le réseau obtenu après apprentissage doit être le réseau maître, aux erreurs d'arrondi près.

Le système d'apprentissage pour un tel problème peut être illustré de la manière suivante :

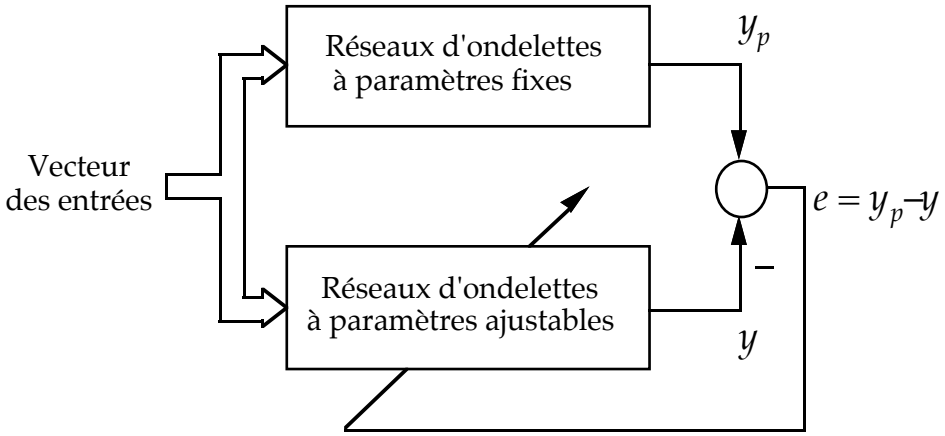


Figure 15. Système d'apprentissage pour le problème du "maître–élève".

Contrairement aux réseaux statiques présentés dans ce chapitre, on a supprimé la partie affine du réseau (sauf le terme constant sur le neurone linéaire de sortie que l'on conserve). La relation (12) donnant l'expression de la sortie du réseau est dans ce cas donnée par :

$$y = \sum_{j=1}^{N_w} c_j \Phi_j(x) + a_0 \tag{86}$$

Nous allons tout d'abord établir le résultat suivant : *l'existence des minima locaux de la fonction de coût ne dépend que de l'architecture du réseau maître, et ne dépend pas de la valeur des paramètres de celui-ci*. Nous préciserons ensuite les conditions des expériences numériques que nous avons menées, puis nous en décrirons et commenterons les résultats.

VII.1 Minima locaux de la fonction de coût.

Considérons un réseau maître réalisant la fonction $f(x, \theta_0)$, où x et θ_0 sont les vecteurs des variables et des paramètres respectivement, et un réseau élève $f(x, \theta)$. La fonction de coût minimisée pendant l'apprentissage est :

$$J = \frac{1}{2} \sum_{n=1}^{N} \left[f(x_n, \theta_0) \pm f(x_n, \theta) \right]^2 \tag{87}$$

où x_n désigne le vecteur des variables pour l'exemple n, et où N est le nombre d'éléments de l'ensemble d'apprentissage.
Le gradient de la fonction de coût a pour expression :

$$\frac{\partial J}{\partial \theta} = \pm \sum_{n=1}^{N} \left[f(x_n, \theta_0) \pm f(x_n, \theta) \right] \frac{\partial f(x_n, \theta)}{\partial \theta} \tag{88}$$

Pour un minimum (local ou global), obtenu pour un vecteur de paramètres θ_m, on a donc :

$$\left[\frac{\partial J}{\partial \theta} \right]_{\theta = \theta_m} = \sum_{n=1}^{N} \left[f(x_n, \theta_0) \pm f(x_n, \theta_m) \right] \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]_{\theta = \theta_m} = 0 \ . \tag{89}$$

Supposons que l'on fasse varier le vecteur des paramètres du réseau maître, et cherchons quelle variation il faut faire subir au vecteur θ_m pour qu'il corresponde toujours à un minimum. Il suffit pour cela d'écrire la différentielle totale du gradient :

$$\begin{aligned} \mathrm{d}\left[\frac{\partial J}{\partial \theta} \right]_{\theta = \theta_m} &= \sum_{n=1}^{N} \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]_{\theta = \theta_0} \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]_{\theta = \theta_m} \mathrm{d}\theta_0 + \\ \sum_{}^{N} \left\{ \left[f(x_n, \theta_0) \pm f(x_n, \theta_m) \right] \left[\frac{\partial^2 f(x_n, \theta)}{\partial \theta^2} \right] \right. & \left. \pm \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]^2_{\theta \quad \theta} \right\} \mathrm{d}\theta_m = 0 \ . \end{aligned} \tag{90}$$

On peut donc écrire :

$$\frac{\mathrm{d}\theta_m}{\mathrm{d}\theta_0} = \pm \frac{\displaystyle\sum_{n=1}^{N} \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]_{\theta = \theta_0} \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]_{\theta = \theta_m}}{\displaystyle\sum_{n=1}^{N} \left\{ \left[f(x_n, \theta_0) \pm f(x_n, \theta_m) \right] \left[\frac{\partial^2 f(x_n, \theta)}{\partial \theta^2} \right]_{\theta = \theta_m} \pm \left[\frac{\partial f(x_n, \theta)}{\partial \theta} \right]^2_{\theta = \theta_m} \right\}} \tag{91}$$

sous réserve que le dénominateur ne soit pas nul (il est nul si la fonction f est constante).

Pour les paramètres θ^l dont f dépend linéairement, on a $d\theta^l_m / d\theta^l_0 = 1$. Si le modèle est linéaire par rapport à tous les paramètres, il y a évidemment un seul minimum $\theta_m = \theta_0$.
Ainsi, si l'on connaît les minima de la fonction de coût pour une valeur donnée θ_0 des paramètres du réseau maître et pour un ensemble d'apprentissage donné, on peut, en principe, en déduire, par intégration de l'équation (91), les valeurs des minima pour toute autre valeur des paramètres du réseau maître. Changer les valeurs des paramètres du réseau maître ne change donc pas le nombre de minima locaux, mais seulement leur position dans l'espace des paramètres.
En conséquence, pour étudier l'influence des minima locaux sur l'apprentissage, nous choisirons, pour une architecture donnée, une seule valeur des paramètres.

VII.2 Choix de la séquence d'apprentissage.

Pour qu'un apprentissage soit efficace, il est nécessaire que la séquence d'apprentissage soit suffisamment riche pour représenter le comportement du processus. D'autre part, le nombre d'exemples constituant la séquence doit être très supérieur à celui des paramètres ajustables. Le réseau le plus volumineux que nous ayons considéré est constitué de 5 entrées et de 5 ondelettes, soit 56 paramètres.

Pour tous les exemples étudiés, nous avons considéré une séquence d'apprentissage formée par 2000 exemples. Nous faisons l'hypothèse que cette séquence représente suffisamment le processus (réseau maître) à apprendre dans le domaine des entrées choisi.

VII.3 Choix du domaine des entrées et des paramètres du réseau maître.

La question du choix du domaine des entrées est très importante puisqu'elle détermine le domaine dans lequel on veut modéliser le processus.

Dans le cas de réseaux d'ondelettes, ce choix est très lié à celui des paramètres du réseau maître. En effet, si l'on fait un choix pour le domaine des entrées, il faut choisir les paramètres du réseau maître de telle manière que les supports des ondelettes aient une intersection non nulle avec ces domaines. Sinon, la sortie y_p sera partout nulle.

Nous avons choisi pour les entrées des valeurs aléatoires suivant une distribution gaussienne centrée réduite (de moyenne nulle et de variance 1). Une analyse simple des entrées obtenues montre qu'elles sont toutes comprises dans un intervalle que l'on peut encadrer par [–4, 4]. Cette information est utile pour le choix des paramètres du réseau maître, comme nous le verrons dans le paragraphe suivant.

On choisit les translations des ondelettes du réseau maître de manière aléatoire (suivant une distribution uniforme) dans le domaine des entrées. Les dilatations sont également choisies de manière uniformément distribuée mais cette fois-ci dans le domaine [0.6, 2.6]. Cet intervalle est centré autour de la valeur $\frac{0.6+2.6}{2}=1.6$. Or, en reprenant la procédure d'initialisation utilisée pour l'apprentissage des réseaux d'ondelettes, on peut remarquer que cette valeur est celle des dilatations initiales (étant donné l'intervalle [–4, 4]). Ce choix est motivé par le fait qu'il correspond à des ondelettes dont les supports sont de l'ordre de la longueur de l'intervalle comprenant les entrées (c'est-à-dire [–4, 4]).

VII.4 Choix de l'algorithme et de l'initialisation du réseau.

Comme nous l'avons indiqué plus haut, nous avons le choix entre l'algorithme de BFGS et celui de Levenberg–Marquardt, afin d'éviter d'introduire, comme paramètre supplémentaire de notre étude, le nombre d'itérations de gradient simple à effectuer avant le démarrage de l'algorithme du second ordre.

L'initialisation du réseau s'effectue sur la base de la procédure présentée au paragraphe IV.2 de ce chapitre. Étant donné que les termes directs ont été retirés (voir relation 86), seules les pondérations des ondelettes et le terme constant sur le neurone linéaire de sortie sont initialisés de manière aléatoire.

VII.5 Approche adoptée pour l'étude du problème.

On se propose d'étudier les performances des réseaux d'ondelettes sur le problème maître–élève en prenant en considération l'influence du nombre des entrées et du nombre d'ondelettes dans la couche cachée.

Pour chaque architecture, nous choisissons un vecteur de paramètres pour le réseau maître, et nous effectuons vingt apprentissages avec vingt initialisations différentes pour les pondérations. Nous estimons qu'un apprentissage est un succès lorsque le vecteur des paramètres trouvé correspond exactement à celui du réseau maître (aux erreurs d'arrondi près).

VII.6 Résultats et commentaires.

Le tableau 10 présente, pour chaque architecture utilisée (caractérisée par le nombre d'entrées et le nombre d'ondelettes), le nombre d'apprentissages effectués avec succès avec un ensemble d'apprentissage constitué d'exemples pour lesquels les entrées suivent une loi gaussienne centrée réduite.

		Nombre d'ondelettes				
		1	2	3	4	5
Nombre d'entrées	1	20	20	0	0	0
	2	20	20	20	0	0
	3	20	20	16	0	0
	4	20	0	0	0	0
	5	20	0	0	0	0

Tableau 10. Résultats du problème maître–élève sur les réseaux d'ondelettes.

On observe que, au-delà de 3 entrées et 3 ondelettes, il devient pratiquement impossible de retrouver le réseau maître : l'apprentissage aboutit à des minima locaux de la fonction de coût. Les résultats obtenus avec d'autres distributions des entrées sont tout-à-fait analogues. Pour les réseaux à sigmoïdes, des expériences similaires ont montré que, au contraire, la probabilité de succès est d'autant plus grande que le réseau est plus grand [Stoppi97]. Les minima locaux semblent donc être plus gênants pour l'apprentissage des réseaux d'ondelettes que pour celui des réseaux de neurones. Il faut noter néanmoins que, dans un problème pratique, les données sont toujours entachées de bruit : on ne cherche donc pas à annuler l'erreur comme dans le cas du problème maître-élève, mais à trouver un minimum tel que la variance de l'erreur de modélisation soit égale à celle du bruit.

VIII. CONCLUSION.

Dans ce chapitre, nous avons présenté la modélisation statique et dynamique de processus à l'aide de réseaux d'ondelettes fondés sur la transformée en ondelette continue. Nous avons montré que les ondelettes peuvent être considérées comme des fonctions paramétrées (à paramètres continus), et qu'une combinaison linéaire d'ondelettes dont les centres et les dilatations sont ajustables peut, au même titre qu'un réseau de neurones,

constituer un modèle non linéaire de processus. Les paramètres de ce modèle peuvent être estimés à partir d'observations, de telle manière que la sortie du modèle approche la fonction de régression de la grandeur à modéliser.

La sortie du modèle n'étant pas linéaire par rapport aux dilatations et aux translations, l'estimation des paramètres doit être effectuée à l'aide d'algorithmes itératifs. Les ondelettes étant locales, le problème de l'initialisation des dilatations et translations est très important. Nous avons proposé une procédure d'initialisation simple qui prend en considération cette propriété.

Nous avons également montré que les réseaux d'ondelettes peuvent être utilisés pour la modélisation dynamique de processus, et peuvent constituer soit des modèles entrée-sortie, soit des modèles d'état. Pour ces deux types de modèles, nous avons établi les procédures de calcul du gradient, par rétropropagation et dans le sens direct. Les expressions obtenues nous ont montré que la complexité des calculs est plus importante que dans le cas de réseaux à fonctions dorsales.

Enfin, nous avons présenté une étude du problème "maître-élève" pour des réseaux d'ondelettes. Nous avons prouvé que, dans un tel cas, le nombre de minima ne dépend pas de la valeur des paramètres du réseau maître, mais seulement de son architecture et de l'ensemble d'apprentissage. Les résultats obtenus montrent que le nombre de minima locaux de la fonction de coût croît rapidement avec le nombre d'ondelettes et avec le nombre de variables du modèle. Nous présenterons dans le chapitre V des exemples de modélisation dynamique à l'aide de réseaux d'ondelettes.

CHAPITRE IV

Réseaux d'ondelettes (approche fondée sur la transformée discrète)

I. INTRODUCTION.

Dans le chapitre précédent, nous avons présenté des procédures d'apprentissage pour des réseaux d'ondelettes fondés sur la transformée en ondelettes continue. L'application de ces procédures est possible, car les paramètres des fonctions (et en particulier ceux des fonctions ondelettes) sont choisis de manière continue dans l'ensemble des réels.

Dans le présent chapitre, nous proposons des procédures de construction de réseaux d'ondelettes dont les paramètres sont à valeurs discrètes. On utilise donc ici la transformée en ondelettes discrète.

Nous présenterons tout d'abord le principe de la transformée en ondelettes discrète. Nous examinerons ensuite les méthodes que l'on peut mettre en œuvre pour la construction de réseaux d'ondelettes à paramètres discrets pour la modélisation de processus. Nous verrons que, comme on peut le prévoir, cette restriction donne moins de souplesse dans le choix d'un modèle que l'approche fondée sur la transformée continue ; de plus, pour des raisons inhérentes aux techniques utilisées, certains modèles-hypothèses possibles avec la transformée en ondelettes continue ne sont pas envisageables dans un contexte de transformée discrète.

Comme pour les ondelettes à paramètres continus, nous proposerons dans ce chapitre une procédure d'initialisation utilisant la transformée discrète. L'apprentissage sera ensuite effectué suivant les algorithmes présentés dans le chapitre précédent. Ainsi, les deux approches jouent un rôle complémentaire pour la construction d'un réseau d'ondelettes.

II. RÉSEAUX ISSUS SUR LA TRANSFORMÉE EN ONDELETTES DISCRÈTE.

Une transformée en ondelettes est dite discrète lorsque les valeurs des (translations et des dilatations) sont à valeurs discrètes (pas nécessairement entières).

Soit ϕ une ondelette mère (qui peut être la même que celle utilisée pour la transformée continue). Une famille Ω de fonctions obtenue à partir de ϕ peut être exprimée de la manière suivante :

$$\Omega(\alpha,\beta, x) = \left\{ \alpha^{m/2}\, \phi\left(\alpha^m x \pm n\, \beta\right), (m, n) \in \mathbf{Z}^2 \right\} \qquad (1)$$

où $\mathbf{Z}$ est l'ensemble des entiers relatifs. Notons que α et β sont des paramètres réels fixes qui définissent, avec ϕ, la famille Ω : α détermine l'échelle des dilatations et β détermine le pas des translations.

Une famille d'ondelettes est donc entièrement connue par la donnée du triplet (ϕ, α, β). Un membre de cette famille (c'est-à-dire une fonction) est désignée par le

couple *(m, n)*. Pour cette raison, on désignera dorénavant une ondelette de la famille de Ω ayant comme paramètres (*m*, *n*) par: $\phi_{m,n}(x)$.
Suivant la relation (1), nous avons :

$$\phi_{m,n}(x) = \alpha^{m/2}\,\phi\left(\alpha^m x \pm n\,\beta\right) \tag{2}$$

qui peut être réécrite de la manière suivante :

$$\phi_{m,n}(x) = \alpha^{m/2}\,\phi\left(\frac{x \pm n\alpha^{\pm m}\beta}{\alpha^{\pm m}}\right) \tag{3}$$

Cette relation est la même que la relation (1) du chapitre précédent, qui donne l'expression d'une ondelette dans un contexte de transformée continue, avec :

Translation : $m_j = n\,\alpha^{\pm m}\,\beta$

Dilatation : $d_j = \alpha^{\pm m}$

Ces relations montrent que la translation dépend de la dilatation, alors que ces quantités sont indépendantes dans le cas de la transformée continue.

II.1 Structures obliques et bases d'ondelettes orthonormales.

II.1.1 Ondelettes à variables continues.

Il a été démontré dans [Daubechies92] qu'une famille d'ondelettes $\phi_{m,n}(x)$ comme celles décrites plus haut possède la propriété de structure oblique de l'ensemble $L^2(\boldsymbol{R})$.
Rappelons cette propriété (déjà citée pour la transformée continue) : étant donné une fonction *f* de carré sommable, il existe deux constantes *c* et *C* positives et de valeurs finies telles que l'inégalité suivante soit vérifiée :

$$c\,|f|^2 \le \sum_{\phi_{m,n}\in\Omega} \left|\langle \phi_{m,n}, f\rangle\right|^2 \le C\,|f|^2 \tag{4}$$

Les valeurs de ces deux constantes (dites "limites de la structure oblique") donnent une indication sur la qualité de l'approximation de la fonction *f* par la famille d'ondelettes Ω [Zhang92] : en particulier, pour une somme finie d'éléments de Ω, plus ces constantes sont proches de 1, meilleure est la qualité de l'approximation.

A partir des structures obliques, on définit les *structures obliques étroites* (*"tight frames"* en anglais) de la manière suivante [Pati93] :

(a) Une structure oblique ayant des limites égales (c'est-à-dire $c = C$) est dite une *structure oblique étroite*.
(b) Une *structure oblique étroite* dont les éléments sont normés et ayant $c = C = 1$ est une base orthonormale.

Nous allons maintenant présenter un exemple d'une famille d'ondelettes qui constitue une base de fonctions orthogonales.

Avec $\alpha = 2$, $\beta = 1$ et un choix adéquat de l'ondelette mère (c'est-à-dire de ϕ) il est possible de construire une base d'ondelettes orthonormales. Elle peut s'exprimer de la façon suivante :

$$\Omega(2, 1) = \left\{ 2^{m/2} \phi\left(2^m x \pm n\right), (m, n) \in Z^2 \right\} \tag{5}$$

Les translations et les dilatations peuvent être obtenues en utilisant la réécriture de la relation (3).

La famille d'ondelettes orthonormales la plus connue est une base appelée "Système de Haar".

L'ondelette mère est une fonction définie par morceaux :

$$\phi(x) = \begin{cases} 1 \text{ si } x \in [0, \frac{1}{2}[\\ \pm 1 \text{ si } x \in [\frac{1}{2}, 1] \\ 0 \text{ ailleurs} \end{cases} \tag{6}$$

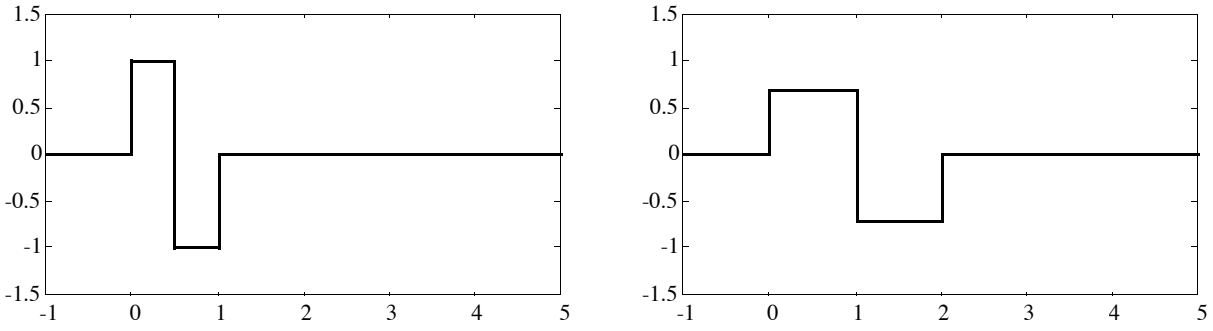
La figure 1 illustre trois ondelettes de cette famille :

$$\phi_{0,0}(x) = \phi(x) \qquad \text{(figure 1.a)}$$

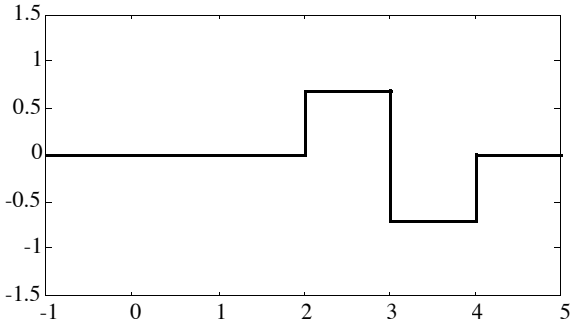
$$\phi_{\pm 1,0}(x) = \frac{\sqrt{2}}{2} \phi\left(\frac{x}{2}\right) \qquad \text{(figure 1.b)}$$

$$\phi_{\pm 1,1}(x) = \frac{\sqrt{2}}{2} \phi\left(\frac{x \pm 2}{2}\right) \qquad \text{(figure 1.c)}$$

Notons que le centre de la fonction (c'est-à-dire le paramètre de translation) n'est pas le centre de symétrie du graphe de la fonction mais la limite gauche de la partie non nulle de la fonction. Par exemple pour l'ondelette mère, le paramètre de translation est égal à 0 et non pas à $\frac{1}{2}$.



(a) (b)



(c)

Figure 1. Trois ondelettes du Système de Haar.

On peut facilement vérifier que ces fonctions possèdent bien la propriété d'orthogonalité.

La norme d'une fonction étant son carré scalaire, elle est égale ici à $\langle \phi_{m,n}, \phi_{m,n} \rangle = \alpha^m \alpha^{\pm m} = 1$. Le système de Haar est donc une famille d'ondelettes orthonormales.

Le système de Haar est l'une des rares familles orthonormales (dont l'ondelette mère possède une expression simple) que l'on peut trouver dans la littérature. Ceci est principalement dû à la difficulté du choix de l'ondelette mère.

II.1.2 Ondelettes à variables discrètes.

Dans tout ce qui précède, nous avons considéré l'orthogonalité de fonctions de variables continues, avec le produit scalaire défini par la relation (7). Dans le cas l'une des fonctions du produit scalaire est connue en un nombre fini de points (typiquement pour un problème d'approximation de fonction) la relation (7) devient une sommation discrète :

$$\langle \phi_{m,n}, \phi_{m',n'} \rangle = \sum_{\substack{i=1 \\ x^i \in A}}^{N} \phi_{m,n}(x^i)\, \phi_{m',n'}(x^i) \qquad (8)$$

où A désigne l'ensemble des points et N leur nombre. Le produit scalaire de deux fonctions, donc leur éventuelle orthogonalité, dépend ainsi du choix des points de A. Il est facile de vérifier (par exemple sur le système de Haar) qu'un choix quelconque du nombre de points et de leur distribution ne conduit pas à la conservation de l'orthogonalité.

Par exemple, dans [Yang96], des familles de fonctions orthogonales sont utilisées pour l'approximation de fonctions. Les séquences de données caractérisant les fonctions sont constituées d'un nombre impair de points régulièrement espacés. Ces deux précautions ont été prises afin de préserver l'orthogonalité des fonctions utilisées.

D'autre part, dans [Zhang95], une famille d'ondelettes orthogonales (dont l'ondelette mère n'est pas précisée) est utilisée. L'ondelette mère appartient à la famille de Daubechies. Les points de l'ensemble d'apprentissage sont uniformément répartis. Cette démarche se justifie par la quasi-orthogonalité des ondelettes.

Lorsque l'on cherche à approcher une fonction *connue*, dont on peut calculer numériquement la valeur en n'importe quel point, la restriction concernant le choix des points de l'ensemble A n'est pas gênante. En revanche, lorsque l'on veut modéliser un processus (c'est-à-dire approcher une fonction de régression hypothétique et *inconnue*), le choix des points d'échantillonnage est rarement libre : le concepteur du modèle ne dispose souvent que d'une base de données existante (par exemple des données économiques relatives à des années écoulées), ou bien le choix des mesures peut être contraint par les conditions normales de fonctionnement du processus, que le modélisateur n'a pas le droit de modifier librement (par exemple pour la modélisation d'un processus industriel opérationnel).

II.1.3 Choix de l'ondelette mère.

Comme nous l'avons souligné plus haut dans ce paragraphe, la propriété d'orthogonalité est obtenue par un bon choix des paramètres α et β mais aussi de l'ondelette mère ϕ. En effet, toute ondelette mère ne permet pas la construction de bases orthonormales.

Le système de Haar constitue une base de fonctions orthogonales, mais ces fonctions ne sont pas régulières[1], ce qui rend leur utilisation malaisée en tant qu'approximateurs. En effet, que ce soit en approximation de fonction ou en modélisation de processus, on utilise des familles de fonctions régulières.

Comme il est souligné dans [Cohen96], un des objectifs de la théorie des bases d'ondelettes est la construction de systèmes ayant les mêmes propriétés que le système de Haar, mais dont l'ondelette mère serait régulière. Ceci permettrait ainsi l'utilisation de ces bases pour résoudre des problèmes d'approximation de fonctions. Pour construire de manière systématique des ondelettes orthogonales, il convient de généraliser le principe de l'approximation multirésolution à celui de l'analyse multirésolution [Meyer90]. Plusieurs mathématiciens se sont penchés sur la construction d'ondelettes orthogonales s'appuyant sur la théorie de la multirésolution. Les premières ont été proposées par [Meyer85].

[1] La notion de régularité d'une ondelette ou plus généralement d'une fonction est liée à ses propriétés de dérivabilité. Une ondelette est dite régulière si elle est dérivable et sa dérivée continue.

Malheureusement, ces ondelettes mères orthogonales ne possèdent pas d'expression analytique simple, ce qui rend difficile l'utilisation de ces fonctions pour des problèmes d'approximation de fonctions. On se contentera donc, dans la suite de cette étude, de faire appel uniquement à des familles d'ondelettes constituant des structures obliques et particulièrement des *structures obliques étroites*, plutôt qu'à des bases orthonormales.

Les structures obliques peuvent être considérées comme des bases d'ondelettes redondantes. Leur principal inconvénient, par rapport à des bases orthonormales proprement dites, réside dans la perte de l'unicité de la représentation d'une fonction et la signification que peut avoir la pondération dans le cas d'ondelettes orthogonales.

Dans la suite de ce chapitre, nous allons nous intéresser aux familles d'ondelettes issues de la transformée discrète en tant qu'outil pour la modélisation de processus, en définissant des réseaux d'ondelettes fondés sur la transformée discrète.

II.2 Réseaux fondés sur la transformée discrète.

La transformation en ondelettes discrète, lorsqu'elle est effectuée de manière appropriée (choix adéquat de la discrétisation des paramètres de translation et de dilatation comme décrit ci-dessus), alors la transformation inverse existe.

De ce fait, toute fonction f de $L^2(\boldsymbol{R})$ peut être représentée comme une somme des éléments d'une base orthonormale ou d'une structure oblique d'ondelettes (comme celles décrites plus haut dans ce chapitre) suivant la relation:

$$f(x) = \sum_{(m,n)\in \boldsymbol{Z}^2} c_{mn}\, \phi_{m,\,n}(x) \tag{10}$$

Nous définissons un réseau d'ondelettes fondé sur la transformée en ondelettes discrète comme une somme finie de la forme de la relation (10), à laquelle on ajoute des termes directs.

La sortie d'un tel réseau est donc donnée par la relation suivante :

$$y = \psi(x) = \sum_{j=1}^{N_w} c_j\, \Phi_j(x) + \sum_{k=0}^{N_i} a_k\, x_k \tag{11}$$

où N_w est le nombre d'ondelettes et N_i le nombre d'entrées. $\Phi_j(x)$ est une ondelette multidimensionnelle obtenue par produit de toutes les ondelettes suivant chacune des entrées..

Ces réseaux sont, du point de vue de leur structure, identiques à ceux définis avec la transformée continue. La principale différence réside dans les méthodes de détermination des translations et des dilatations.

III. TECHNIQUES DE CONSTRUCTION DE RÉSEAUX D'ONDELETTES.

III.1 Impossibilité d'utiliser les techniques de gradient.

L'architecture des réseaux d'ondelettes fondés sur la transformée discrète ayant été définie, nous nous posons la question de la construction d'un modèle, constitué d'un réseau d'ondelettes, d'un processus donné. Comme précédemment, nous considérons que les seules connaissances sur le processus à modéliser sont constituées d'une (ou plusieurs) séquence(s) d'entrées et de sorties mesurées (si le processus est réel). Les paramètres à déterminer pour la construction du réseau sont

- le nombre d'ondelettes nécessaires pour atteindre une performance voulue,
- les valeurs à donner aux différents paramètres du réseau : paramètres structurels, pondérations des ondelettes et termes directs.

Comme dans toute méthode de modélisation par des fonctions paramétrées, la difficulté essentielle réside dans la détermination des paramètres du réseau. Ceux-ci prenant des valeurs discrètes, la minimisation d'un coût utilisant le gradient n'est pas envisageable. En revanche, on peut tirer profit du fait que les paramètres prennent des valeurs discrètes pour concevoir des méthodes de sélection des ondelettes dans un ensemble (bibliothèque) d'ondelettes discrètes. La performance du modèle ainsi conçu dépend du choix initial des ondelettes de la bibliothèque, et d'une sélection judicieuse dans cette bibliothèque.

III.2 Différentes approches pour construire un réseau d'ondelettes fondé sur la transformée discrète.

Contrairement aux techniques de gradient qui ne tirent pas parti des propriétés des ondelettes, des techniques qui utilisent les propriétés de ces fonctions et particulièrement le rôle de leurs paramètres structurels sont ici envisageables. Dans ce paragraphe, nous allons passer en revue les différentes techniques qui ont été proposées pour construire des réseaux d'ondelettes à partir de l'ensemble d'apprentissage. Pour chacune de ces techniques, nous préciserons les avantages et les inconvénients de chacune d'elles, dans la perspective de la mise au point d'une méthode simple à appliquer et peu coûteuse en temps de calcul. On distingue deux classes de techniques, selon qu'elles utilisent ou non une procédure de sélection.

III.2.1 Approches n'utilisant pas de procédure de sélection.

III.2.1.1 Technique fondée sur l'analyse fréquentielle.

Cette technique a été proposée dans [Pati93]. Elle repose sur l'estimation du spectre d'énergie de la fonction à approcher. Le domaine de fréquence contenant

le spectre d'énergie étant connu (il est obtenu en calculant la transformée de Fourier de la fonction à approcher), ainsi que le domaine des amplitudes des variables d'entrées couvert par la séquence d'exemples, on peut alors déterminer les ondelettes correspondant à ce domaine amplitude–fréquence.

Cette technique présente l'avantage de tirer parti des propriétés de localité des ondelettes dans les domaines spatial et fréquentiel. En revanche, elles présentent un inconvénient majeur, notamment pour les modèles multi-variables : le volume de calcul nécessaire à l'estimation du spectre de fréquence.

III.2.1.2 Technique fondée sur la théorie des ondelettes orthogonales.

Cette approche utilisant des bases d'ondelettes orthogonales a été proposée dans [Zhang95]. Étant donné le domaine des amplitudes des entrées de l'ensemble d'apprentissage, on choisit les ondelettes ayant leur centre à l'intérieur de ce domaine. Le nombre de dilatations différentes à considérer dépend de la performance désirée.

Cette technique présente l'avantage de mettre à profit la propriété d'orthogonalité des ondelettes. En revanche, sa mise en œuvre est malaisée, car, si l'on excepte le système de Haar (présenté plus haut dans ce chapitre), on ne connaît pas, à ce jour, d'expression analytique simple pour les ondelettes mères qui engendrent des familles de fonctions orthogonales. Dans un contexte de modélisation de processus, où la simplicité des fonctions utilisées et la parcimonie du réseau sont recherchées, cet inconvénient rend cette technique peu efficace.

III.2.1.3 Réseaux d'ondelettes pour un système adaptatif.

Cette technique a été proposée dans [Cannon95] pour la construction de réseaux d'ondelettes en vue de leur utilisation dans un système adaptatif de commande.

Une bibliothèque d'ondelettes est construite en considérant le domaine des valeurs des variables d'état du modèle. Le paramètre α du triplet (ϕ, α, β), qui détermine l'échelle des dilatations, est estimé en utilisant le spectre d'énergie de la fonction à approcher. Le réseau est constitué d'ondelettes de la bibliothèque sélectionnées et pondérées périodiquement. Les pondérations des ondelettes sont comparées à un seuil. Une fonction est gardée ou exclue du réseau suivant que sa pondération est supérieure ou inférieure à ce seuil.

Cette technique de construction de réseaux d'ondelettes peut être utilisée indifféremment pour la construction de modèles statiques ou dynamiques. Elle présente l'inconvénient de nécessiter l'estimation du spectre d'énergie de la fonction à approcher.

III.2.2 Approches utilisant une procédure de sélection.

III.2.2.1 Technique fondée sur la construction de structures obliques étroites.

Étant donné les limites théoriques auxquelles on se heurte pour la construction de réseaux d'ondelettes orthogonales (il n'existe pas d'expression analytique simple pour des ondelettes mères engendrant des bases orthonormales), on se propose ici d'utiliser des structures obliques. La question qui se pose alors est le choix des paramètres α et β. Pour éviter le calcul du spectre d'énergie de la fonction à approcher, on construit la bibliothèque à l'aide d'une *structure oblique étroite*. Les paramètres α et β sont alors respectivement égaux à 2 et à 1 [Juditsky94, Zhang97].

La bibliothèque est construite avec quatre ou cinq dilatations différentes. L'ondelette la plus large est celle dont le support a la taille du domaine des exemples. Les ondelettes retenues sont celles dont les centres sont à l'intérieur de ce domaine. Une méthode de sélection constructive ou destructive est ensuite appliquée aux ondelettes retenues dans la bibliothèque pour déterminer celles qui sont les plus significatives pour modéliser le processus étudié.

Dans [Zhang93, Zhang97], on propose d'appliquer une première réduction de la bibliothèque en éliminant les ondelettes comportant peu ou pas d'exemples sur leurs supports. Ces situations sont particulièrement fréquentes pour des modèles à plusieurs entrées où les exemples ne sont pas répartis de manière uniforme.

Cette technique présente l'avantage de procéder à une construction de la bibliothèque de manière simple, qui nécessite peu de calculs.

IV. PROPOSITION D'UNE PROCÉDURE DE CONSTRUCTION DE RÉSEAUX ET D'INITIALISATION DE L'APPRENTISSAGE.

Dans ce paragraphe, nous proposons une méthode de sélection d'ondelettes que nous mettons en œuvre

- pour la construction de réseaux d'ondelettes fondés sur la transformée discrète,
- pour l'initialisation, avant apprentissage, des translations et des dilatations de réseaux d'ondelettes fondés sur la transformée continue.

Dans les deux cas, la première étape consiste en la construction de la bibliothèque des ondelettes qui sont soumises à la procédure de sélection.

L'étape de la construction de la bibliothèque est fondée sur la théorie des *structures obliques étroites* d'ondelettes. Elle est donc semblable à celle présentée dans [Zhang97] excepté par le fait que, contrairement aux réseaux présentés dans

cette référence, chaque ondelette a des dilatations différentes suivant différentes entrées.

Ce choix présente l'avantage d'enrichir la bibliothèque, et d'obtenir une meilleure performance pour un nombre de fonctions donné. L'inconvénient introduit par ce choix concerne la taille de la bibliothèque. Une bibliothèque d'ondelettes ayant des dilatations différentes pour chaque entrée est plus volumineuse que celle dont les ondelettes possèdent la même dilatation suivant toutes les entrées. Ceci implique un coût de calcul plus élevé pendant l'étape de sélection. Néanmoins, la sélection d'ondelettes est souvent plus courte que l'apprentissage des dilatations et translations par les techniques de gradient utilisées pour les ondelettes à paramètres continus ; le coût supplémentaire introduit par des dilatations différentes peut donc être acceptable.

Dans ce qui suit, nous exposerons successivement la construction de la bibliothèque, la sélection des ondelettes dans cette bibliothèque, puis les procédures de construction et d'initialisation. On insistera particulièrement sur l'étape de construction de la bibliothèque. En effet, dans la littérature, la description des détails pratiques de cette étape est très rarement abordée en détail.

IV.1 Description de la procédure de construction de la bibliothèque.

Rappelons que l'on se propose de construire une bibliothèque d'ondelettes candidates pour la modélisation d'un processus. On dispose d'une séquence d'exemples répartis dans l'intervalle [*a, b*]. On considérera d'abord un modèle à une seule entrée. La généralisation pour un modèle à N_i entrées sera décrite au paragraphe IV.1.2.

IV.1.1 Famille engendrant la bibliothèque pour un modèle à une entrée.

Rappelons que la bibliothèque est engendrée à partir de la famille suivante :

$$\Omega(\alpha, \beta) = \left\{ \alpha^{m/2}\, \phi\left(\alpha^m x \pm n\, \beta\right), (m, n) \in \mathbf{Z}^2 \right\} \tag{12}$$

Étant donné que la construction est fondée sur une *structure oblique étroite*, nous avons $\alpha = 2$ et $\beta = 1$.

Une ondelette de cette famille avec les paramètres m et n s'exprime en fonction de l'ondelette mère de la façon suivante :

$$\phi_{m,\,n}(x) = 2^{m/2}\phi\left(\frac{x \pm 2^{-m} n}{2^{-m}}\right) \tag{13}$$

C'est donc une ondelette ayant pour centre $2^{-m}n$ et pour dilatation 2^{-m}.

Choix des dilatations.

Plus m est grand, plus le nombre d'ondelettes nécessaires est grand. Pour des raisons de taille de la bibliothèque, on se limite à trois dilatations successives

(c'est-à-dire trois valeurs entières successives du paramètre m). Il suffit donc de choisir la plus grande dilatation, ou la plus petite, pour que les deux autres soient déterminées.

Pour l'ondelette mère que nous utilisons dans nos exemples $\phi(x)=\pm x e^{\pm\frac{x^2}{2}}$, la valeur de la dilatation assurant que l'ondelette, centrée au milieu de l'intervalle (en $\frac{a+b}{2}$), ait sa partie utile aussi large que le domaine $[a, b]$ est : $0{,}2(b-a)$. Cette valeur de la dilatation est obtenue en estimant que la partie utile de l'ondelette coïncide avec le domaine où la sortie de l'ondelette est supérieure à 0.1 ou inférieure à –0.1. Elle est solution d'une équation non algébrique donnée par : $\frac{b \pm a}{2\,d} \exp\left(\pm\frac{1}{2}\left(\frac{b \pm a}{2\,d}\right)^2\right) = 0.1$.

Notons que cette propriété a été utilisée dans le chapitre précédent pour l'initialisation des réseaux fondés sur la transformée continue. On considère que cette valeur est celle de la plus grande dilatation. Les dilatations suivantes sont donc plus petites. Ceci peut se traduire par la relation suivante :

$$2^{\pm m} \le 0.2\,(b \pm a) \quad , \tag{14}$$

ce qui est équivalent à :

$$m \ge \pm \frac{\mathrm{Log}\left(0.2\,(b \pm a)\right)}{\mathrm{Log}\,2} \tag{15}$$

m est entier, alors que le second membre de cette inégalité ne l'est probablement pas. En pratique, la plus petite valeur de m à considérer (elle correspond à la plus grande dilatation) sera :

$$\left\lfloor \pm \frac{\mathrm{Log}\left(0.2\,(b \pm a)\right)}{\mathrm{Log}\,2} \right\rfloor + 1 \tag{16}$$

où l'opérateur [ ] désigne la fonction partie entière.

Les trois valeurs du paramètre m que l'on utilise pour la construction de la bibliothèque sont donc :

$$\left\{ \left\lfloor \pm \frac{\mathrm{Log}\left(0.2\,(b \pm a)\right)}{\mathrm{Log}\,2} \right\rfloor + 1,\ \left\lfloor \pm \frac{\mathrm{Log}\left(0.2\,(b \pm a)\right)}{\mathrm{Log}\,2} \right\rfloor + 2,\ \left\lfloor \pm \frac{\mathrm{Log}\left(0.2\,(b \pm a)\right)}{\mathrm{Log}\,2} \right\rfloor + 3 \right\} \tag{17}$$

Choix des translations.

Pour une dilatation donnée, on retient dans la bibliothèque toutes les ondelettes dont les centres sont à l'intérieur du domaine $[a, b]$. Pour une valeur de m donnée, cette condition peut s'exprimer de la manière suivante :

$$a \le 2^{\pm m}\, n \le b \tag{18}$$

Étant donné que l'on cherche ici à déterminer les valeurs possibles pour n, la condition précédente est équivalente à :

$$2^m a \leq n \leq 2^m b \tag{19}$$

Là aussi, étant donné que n est un nombre entier, les valeurs possibles sont (avec [ ] l'opérateur partie entière) :

$$\{ [2^m a]+1, [2^m a]+2 , \ldots , [2^m b] \} \tag{20}$$

En pratique, chaque fois que m est augmenté d'une unité, le nombre d'ondelettes ajoutées par $m+1$ à la bibliothèque est double de celui apporté par m. La bibliothèque est donc construite suivant un schéma pyramidal.

IV.1.2 Cas des bibliothèques pour modèles à plusieurs entrées.

Dans le cas d'un problème multidimensionnel, ce calcul est effectué pour chacune des entrées. Étant donné qu'une ondelette multidimensionnelle est le produit des ondelettes scalaires, le cardinal de la bibliothèque est égal au produit du nombre d'ondelettes suivant chacune des entrées.

IV.2 La méthode de sélection.

La bibliothèque étant construite, une méthode de sélection est ensuite appliquée afin de déterminer les ondelettes les plus significatives pour modéliser le processus considéré.

Soit M_w le nombre d'éléments dans la bibliothèque et N_w le nombre d'ondelettes dans le réseau. Pour sélectionner les N_w ondelettes qui permettent de constituer le modèle dont l'EQMA est la plus faible possible, l'idéal serait de calculer les EQMA obtenues avec tous les sous–ensembles de cardinal N_w qu'on peut former à partir d'un ensemble de cardinal M_w. Ce nombre de sous–ensembles est généralement très grand. De ce fait, on a recours à une méthode de sélection qui présente un moindre coût du point de vue du volume de calculs nécessaires.

La technique de sélection qu'on utilise effectue un classement des ondelettes de la bibliothèque sur la base de la procédure d'orthogonalisation de Gram–Schmidt. Cette procédure est proposée dans plusieurs références. Citons parmi elles [Chen89, Zhang93] dans le cadre de la construction de réseaux d'ondelettes, et aussi [Urbani95] dans un contexte de sélection d'architectures neuronales.

IV.2.1 Principe de la méthode de sélection par orthogonalisation.

Soit une séquence d'apprentissage formée de N exemples. On considère une bibliothèque contenant M_W ondelettes candidates. A chaque ondelette Φ_j on associe un vecteur dont les composantes sont les valeurs de cette fonction suivant les exemples de la séquence d'apprentissage. On constitue ainsi une matrice P dont l'expression est :

$$P = \begin{pmatrix} \Phi_1(x_1) & \Phi_2(x_1) & \cdots & \cdots & \cdots & \Phi_{M_w}(x_1) \\ \Phi_1(x_2) & \Phi_2(x_2) & & & & \vdots \\ \Phi_1(x_3) & \vdots & \ddots & & & \vdots \\ \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ \Phi_1(x_N) & \Phi_2(x_N) & \cdots & \cdots & \cdots & \Phi_{M_w}(x_N) \end{pmatrix} \quad (21)$$

On peut l'écrire de la façon suivante :

$$P = (p_1 \;\; p_2 \;\; \cdots \;\; p_{M_w}) \quad (22)$$

Avec

$$p_i = \left(\Phi_i(x_1) \;\; \Phi_i(x_2) \;\; \cdots \;\; \Phi_i(x_N) \right)^T \;\; avec \;\; i = 1 , \ldots , M_w \quad (23)$$

Les vecteurs p_i sont généralement linéairement indépendants, (car $N >> M_W$) et non orthogonaux.

Les vecteurs p_i engendrent donc un sous-espace vectoriel de dimension M_W. On estime que ces M_W vecteurs sont suffisants pour expliquer la sortie du processus à modéliser avec une précision satisfaisante. En d'autres termes, la projection du vecteur des sorties du processus Y_p dans cette espace correspond à une modélisation satisfaisante. La procédure de sélection consiste, en premier lieu, à classer les entrées par ordre de "pertinence" décroissante. Pour cela, on détermine, à chaque étape, l'ondelette qui a la plus grande projection sur la partie du vecteur des sorties qui n'est pas expliquée par les entrées précédemment classées. La figure suivante propose une interprétation géométrique de cette procédure pour un exemple de dimension 2 (les ondelettes sont représentées par des vecteurs).

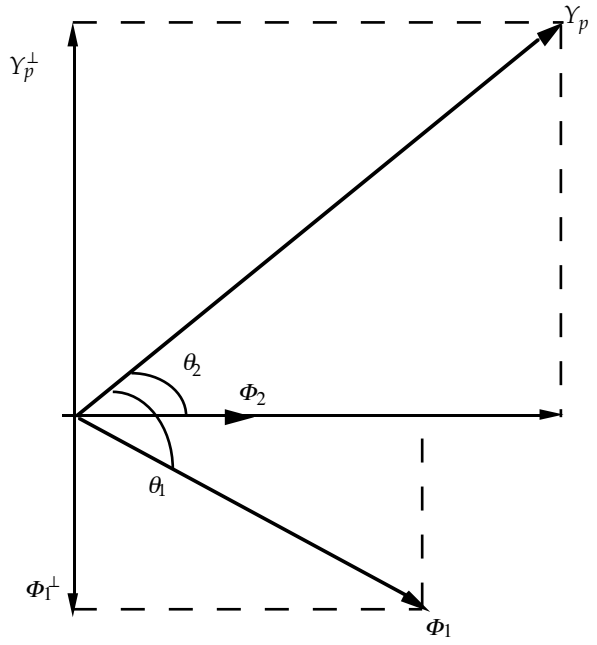


Figure 2. Interprétation géométrique de la sélection par orthogonalisation.

Sur cette figure, l'ondelette Φ_2 est celle qui explique le mieux le vecteur des sorties Y_p puisque l'angle qu'elle fait avec ce vecteur est plus petit que celui entre Φ_1 et Y_p. Elle est donc classée en premier rang par la procédure. Pour supprimer la partie de Y_p expliquée par Φ_2 , on projette Y_p et les vecteurs correspondants aux ondelettes non encore classées (ici Φ_1) dans l'espace orthogonal au vecteur que l'on vient de classer (ici Φ_2). On a représenté ces projections par $Y_p^{\perp}$ et $\Phi_1^{\perp}$.

IV.2.2 Cas des termes directs.

Étant donné que l'on s'intéresse à la détermination des ondelettes les plus significatives pour la modélisation d'un processus, la matrice P regroupe toutes les ondelettes de la bibliothèque, mais pas tous les régresseurs contenus dans le réseau donné par la relation (11). En effet, il manque les entrées qui sont pondérées par les termes directs. Ces régresseurs peuvent être ajoutés à la matrice P pour être sélectionnés. Mais, puisque l'on souhaite avoir des coefficients directs dans le réseau, ces régresseurs ne sont pas soumis à la procédure de sélection et sont systématiquement admis dans le réseau. Ce choix est motivé par le fait que la procédure de construction des réseaux (présentée dans le paragraphe suivant) sera également utilisée pour l'initialisation de réseaux fondés sur la transformée continue, qui, comme nous l'avons vu au chapitre précédent, possèdent des termes directs.

IV.3 La procédure de construction du réseau.

IV.3.1 Présentation de la procédure de construction.

Étant donné que l'on dispose des méthodes de construction de la bibliothèque d'ondelettes et de leur sélection, il reste à décrire les étapes de construction du réseau. Nous proposons le schéma suivant :

1. Effectuer l'apprentissage du réseau contenant uniquement les termes directs (la solution est celle des moindres carrés)..
2. Déduire une nouvelle séquence d'apprentissage dont les sorties sont définies comme les erreurs du réseau "affine". Cette séquence décrit donc la partie non modélisée par le réseau constitué par les termes directs.
3. Sélectionner un nombre N_W d'ondelettes de la bibliothèque préalablement construite, sur la base de la nouvelle séquence d'apprentissage.
4. Effectuer l'apprentissage du réseau complet, avec la séquence initiale, en ajustant les pondérations des ondelettes et les termes directs.

Lors du second apprentissage, on réajuste les termes directs pour ne pas aboutir à une solution sous–optimale. Une comparaison de leurs valeurs avant et après le second apprentissage montre qu'ils ne sont généralement pas modifiés.

Dans le cas où la taille du réseau n'est pas une contrainte, le nombre d'ondelettes N_w peut être augmenté tant qu'on n'observe pas de surajustement. En revanche, si l'on cherche à atteindre une performance donnée, on augmente N_w jusqu'à ce que cette performance soit atteinte.
Dans les deux cas, la recherche de N_w se fait selon un processus itératif. La figure suivante illustre le schéma d'application de la procédure :

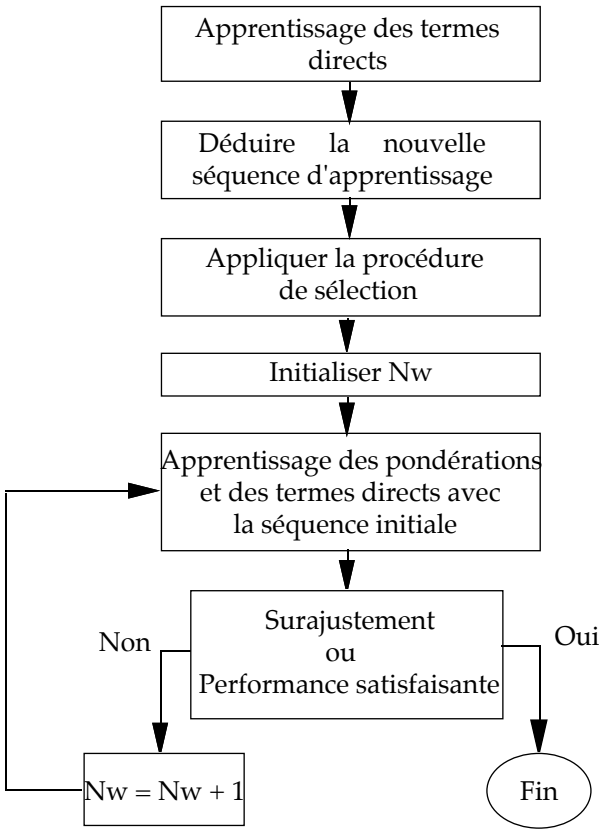


Figure 3. Schéma d'application de la procédure.

IV.3.2 Avantages et inconvénients de cette approche.

Nous avons présenté cette procédure comme une technique de construction de réseaux d'ondelettes fondés sur la transformée discrète.

Les réseaux obtenus sont plus volumineux, pour une même EQMP, que ceux, fondés sur la transformée continue, qui utilisent les techniques d'optimisation non linéaire (voir l'exemple présenté au paragraphe V.1). De plus, pour des modèles de trois entrées ou plus, le nombre des régresseurs dans la bibliothèque devient très grand.

En revanche, cette approche présente l'avantage d'utiliser les spécificités de cette famille de fonctions pour une construction de réseaux à moindre coût (temps de calcul très inférieur à celui d'un apprentissage utilisant une technique d'optimisation non linéaire).

Cette approche est donc conseillée dans le cas où l'on désire construire un modèle rapidement sans avoir recours à plusieurs apprentissages. En revanche, si la recherche d'un modèle parcimonieux est une priorité, ce type de réseaux n'est donc pas intéressant. Néanmoins, la procédure proposée peut être mise à profit d'une autre façon. C'est l'objet du paragraphe suivant.

IV.4 Autre application de la procédure : initialisation des translations et dilatations pour l'apprentissage de réseaux d'ondelettes à paramètres continus.

Une autre application de cette procédure est envisageable : l'initialisation des translations et des dilatations pour l'apprentissage de réseaux fondés sur la transformée continue[2].

IV.4.1 Principe de la procédure d'initialisation.

Nous avons vu dans le chapitre précédent, lors de l'étude de réseaux fondés sur la transformée continue, que l'étape d'initialisation des paramètres du réseau fait intervenir la propriété de localité des ondelettes. Une procédure d'initialisation simple fondée sur une heuristique a été proposée. Cette heuristique utilise peu les propriétés de ces fonctions et la théorie des structures obliques d'ondelettes.

Nous proposons ici d'utiliser la procédure qui vient d'être proposée pour l'initialisation des translations et des dilatations. Le schéma d'apprentissage utilisant cette approche pour l'initialisation du réseau se présente de la manière suivante :

1. Choisir le nombre N_W d'ondelettes constituant le réseau.
2. Utiliser la procédure de construction d'une bibliothèque et de sélection des ondelettes présentée ci-dessus pour sélectionner les N_W meilleures ondelettes expliquant la sortie du processus à modéliser.

[2] Dans [Lehtokangas95], une technique d'initialisation semblable a été proposée pour l'apprentissage de réseaux de neurones à fonctions sigmoïdes. Étant donné qu'il n'existe pas de théorie de construction de bibliothèque de neurones à fonctions sigmoïdes, cette bibliothèque est formée de neurones choisis aléatoirement. En ce qui concerne les réseaux d'ondelettes, cette approche de l'initialisation a été citée dans [Zhang93] sans être mise en œuvre.

3. Initialiser le réseau utilisant pour translations et dilatations celles des ondelettes sélectionnées. Les pondérations sont initialisées aléatoirement.
4. Effectuer l'apprentissage du réseau suivant les algorithmes décrits dans le cadre de réseaux fondés sur la transformée continue.

IV.4.2 Avantages et inconvénients de cette méthode d'initialisation.

Cette nouvelle méthode d'initialisation présente l'avantage d'utiliser la séquence d'apprentissage pour initialiser les translations et dilatations des ondelettes. La sélection étant fondée sur la minimisation du critère des moindres carrés (le même que celui utilisé lors de l'apprentissage), cette procédure est de nature à rapprocher le réseau d'un minimum de la fonction de coût en début d'apprentissage.

La méthode de sélection nécessite que les valeurs des entrées soient disponibles : elle n'est donc pas applicable pour l'initialisation de réseaux bouclés. Néanmoins, si les états sont mesurables, ou dans le cas d'une modélisation entrée–sortie, l'application de cette technique pour des réseaux non bouclés est envisageable.

V. ÉTUDE D'EXEMPLES.

On se propose de mettre en œuvre les procédures décrites ci-dessus pour la modélisation de processus à l'aide de réseaux d'ondelettes fondés sur, ou initialisés à l'aide de, la transformée discrète. Nous présentons tout d'abord une application de la procédure de construction, pour un modèle dynamique. Nous présentons ensuite deux exemples d'application de la procédure d'initialisation des dilatations et translations d'ondelettes à paramètres continus.

V.1 Exemple de construction de réseaux à l'aide de la procédure de sélection.

V.1.1 Présentation du processus.

Le système à modéliser est un processus simulé avec une équation d'ordre 1 :

$$y_p(k+1) = f\left(y_p(k),\, u(k)\right) = \left(1 \pm \frac{0.1}{1 + 5y_p^2(k)}\right) y_p(k) + \frac{e^{\pm y_p^2(k)/32}}{1 + 5\, y_p^2(k)}\, u(k) \pm e^{\pm 50\left[(y_p(k) - 0.5)^2 + (u(k) + 0.5)^2\right]} \tag{35}$$

Les séquences d'apprentissage et d'évaluation de la performance ont une taille N=1000. Les séquences d'entrée sont aléatoires, de distribution uniforme dans l'intervalle $[-1, 1]$. L'équation ayant été préalablement normalisée, la sortie est comprise dans le même intervalle. La figure suivante illustre la distribution des exemples de la séquence d'apprentissage dans le plan $(u(k), y_p(k))$.

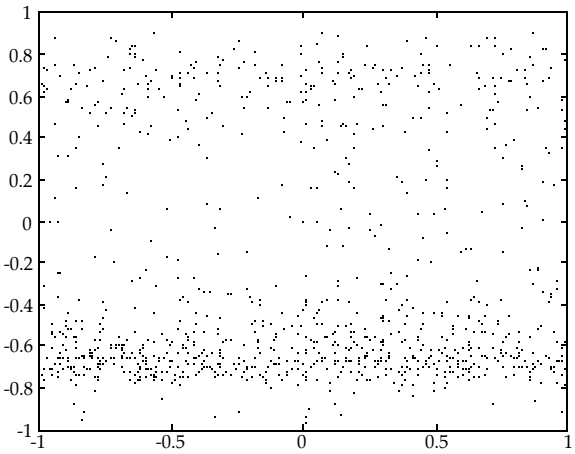


Figure 4. Répartition des exemples dans le plan $(u(k), y_p(k))$.

On effectuera également une modélisation du processus perturbé avec un bruit d'état de distribution uniforme et de variance 10^{-3}. L'équation de simulation est :

$$y_p(k+1) = f(y_p(k),\ u(k)) + w(k) \tag{36}$$

$\{w(k)\}$ est un bruit pseudo-blanc de moyenne nulle.

Le choix d'une perturbation bruit d'état coïncide avec le fait que, le prédicteur optimal pour l'apprentissage étant non bouclé, notre méthode d'initialisation qui utilise une procédure de sélection est applicable.

V.1.2 Construction d'un modèle dynamique à l'aide de la procédure.

On se propose donc de modéliser ce processus en utilisant un prédicteur pour l'apprentissage ayant l'expression :

$$y(k+1) = \psi\big(y_p(k),\ u(k)\big) \tag{37}$$

où la fonction ψ est réalisée par un réseau d'ondelettes comme dans la relation (11). Dans ce cas, nous avons une entrée externe ($N_e = 1$) et une entrée d'état ($N_s = 1$). Le nombre d'entrées du réseau est donc $N_i = N_e + N_S = 2$.

Les deux entrées étant dans l'intervalle [-1, +1], la construction de la bibliothèque est relativement simple puisque le calcul des dilatations et des translations est le même pour chacune des entrées. On l'effectue donc une seule fois (en général, les entrées sont réparties dans des intervalles différents ; pour se ramener à un seul intervalle et simplifier la construction de la bibliothèque, une normalisation des entrées peut être effectuée).

On construit la bibliothèque avec trois niveaux différents de dilatation dont le plus petit est donné par la relation (16) : $m = 2$. Les autres sont donnés par la relation (17), et sont donc 3 et 4. On aboutit à 31 ondelettes ayant leurs centres dans l'intervalle [-1, +1]. Étant donné que nous avons deux entrées, le nombre d'ondelettes multidimensionnelles est donc de $31 \times 31 = 961$.

On s'intéresse à la construction de tous les réseaux formés de 1 à 40 ondelettes. Au delà de cette valeur de N_W la contribution des ondelettes sélectionnées améliore peu les fonctions de coût. On exécute la procédure de sélection une seule fois et, pour chaque réseau, on calcule uniquement les pondérations des ondelettes et les coefficients de la partie affine.

V.1.2.1 Modélisation dynamique sans bruit du processus simulé.

La figure suivante illustre l'évolution de l'EQMA et de l'EQMP en fonction du nombre d'ondelettes dans le réseau.

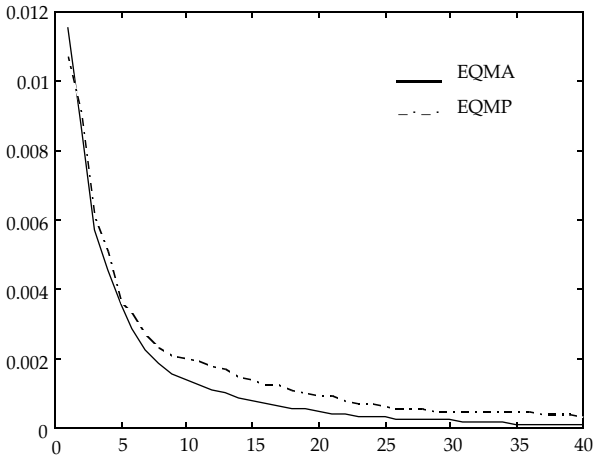


Figure 5. EQMA et EQMP en fonction du nombre d'ondelettes (sans bruit).

Une performance de 10^{-3} est atteinte avec un réseau de 19 ondelettes. Un tel réseau possède 98 paramètres, dont 76 (translations et dilatations) sont déterminés par la procédure de sélection et 22 (pondérations des ondelettes et termes directs) sont solution de la méthode des moindres carrés. La valeur correspondante de l'EQMA est 5×10^{-4}.

Nous avons mentionné, parmi les inconvénients de cette méthode de construction de réseaux, son manque de parcimonie par comparaison avec les réseaux non linéaires par rapport aux coefficients ajustables. A titre d'exemple, un réseau fondé sur la transformée continue atteint cette performance avec 7 ondelettes. En revanche, le temps d'apprentissage reste plus important.

V.1.2.2 Modélisation dynamique avec bruit du processus simulé.

On simule le processus avec un bruit d'état comme décrit dans le paragraphe V.1.1. La figure suivante illustre l'évolution des EQM en fonction du nombre de fonctions dans le réseau.

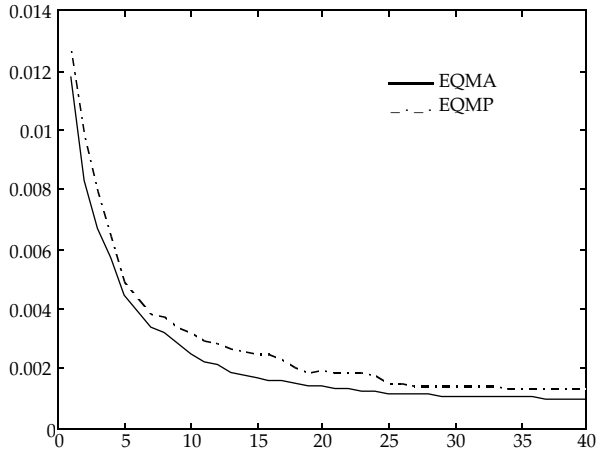


Figure 6. EQMA et EQMP en fonction du nombre d'ondelettes (avec bruit).

Pour un réseau de 19 ondelettes, les EQM d'apprentissage et d'évaluation de la performance sont égales à 1.4×10^{-3} et 1.9×10^{-3}. Avec 40 ondelettes, nous avons les chiffres suivants : 9.8×10^{-4} et 1.3×10^{-3}. Comme le nombre d'exemples est très grand par rapport aux nombres de paramètres, nous ne sommes pas confrontés au problème du surajustement.

V.1.2.3 Conclusion.

L'étude de cet exemple nous a montré que la mise en œuvre de la procédure est simple et nécessite peu de calculs (particulièrement la construction de la bibliothèque). De plus, si une normalisation des entrées est effectuée on peut construire une bibliothèque standard. Le prix à payer reste la taille du réseau (plus importante que celle des réseaux présentés au chapitre III où tous les paramètres sont ajustables) et aussi la croissance très rapide du nombre d'éléments de la bibliothèque dès que le modèle possède plus de trois entrées.

V.2 Exemple d'initialisation des translations et des dilatations de réseaux à l'aide de la procédure de sélection.

La méthode d'initialisation par sélection présentée dans le paragraphe IV.4 de ce chapitre est illustrée ici par la modélisation statique de deux processus. Le "processus 1" est un processus à deux entrées, tandis que le "processus 2" est le processus à une entrée présenté dans le chapitre III, paragraphe IV.3.

V.2.1 Processus 1.

V.2.1.1 Présentation du processus.

Le processus simulé dont nous étudions la modélisation statique a été utilisé dans [Hwang94] pour l'application d'un schéma d'apprentissage utilisant

la technique de la "Projection Pursuit Regression" (voir le chapitre II de ce mémoire pour un commentaire sur cette technique).

Le processus possède deux entrées et une sortie. Le comportement statique est simulé par la fonction suivante :

$$f(x_1, x_2) = 1.335\begin{pmatrix} 1.5\,(1 \pm x_1) + \exp(2\,x_1 \pm 1)\sin\left(3\,\pi\,(x_1 \pm 0.6)^2\right) \\ + \exp\left(3\,(x_2 \pm 0.5)\right)\sin\left(4\,\pi\,(x_2 \pm 0.9)^2\right) \end{pmatrix} \qquad (34)$$

On choisit pour les deux entrées le domaine [0,1].

La séquence d'apprentissage est constituée de 1000 points répartis suivant une distribution uniforme pour les deux entrées. La séquence d'évaluation de la performance est formée de 1600 points répartis suivant une grille régulière.

V.2.1.2 Initialisation de réseaux à l'aide de la procédure de sélection.

On se propose d'appliquer la procédure de sélection pour l'initialisation de réseaux d'ondelettes. Les pondérations initiales sont choisies suivant une distribution uniforme dans l'intervalle $[-10^{-2},10^{2}]$.

Pour chaque réseau, on effectue cent apprentissages, correspondant chacun à un tirage différent des pondérations initiales, à l'aide de l'algorithme de BFGS. On présentera l'histogramme des EQMA et des EQMP.

L'analyse de ces résultats nous permettra de comparer la performance de chacune de ces deux techniques d'initialisation et aussi d'évaluer leur robustesse relative vis-à-vis de l'initialisation aléatoire des pondérations.

On considère des réseaux constitués de 5, 10 et 15 ondelettes.

V.2.1.2.1 Modélisation du processus non bruité.

La figure 7 présente les histogrammes des EQMA et des EQMP après apprentissage du processus non bruité, avec intialisation des translations et dilatations par la procédure heuristique présentée dans le chapitre III. La Figure 8 présente les résultats obtenus, toutes choses égales par ailleurs, en utilisant la procédure d'initialisation par sélection présentée dans ce chapitre. La comparaison entre ces deux figures montre clairement que l'initialisation par sélection permet d'obtenir des résultats moins dispersés que l'initialisation heuristique. Les translations et dilatations étant, dans les deux cas, les mêmes pour tous les apprentissages, nous pouvons en conclure que l'initialisation par sélection confère à l'apprentissage une meilleure indépendance par rapport à l'initialisation aléatoire des pondérations des ondelettes.

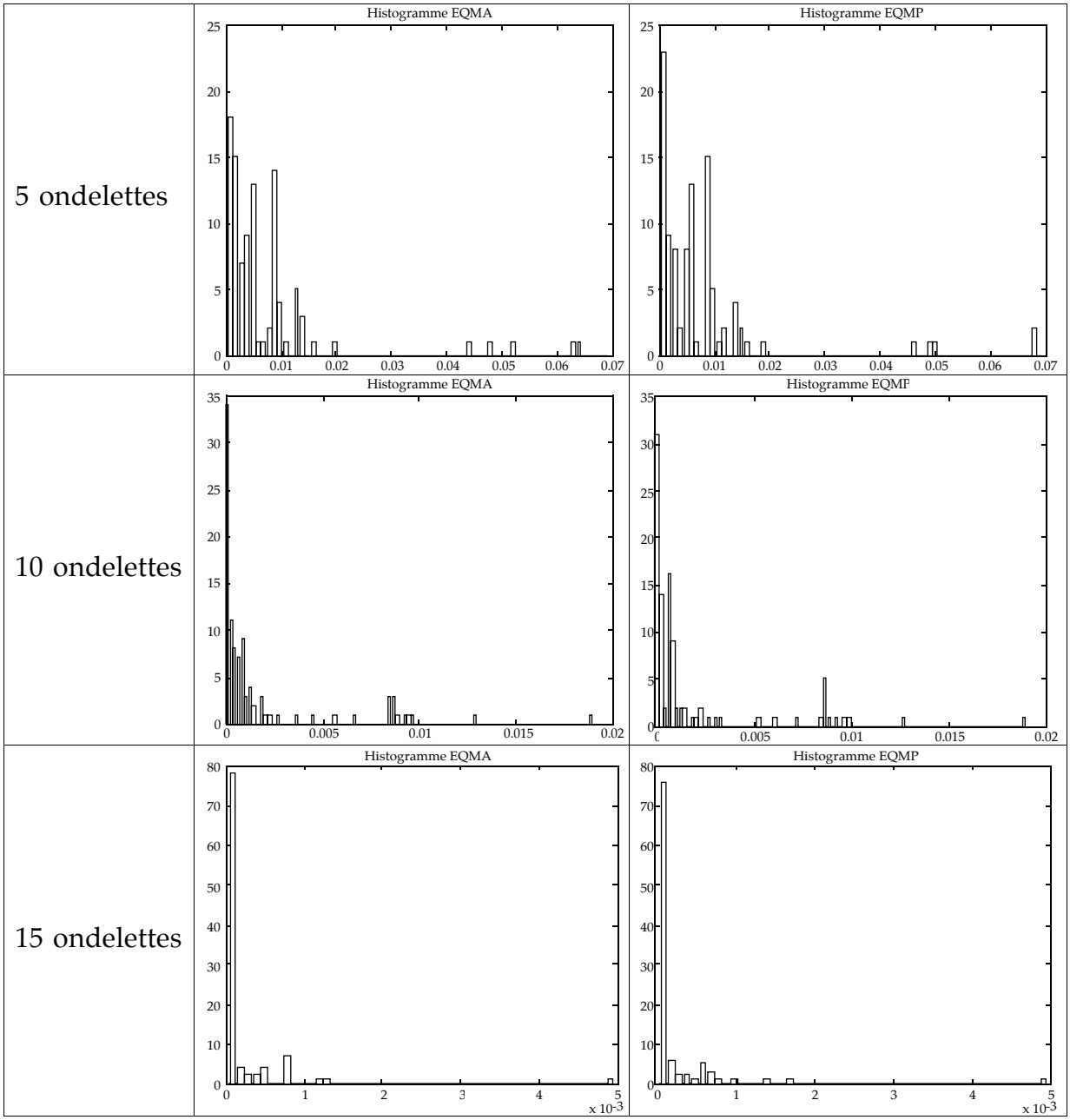


Figure 7. Histogrammes des EQMA et EQMP pour 100 apprentissages initialisés avec la procédure heuristique.

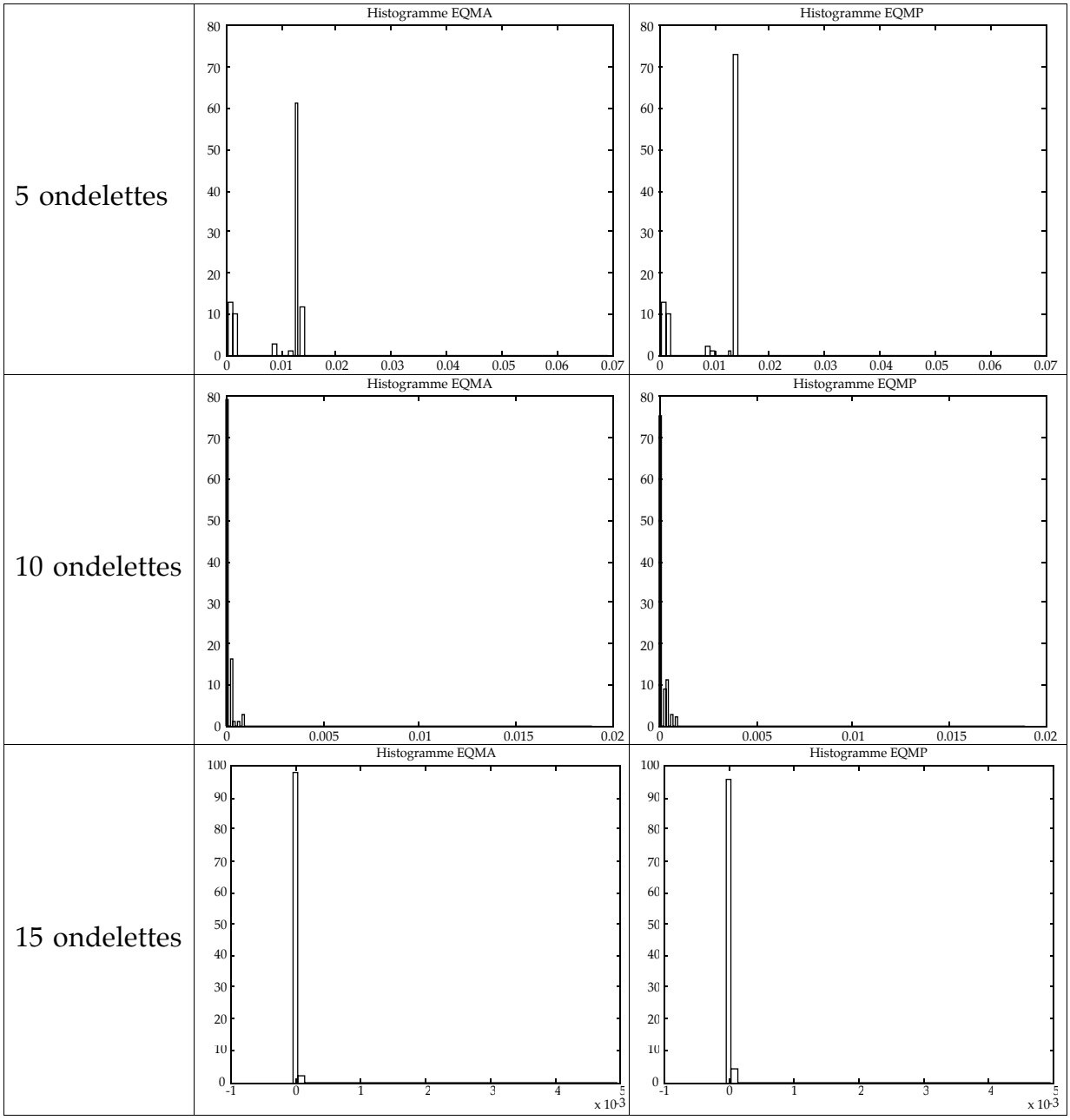


Figure 8. Histogrammes des EQMA et EQMP pour 100 apprentissages initialisés avec la procédure de sélection.

V.2.1.2.2 Modélisation avec bruit du processus.

Nous avons effectué les mêmes expériences numériques pour l'apprentissage du processus simulé avec un bruit additif de sortie, uniformément distribué de moyenne nulle et de variance 10^{-2}. Pour alléger la présentation, nous ne présenterons (Figure 9) que les résultats obtenus avec un réseau de 10 ondelettes, qui permet d'obtenir une EQMA et une EQMP égales à la variance du bruit (qui, ici, est connue puisqu'il s'agit d'un processus simulé).

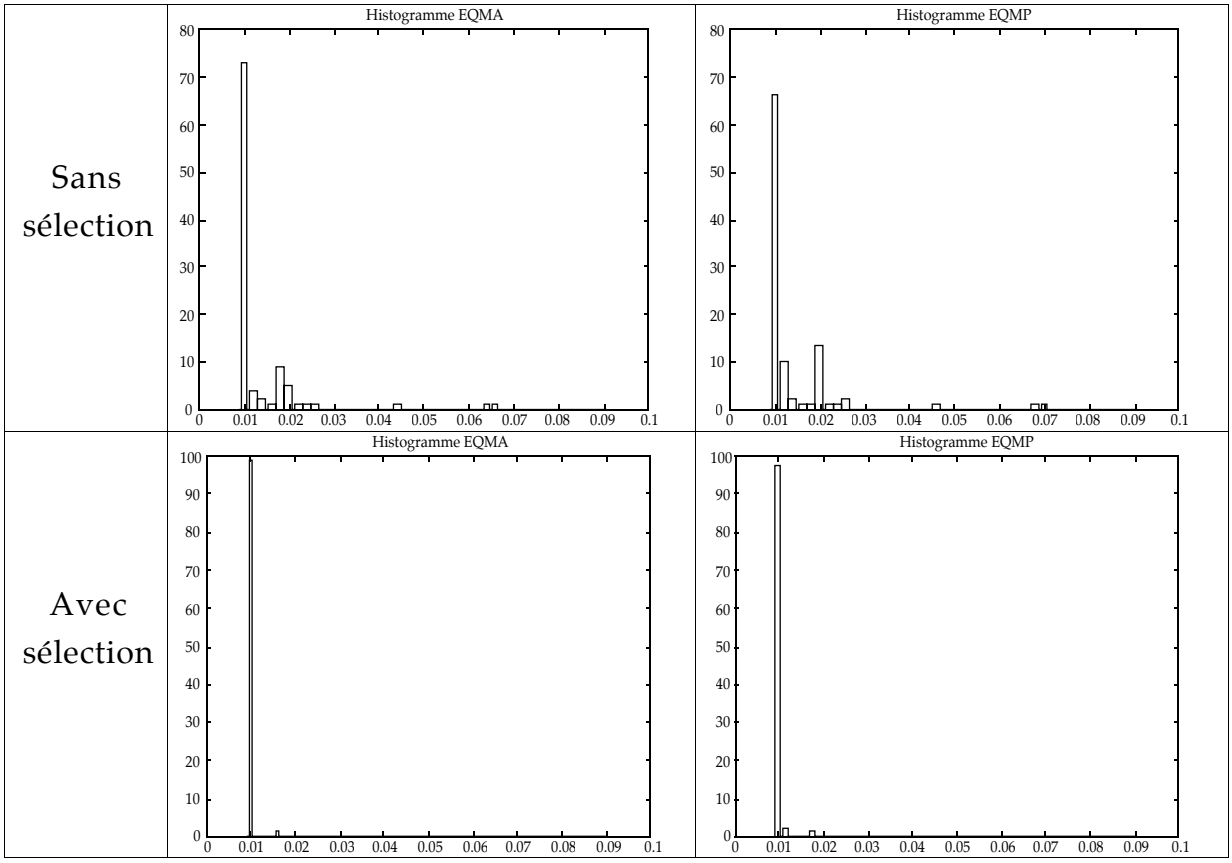


Figure 9. Histogrammes des EQMA et EQMP pour 100 apprentissages pour la modélisation avec bruit.

On observe, comme pour l'apprentissage du processus non bruité, que l'initialisation par sélection permet d'obtenir des résultats beaucoup moins dispersés, donc une meilleure indépendance vis-à-vis des initialisations aléatoires des pondérations. La valeur de l'EQMA correspondant à la variance du bruit est obtenue dans 97 % des cas.

V.2.2 Processus 2.

Ce processus a déjà été étudié au chapitre III. Nous rappelons ici pour mémoire (Figure 10) l'histogramme des EQMA et EQMP obtenues après apprentissage d'un réseau de 10 ondelettes par l'algorithme de BFGS, avec un ensemble de 300 points et initialisation à l'aide de la procédure heuristique. La Figure 11 représente les résultats obtenus dans les mêmes conditions, avec la procédure d'initialisation présentée dans ce chapitre.
Comme pour le processus précédent, l'utilisation de l'initialisation par sélection permet d'obtenir, avec une plus grande fréquence, les meilleures performances.

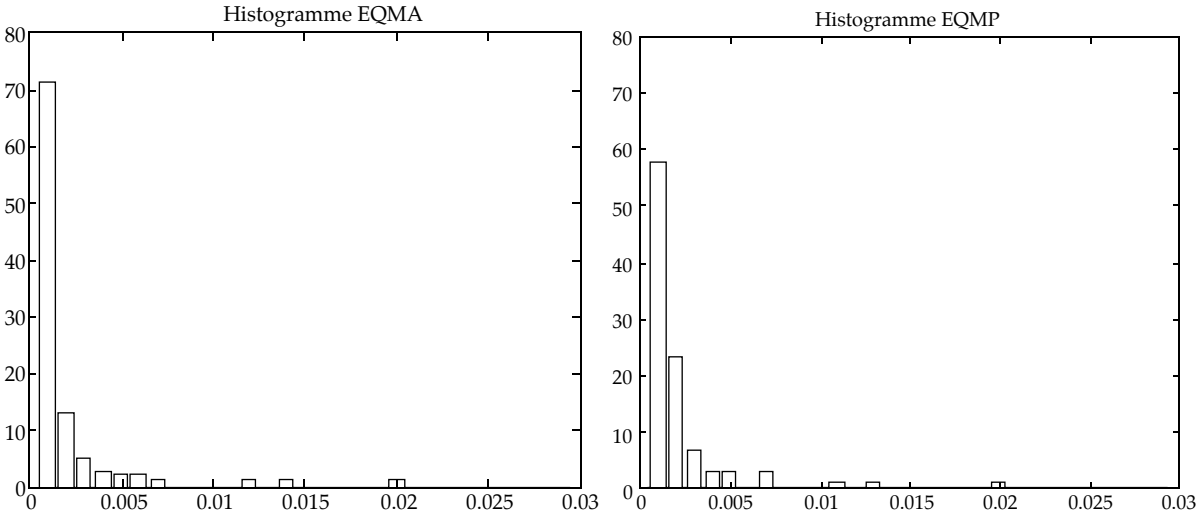


Figure 10. Histogrammes de l'EQMA et l'EQMP pour 100 apprentissages initialisés avec la procédure heuristique.

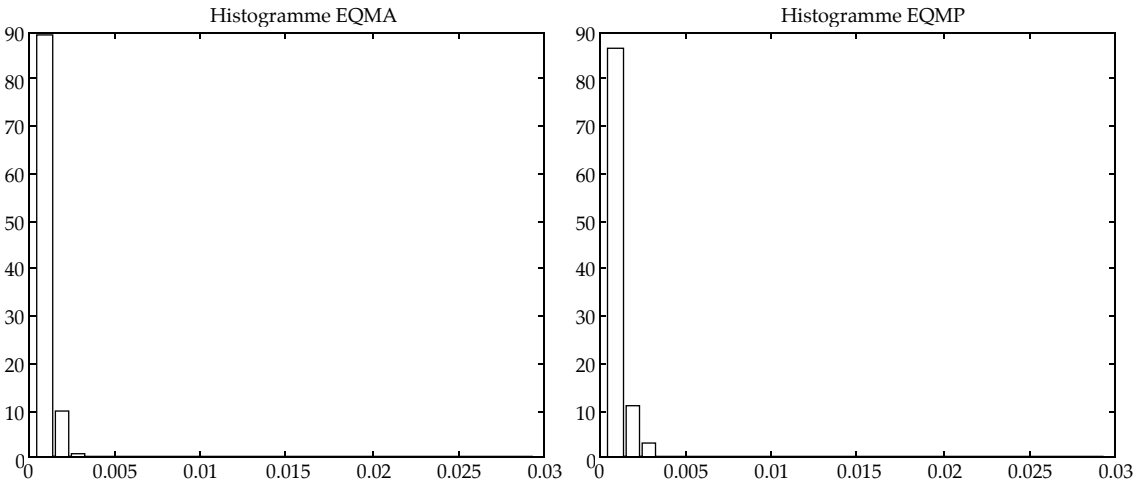


Figure 11. Histogrammes de l'EQMA et l'EQMP pour 100 apprentissages initialisés avec la procédure de sélection.

VI. CONCLUSION.

Dans ce chapitre, nous avons présenté les réseaux d'ondelettes fondés sur la transformée en ondelettes discrète. Il existe principalement les structures obliques et les bases d'ondelettes orthogonales. Le fait que les ondelettes mères orthogonales ne possèdent pas d'expression analytique simple les rend peu pratiquent pour des problèmes d'approximation de fonctions.

Nous avons proposé une procédure de modélisation de processus fondée sur la construction d'une *structure oblique étroite* et de la sélection des ondelettes les plus significatives. Cette procédure est également applicable comme méthode d'initialisation des réseaux fondés sur la transformée continue, étudiés dans le chapitre précédent.

Les résultats obtenus sur l'étude d'un exemple montrent que la construction de réseaux à l'aide de la procédure proposée peut être une solution

intéressante si l'on désire modéliser un processus sans contrainte de parcimonie, car sa mise en œuvre nécessite peu de calculs.

D'autre part, l'application de cette procédure pour l'initialisation des translations et dilatations d'ondelettes de réseaux fondés sur la tranformée continue a montré que les EQMA et EQMP présentent une dispersion plus faible que lors de l'utilisation d'une initialisation heuristique ; en d'autres termes, cette procédure permet une plus grande indépendance vis-à-vis de l'initialisation des pondérations des ondelettes.

CHAPITRE V

Étude de quelques exemples

I. INTRODUCTION.

Dans ce chapitre, nous présentons deux exemples de mise en œuvre des réseaux et algorithmes présentés dans ce mémoire pour la modélisation de processus. Le premier est simulé à partir d'une équation aux différences. Le second est un processus réel connu à partir d'une séquence de mesures.

Les prédicteurs que nous considérerons seront des réseaux de fonctions dorsales à sigmoïdes et des réseaux d'ondelettes fondés sur la transformée continue ; nous avons vu que les premiers permettent de réaliser des approximateurs plus parcimonieux que les modèles linéaires par rapport aux paramètres ajustables. D'autre part, il s'agit de modélisation dynamique de processus : on souhaite donc obtenir des modèles de simulation. Les prédicteurs construits à l'aide de réseaux d'ondelettes fondés sur la transformée discrète ne peuvent être candidats dans ce cas, comme nous l'avons vu dans le chapitre IV.

Nous présenterons, en premier lieu, un processus simulé. Il est en effet intéressant, d'un point de vue académique, de tester de cette manière des méthodes d'apprentissage ou des architectures de réseaux :

- le nombre d'exemples peut être arbitrairement grand,
- l'amplitude et la nature du bruit sont parfaitement connues,
- l'ordre du processus simulé est connu.

On s'affranchit ainsi des incertitudes, inévitables lorsque l'on modélise un processus réel, relatives au nombre et au choix des exemples, ainsi qu'au choix du modèle-hypothèse.

Nous présenterons ensuite la modélisation d'un processus réel, qui a été étudié en détail par d'autres auteurs.

Nous nous intéresserons essentiellement à l'apprentissage de réseaux bouclés. L'algorithme utilisé sera donc semi-dirigé (paragraphe V.1.2 du chapitre III).

Pour tous les réseaux, on effectue 50 apprentissages correspondant chacun à une initialisation différente

- des pondérations et des termes directs dans le cas des réseaux d'ondelettes (les translations et les dilatations étant initialisées suivant la technique proposée dans le paragraphe IV.2 du chapitre III)
- de tous les coefficients dans le cas d'un réseau de fonctions dorsales.

Le résultat retenu est celui présentant l'erreur la plus petite sur l'ensemble d'estimation de la performance (EQMP).

Chaque fois que nous serons amenés à comparer l'efficacité de deux algorithmes d'apprentissage, nous initialiserons les réseaux de manière identique avant application de chaque algorithme.

II. MODÉLISATION DE PROCESSUS SIMULÉS.

II.1 Présentation du processus simulé sans bruit.

Le processus dont nous effectuons la modélisation a été proposé dans [Urbani95] pour la validation d'une procédure de sélection de modèles neuronaux (réseaux de neurones à fonctions dorsales). Il est simulé à partir de l'équation aux différences entrée-sortie suivante :

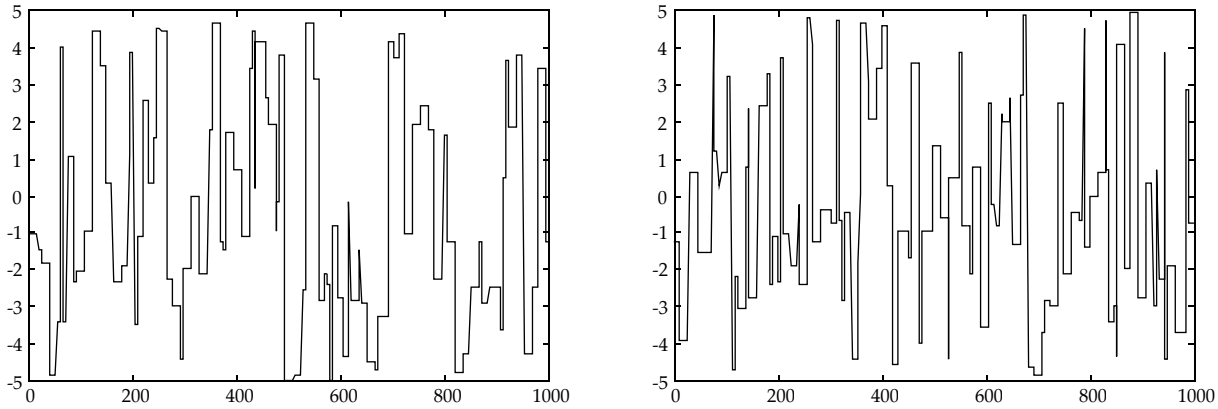
$$y_p(n) = f\left(y_p(n\text{-}1), y_p(n\text{-}2), u(n\text{-}1)\right) = \frac{24 + y_p(n\text{-}1)}{30} y_p(n\text{-}1) \pm 0.8 \frac{u(n\text{-}1)^2}{1 + u(n\text{-}1)^2} y_p(n\text{-}2) + 0.5u(n\text{-}1) \quad (1)$$

où $y_p(n)$ et $u(n)$ sont respectivement la sortie mesurée du processus et l'entrée de commande à l'instant n. Pour de faibles amplitudes de l'entrée de commande, comprises dans l'intervalle [–0.1, 0.1], l'équation aux différences ci-dessus est proche de l'équation linéaire du premier ordre suivante :

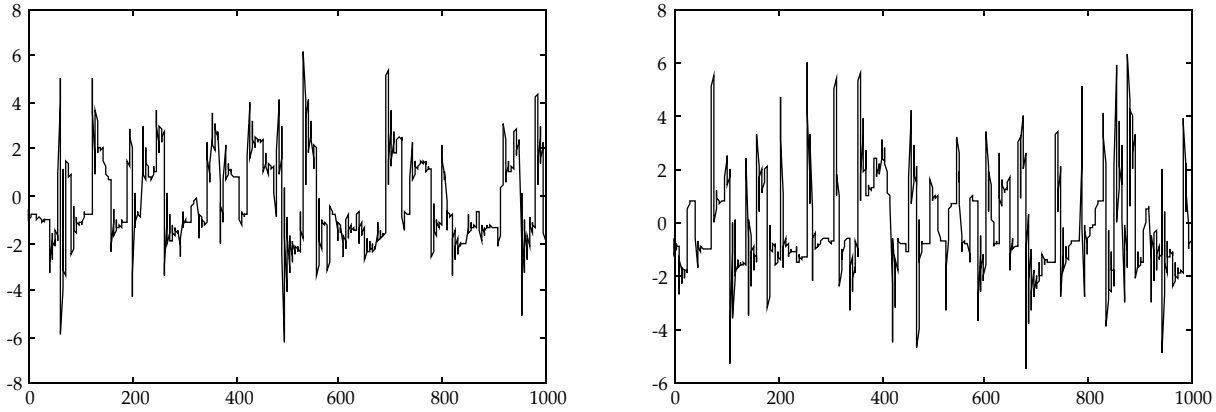
$$y_p(n) = 0.8\, y_p(n \pm 1) + 0.5\, u(n \pm 1) \quad (2)$$

Le comportement est alors celui d'un filtre passe-bas du 1er ordre de gain statique égal à 2.5. Lorsque l'entrée de commande est de plus grande amplitude, le comportement est non linéaire. Pour une entrée variant dans l'intervalle [–10, 10] le processus reste stable.

On choisit de modéliser ce processus simulé à l'aide de réseaux de fonctions dorsales et d'ondelettes. Pour cela, on calcule une séquence d'apprentissage et une séquence d'estimation de la performance, comprenant chacune 1000 exemples. L'entrée est une séquence de créneaux d'amplitude aléatoire comprise dans l'intervalle [-5, +5] et de durées aléatoires variant de 1 à 20 périodes d'échantillonnage. La figure 1(a) montre les séquences de l'entrée de commande pour l'apprentissage (à gauche) et pour l'estimation de la performance (à droite), et la figure 1(b) les séquences correspondantes de la sortie, utilisées pour l'apprentissage (à gauche) et l'évaluation de la performance (à droite).



(a)



(b)

Figure 1. Séquence de l'entrée de commande (a) et séquence de la sortie calculée (b) du processus pour l'apprentissage.

II.2 Modélisation du processus simulé non bruité.

Nous allons chercher tout d'abord à modéliser le processus simulé à partir de l'équation (1) sans introduire de bruit. En l'absence de bruit, les performances sont limitées par le nombre d'exemples (ainsi que leur distribution) et par l'efficacité de l'algorithme d'apprentissage.

Dans ce cas, comme nous l'avons indiqué dans le chapitre I, on peut indifféremment effectuer l'apprentissage d'un réseau non bouclé ou celui d'un réseau bouclé. Dans la mesure où nous sommes intéressés par la conception de modèles de simulation, nous avons choisi de réaliser l'apprentissage de réseaux bouclés. En effet, un modèle de simulation est destiné à prédire des *séquences* de valeurs des sorties, donc c'est nécessairement un réseau bouclé.

D'autre part, il nous est possible de choisir un *modèle entrée–sortie* ou un *modèle d'état*. Pour ce processus, nous considérerons uniquement sa modélisation par des prédicteurs de type entrée-sortie. Ce choix est motivé par le fait que le processus est simulé à partir d'une équation aux différences de type

entrée–sortie ; un exemple de modélisation d'état est présenté dans le paragraphe III.3 à l'occasion de la modélisation d'un processus réel.

L'ordre du processus simulé étant connu, ainsi que la mémoire sur l'entrée de commande, nous considérons pour ce processus l'apprentissage d'un prédicteur ayant l'expression suivante :

$$y(n) = \psi\left(y(n\pm1), y(n\pm2), u(n\pm1), \theta\right) \tag{3}$$

La fonction ψ est à approcher par un réseau de fonctions dorsales ou d'ondelettes fondé sur la transformée continue (puisque le réseau est bouclé) et θ est le vecteur des paramètres à ajuster.

II.2.1 Réseau prédicteur à fonctions ondelettes.

Nous considérons des réseaux d'ondelettes bouclés fondés sur la transformée continue, présentés dans le chapitre III de ce mémoire. Nous effectuons l'apprentissage de plusieurs architectures ayant un nombre d'ondelettes croissant afin de rechercher celle qui permet d'obtenir la meilleure performance (EQMP). Les apprentissages de ces réseaux sont effectués avec l'algorithme de BFGS et celui de Levenberg–Marquardt.

II.2.1.1 Apprentissage avec l'algorithme de BFGS.

Le tableau 1 présente les résultats obtenus lors de l'apprentissage de réseaux d'ondelettes bouclés à l'aide de l'algorithme de BFGS. Rappelons que, pour chaque architecture, 50 apprentissages ont été effectués, avec, à chaque fois, une initialisation aléatoire différente des pondérations et des termes directs (coefficients de la partie affine). Les paramètres de translation et de dilatation sont initialisés suivant la procédure présentée au chapitre III. Le résultat indiqué (pour chacune des architectures) est le meilleur obtenu sur les 50 apprentissages.

Nombre d'ondelettes	Nombre de paramètres	EQMA	Meilleure EQMP sur 50 apprentissages
1	11	$7{,}6\ 10^{-2}$	$1{,}5\ 10^{-1}$
2	18	$2{,}0\ 10^{-2}$	$3{,}6\ 10^{-2}$
3	25	$2{,}2\ 10^{-3}$	$6{,}7\ 10^{-3}$
4	32	$2{,}8\ 10^{-4}$	$1{,}3\ 10^{-3}$
5	39	$5{,}2\ 10^{-5}$	$2{,}9\ 10^{-4}$
6	46	$3{,}8\ 10^{-6}$	$2{,}9\ 10^{-5}$

Tableau 1. Résultats de la modélisation du processus simulé sans bruit avec réseaux d'ondelettes ; apprentissage à l'aide de l'algorithme de BFGS.

Au delà de 6 ondelettes, les réseaux contenant plus de fonctions ne permettent pas d'obtenir de meilleures performances.

II.2.1.2 Apprentissage avec l'algorithme de Levenberg–Marquardt.

Comme précédemment, nous nous intéressons à l'apprentissage de plusieurs architectures de réseaux d'ondelettes mais cette fois en utilisant l'algorithme de Levenberg–Marquardt pour l'ajustement des paramètres. Ainsi que nous l'avons indiqué au chapitre III, le calcul du gradient se fait dans le sens direct. L'initialisation des paramètres est la même que pour l'algorithme de BFGS.

Nombre d'ondelettes	Nombre de paramètres	EQMA	Meilleure EQMP sur 50 apprentissages
1	11	$7{,}6\ 10^{-2}$	$1{,}5\ 10^{-1}$
2	18	$2{,}1\ 10^{-2}$	$4{,}3\ 10^{-2}$
3	25	$2{,}5\ 10^{-3}$	$5{,}8\ 10^{-3}$
4	32	$9{,}7\ 10^{-4}$	$2{,}4\ 10^{-3}$
5	39	$5{,}0\ 10^{-5}$	$2{,}5\ 10^{-4}$
6	46	$3{,}4\ 10^{-5}$	$1{,}3\ 10^{-4}$
7	53	$2{,}2\ 10^{-6}$	$2{,}1\ 10^{-5}$

Tableau 2. Résultats de la modélisation du processus simulé sans bruit avec réseaux d'ondelettes ; apprentissage à l'aide de l'algorithme de Levenberg–Marquardt.

Ces résultats sont équivalents à ceux obtenus avec l'algorithme de BFGS et représentés sur le tableau 1. Notons que dans le cas de l'utilisation de l'algorithme de Levenberg–Marquardt, il est nécessaire d'utiliser un réseau de 7 ondelettes pour atteindre une performance de 10^{-5}.

II.2.2 Réseau prédicteur à fonctions dorsales.

Afin d'évaluer les performances des réseaux d'ondelettes fondés sur la transformée continue par rapport à celles que l'on peut obtenir avec une classe de réseaux possédant de bonnes propriétés de parcimonie, on se propose d'effectuer la modélisation de ce processus à l'aide de prédicteurs fondés sur des réseaux de fonctions dorsales. On choisit pour fonction d'activation la fonction sigmoïde (qui est la brique élémentaire des réseaux de neurones conventionnels) et l'on effectue l'apprentissage de réseaux bouclés dont la partie statique est constituée d'une couche de fonctions dorsales et d'un neurone de sortie linéaire (figures 4 et 7 du chapitre II).

II.2.2.1 Apprentissage avec l'algorithme de BFGS.

Comme précédemment, nous effectuons l'apprentissage de plusieurs architectures en augmentant à chaque fois le nombre de neurones. Le tableau 3 illustre les meilleurs résultats obtenus pour chacune des architectures.

Nombre de sigmoïdes	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	9	$1{,}1\ 10^{-1}$	$1{,}8\ 10^{-1}$
2	14	$7{,}1\ 10^{-2}$	$1{,}0\ 10^{-1}$
3	19	$1{,}1\ 10^{-3}$	$8{,}4\ 10^{-3}$
4	24	$3{,}9\ 10^{-4}$	$2{,}3\ 10^{-3}$
5	29	$4{,}5\ 10^{-6}$	$1{,}8\ 10^{-5}$
6	34	$4{,}2\ 10^{-6}$	$1{,}6\ 10^{-5}$

Tableau 3. Résultats de la modélisation du processus simulé sans bruit avec réseaux de sigmoïdes et algorithme de BFGS.

Là encore, une augmentation du nombre de fonctions n'améliore pas la performance, au-delà de 6 neurones cachés.

Les résultats, concernant l'EQMP, portés dans les tableaux 1 et 3, sont représentés graphiquement sur la figure 2 qui illustre l'évolution de l'EQMP en fonction du nombre de fonctions pour les deux types de réseaux. On ne constate pas de différence significative entre les deux types de réseaux.

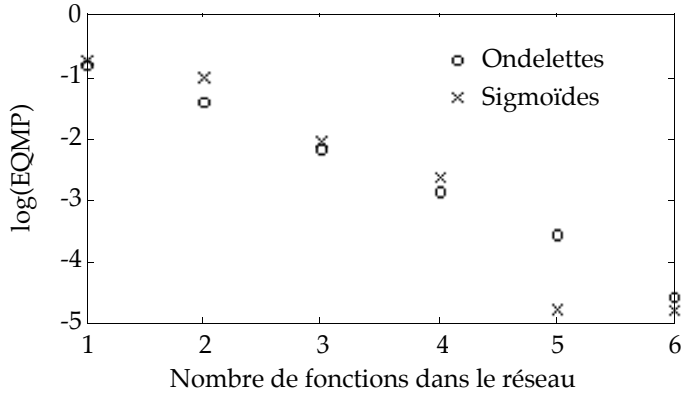


Figure 2. Évolution de la performance en fonction de l'architecture du réseau (BFGS).

II.2.2.2 Apprentissage avec l'algorithme de Levenberg–Marquardt.

On effectue également l'apprentissage de ces réseaux bouclés de sigmoïdes à l'aide de l'algorithme de Levenberg–Marquardt. Les remarques faites dans le

chapitre III concernant la nécessité de faire le calcul du gradient de la fonction de coût dans le sens direct s'appliquent également aux réseaux de fonctions dorsales.

Le tableau 4 illustre les meilleurs résultats obtenus pour chacune des architectures.

Nombre de sigmoïdes.	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	9	$1{,}1\ 10^{-1}$	$1{,}8\ 10^{-1}$
2	14	$7{,}2\ 10^{-2}$	$1{,}0\ 10^{-1}$
3	19	$1{,}8\ 10^{-3}$	$6{,}0\ 10^{-3}$
4	24	$5{,}7\ 10^{-4}$	$2{,}0\ 10^{-3}$
5	29	$6{,}4\ 10^{-6}$	$1{,}7\ 10^{-5}$
6	34	$3{,}5\ 10^{-6}$	$1{,}5\ 10^{-5}$
7	39	$1{,}8\ 10^{-6}$	$8{,}5\ 10^{-6}$

Tableau 4. Résultats de la modélisation du processus simulé sans bruit avec réseaux de sigmoïdes ; apprentissage avec l'algorithme de Levenberg–Marquardt.

Les résultats, concernant l'EQMP, portés dans les tableaux 2 et 4, sont représentés graphiquement sur la figure 3 qui illustre l'évolution de l'EQMP en fonction du nombre de fonctions pour les deux types de réseaux. Comme précédemment, on ne constate pas de différence significative entre les deux types de réseaux.

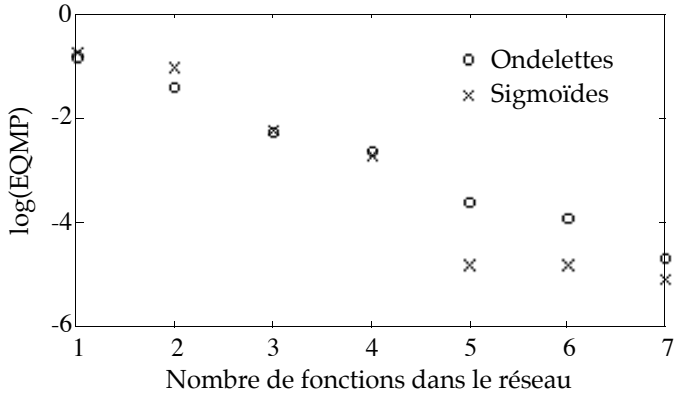


Figure 3. Évolution de la performance en fonction de l'architecture du réseau (Levenberg-Marquardt).

II.3 Modélisation du processus simulé avec bruit.

Nous nous proposons à présent de modéliser un processus bruité. Étant donné que l'on simule le processus, on a la possibilité de choisir la manière avec laquelle agit le bruit. Nous simulerons l'existence d'un bruit additif de sortie, puis celle d'un bruit additif d'état ; dans chacun des cas, nous ferons le choix du

modèle-hypothèse qui correspond au type de bruit qui est effectivement mis en œuvre dans les simulations, car nous nous intéressons, dans ce travail, à l'influence du choix des fonctions utilisées pour construire le modèle ; l'influence du choix du modèle-hypothèse (et notamment l'effet d'un choix erroné) a été étudiée dans [Nerrand94].

II.3.1 Modélisation du processus simulé avec bruit additif de sortie.

Lorsque l'on fait l'hypothèse d'un bruit additif en sortie (Output Error), le prédicteur optimal associé est bouclé. On considère un prédicteur d'ordre 2, dont la mémoire sur l'entrée de commande est de 1 période d'échantillonnage .Ce prédicteur peut être approché par un réseau de fonctions réalisant le modèle suivant:

$$y(n) = \psi\left(y(n\pm1), y(n\pm2), u(n\pm1), \theta\right) \tag{4}$$

Si la séquence d'apprentissage est suffisamment riche et représentative du comportement processus, si l'algorithme d'apprentissage est efficace, et si la taille du réseau est suffisante pour approcher la partie déterministe du processus avec une bonne précision, alors les EQMA et EQMP obtenues doivent être égales à la variance du bruit (c'est à dire à la partie non prédictible du comportement du processus).

Nous simulons le processus avec un bruit pseudo-blanc additif en sortie de distribution uniforme et de variance 10^{-2}. Les résultats de la modélisation sans bruit montrent qu'un réseau constitué de cinq fonctions permet d'effectuer une bonne approximation de la partie déterministe du processus (performance inférieure à la variance du bruit).

Nous avons donc effectué l'apprentissage de réseaux constitués de cinq fonctions (sigmoïdes ou ondelettes). Le tableau 5 illustre les meilleurs résultats obtenus sur 20 apprentissages.

	EQMA	EQMP
Réseaux d'ondelettes	$1{,}02\ 10^{-2}$	$1{,}05\ 10^{-2}$
Réseaux de sigmoïdes	$1{,}03\ 10^{-2}$	$1{,}07\ 10^{-2}$

Tableau 5. Résultats de la modélisation avec bruit de sortie.

Les deux types de réseaux permettent d'obtenir de façon quasiment identique une fonction de coût en fin d'apprentissage, et une performance, très proches de la variance du bruit.

II.3.2 Modélisation du processus simulé avec bruit d'état additif.

Nous considérons à présent la modélisation du processus simulé avec un bruit d'état additif (Equation Error) de variance 10^{-2}. Le prédicteur optimal associé est non bouclé. On effectue donc l'apprentissage de réseaux réalisant le modèle suivant :

$$y(n) = \psi\left(y_p(n\pm1), y_p(n\pm2), u(n\pm1), \theta\right) \tag{5}$$

On utilise des architectures de réseaux constitués de 5 fonctions (sigmoïdes ou ondelettes) et pour chaque architecture on retient la meilleure performance obtenue sur 20 apprentissages. Les résultats sont illustrés sur le tableau 6.

	EQMA	EQMP
Réseaux d'ondelettes	$9{,}68\ 10^{-3}$	$1{,}00\ 10^{-2}$
Réseaux de sigmoïdes	$9{,}67\ 10^{-3}$	$1{,}05\ 10^{-2}$

Tableau 6. Résultats de la modélisation avec bruit d'état.

Comme dans le cas d'un bruit de sortie additif, réseaux d'ondelettes et de fonctions dorsales aboutissent à des prédicteurs optimaux avec des précisions quasiment identiques.

II.4 Conclusion.

Dans la première partie de ce chapitre, nous avons effectué une modélisation de type entrée–sortie d'un processus simulé mono-entrée-mono-sortie d'ordre 2. Nous avons utilisé, dans des conditions identiques, des réseaux d'ondelettes fondés sur la transformée continue et des réseaux de fonctions dorsales sigmoïdes. Lors de la modélisation sans bruit puis avec brut, les deux types de réseaux montrent des performances équivalentes, que les apprentissages soient effectués à l'aide de l'algorithme de BFGS ou à l'aide de l'algorithme de Levenberg–Marquardt.

III. MODÉLISATION D'UN PROCESSUS RÉEL.

Les simulations précédentes ont permis d'étudier les performances de réseaux de neurones et de réseaux d'ondelettes en fonction du nombre de fonctions présentes dans ces réseaux, toutes choses égales et connues par ailleurs (ordre du modèle, mémoire sur l'entrée de commande, variance du bruit). Dans la plupart des applications réelles, on ignore :

- l'ordre nécessaire pour le modèle (c'est-à-dire la valeur du paramètre N_S défini au chapitre I),
- la mémoire sur l'entrée de commande (c'est-à-dire la valeur du paramètre N_e défini au chapitre I)

- la nature et la variance du bruit.

Lorsque l'on cherche à modéliser un processus réel, il est donc nécessaire, d'une part, d'essayer plusieurs modèles-hypothèses et de retenir celui qui semble le mieux adapté, et, d'autre part, de recourir à des techniques de sélection de modèles [Urbani95] pour trouver des valeurs satisfaisantes de N_s et N_w.

III.1 Présentation du processus.

Le processus dont nous nous proposons de faire la modélisation dans ce qui suit est l'actionneur hydraulique d'un bras de robot articulé. La sortie d'intérêt $y_p(n)$ est la pression d'huile de l'actionneur qui détermine la position du bras. L'entrée de commande qui agit sur la pression est l'ouverture d'une vanne $u(n)$. Les données relatives à ce processus ont été fournies par l'Université de Linköping ; ce processus a fait l'objet de modélisations "boîtes noires" de la part de plusieurs équipes [Sjöberg95].

Le processus est connu par une séquence de 1024 couples d'entrées et de sorties $\{u(k), y_p(k)\}$ mesurées. La première moitié de ces données (c'est à dire 512 points) est utilisée comme séquence d'apprentissage et la seconde moitié comme séquence pour l'estimation de la performance.

La figure 4 illustre la séquence de l'entrée de commande (a) et celle de la sortie mesurée (b) dont nous disposons.

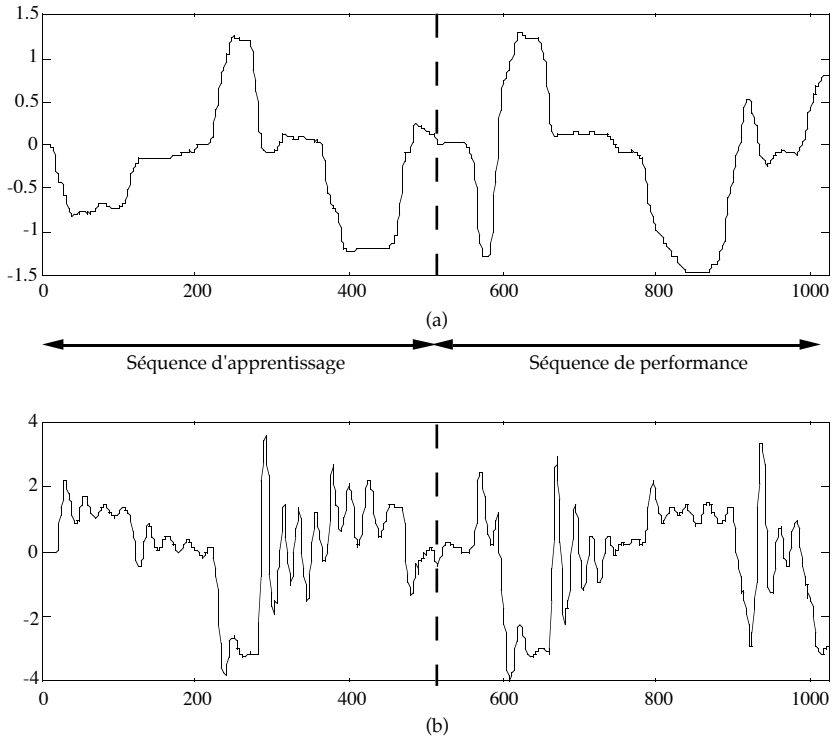


Figure 4. Séquences de l'entrée de commande (a) et de la sortie (b).

On se propose dans la suite d'effectuer une modélisation de ce processus avec des prédicteurs entrée–sortie et d'état fondés sur des réseaux d'ondelettes et de neurones à fonctions sigmoïdes.

III.2 Modélisation entrée–sortie.

Tout d'abord, nous nous proposons d'effectuer une modélisation entrée–sortie du processus. Comme toute modélisation, la première étape consiste à choisir un modèle-hypothèse. Le prédicteur optimal pour l'apprentissage associé à une hypothèse "bruit d'état" est non bouclé. Une étude antérieure de ce même processus[Rivals95b] a montré que les modèles construits à partir de l'hypothèse "bruit d'état" ont de mauvaises performances. Nous avons donc opté pour l'apprentissage de prédicteurs bouclés.

La deuxième question à résoudre concerne le choix de l'ordre du modèle et de la mémoire sur l'entrée de commande. Nous avons adopté une démarche qui consiste à considérer d'abord le modèle le plus simple, puis à le rendre plus complexe et à retenir celui qui présente la meilleure performance. Dans ce cas, nous partons d'un prédicteur avec $N_S = 2$ (le caractère oscillatoire de la réponse suggère que le modèle est au moins du second ordre) et $N_e = 1$.

III.2.1 Réseau prédicteur à fonctions ondelettes.

Nous commençons par présenter les résultats obtenus pour des prédicteurs fondés sur des réseaux d'ondelettes bouclés, comme ceux présentés dans le chapitre III. Comme pour les processus simulés, nous effectuons l'apprentissage de plusieurs réseaux en augmentant le nombre d'ondelettes.

III.2.1.1 Apprentissage avec l'algorithme de BFGS.

Le tableau 7 présente les résultats obtenus pour des apprentissages utilisant l'algorithme de BFGS. Comme pour le processus simulé, on effectue 50 apprentissages pour chacune des architectures avec à chaque fois une initialisation différente ; les résultats présentés correspondent aux apprentissages présentant les meilleures EQMP.

Nombre d'ondelettes.	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	11	0,25	0,30
2	18	0,11	0,13
3	25	0,13	0,15

Tableau 7. Résultats de la modélisation du processus réel avec réseaux d'ondelettes ; apprentissage à l'aide de l'algorithme de BFGS.

Pour plus de 3 ondelettes, la performance du modèle se dégrade. Le réseau présentant la meilleure performance est donc celui qui est constitué de deux ondelettes. C'est d'ailleurs, à notre connaissance, le meilleur résultat publié, relatif à un prédicteur entrée-sortie de ce processus [Rivals95b, Pucar95, Sjöberg95].

III.2.1.2 Apprentissage avec l'algorithme de Levenberg–Marquardt.

Dans les mêmes conditions que précédemment (mêmes séquences, mêmes architectures de réseaux, mêmes initialisations), on effectue des apprentissages de ces réseaux d'ondelettes en utilisant l'algorithme de Levenberg–Marquardt. Pour chaque réseau l'apprentissage présentant la meilleure EQMP sur les 50 est présenté sur le tableau 8.

Nombre d'ondelettes	Nombre de paramètres	EQMA	Meilleure EQMP sur 50 apprentissages
1	11	0,22	0,39
2	18	0,094	0,19
3	25	0,084	0,20
4	32	0,046	0,24

Tableau 8. Résultats de la modélisation du processus réel avec réseaux d'ondelettes ; apprentissage à l'aide de l'algorithme de Levenberg–Marquardt.

Comme avec l'algorithme de BFGS, le réseau de 2 ondelettes présente la meilleure performance.

La figure suivante, qui montre l'évolution de l'EQMA et de l'EQMP en fonction du nombre d'ondelettes, met en évidence que pour une architecture comprenant 3 ondelettes ou plus, on observe un phénomène de surajustement.

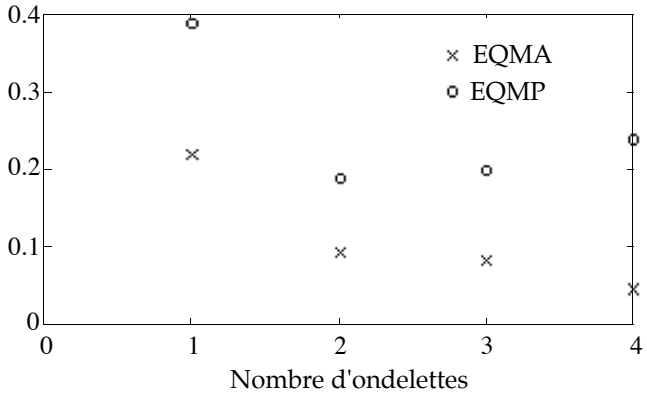
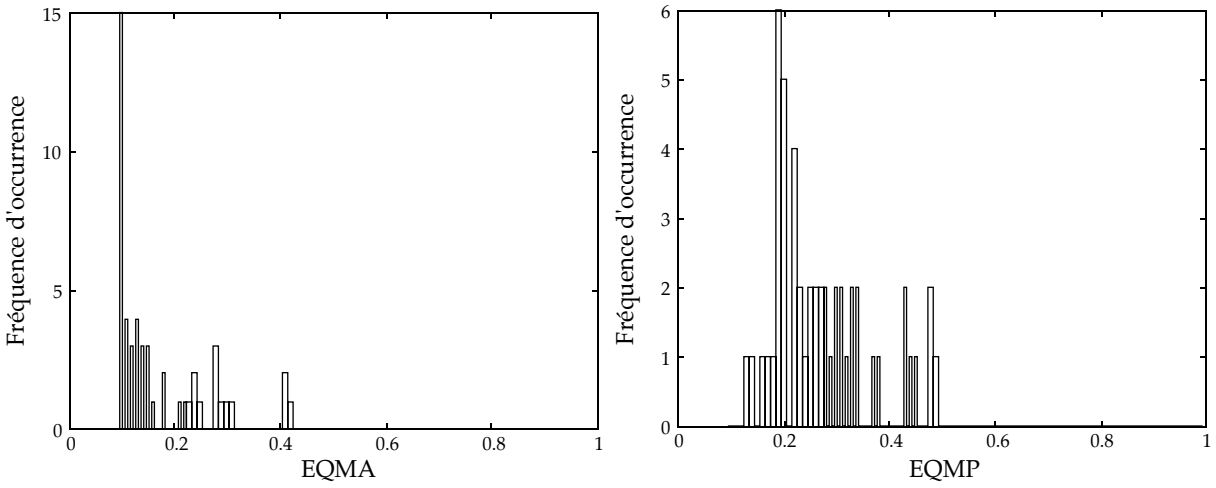


Figure 5. Évolution de la meilleure EQMP sur 50 apprentissages et de l'EQMA correspondante (apprentissages avec l'algorithme de Levenberg–Marquardt).

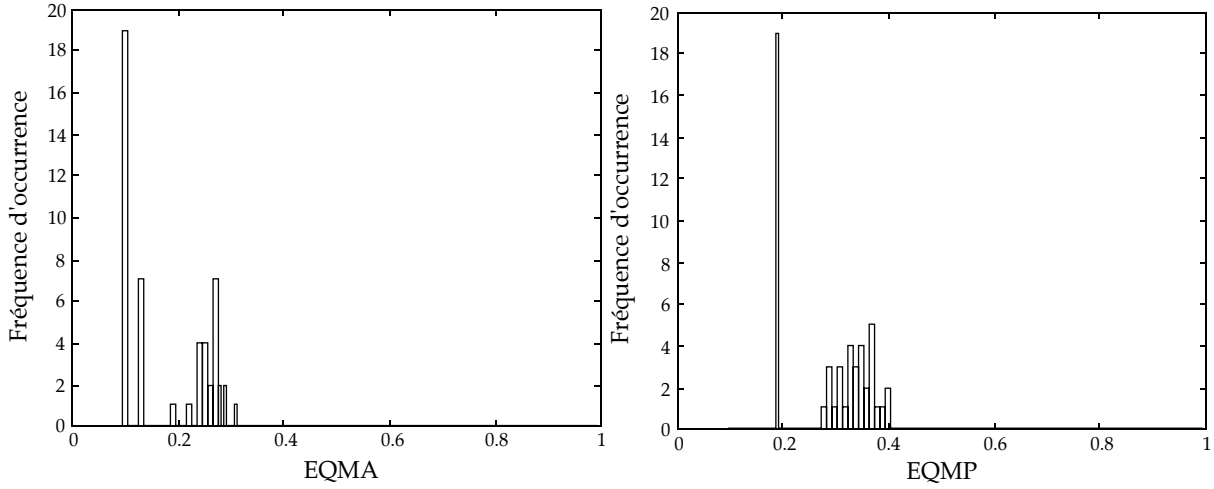
III.2.1.3 Fréquence d'occurrence du meilleur résultat.

Les résultats présentés dans les tableaux ci-dessus sont, pour chaque réseau, le meilleur apprentissage obtenu sur 50. La fréquence d'occurrence du meilleur résultat, parmi tous les essais effectués, peut constituer un élément de choix entre plusieurs algorithmes. Pour le réseau de 2 ondelettes, nous avons représenté, sur la figure 6, les histogrammes d'apparition des différentes valeurs de la fonction de coût en fin d'apprentissage (a) et de l'estimation de la performance (b) obtenues parmi les 50 apprentissages en utilisant l'algorithme de BFGS. La figure 7 illustre ces deux histogrammes dans le cas de l'utilisation de l'algorithme de Levenberg–Marquardt.



(a) (b)

Figure 6. Histogrammes des EQMA (a) et des EQMP (b) de réseaux d'ondelettes avec apprentissage à l'aide de l'algorithme de BFGS.



(a) (b)

Figure 7. Histogrammes des EQMA (a) et des EQMP (b) de réseaux d'ondelettes avec apprentissage à l'aide de l'algorithme de Levenberg–Marquardt.

On observe que, du point de vue considéré ici, l'algorithme de Levenberg Marquardt possède, pour cet exemple qui met en jeu un petit nombre d'ondelettes, une efficacité supérieure à celle de la méthode de BFGS : une des valeurs de l'EQMP (= 0.19) ainsi que l'EQMA (= 0.094) qui lui correspond sont obtenues avec une plus grande fréquence que les autres, et il s'agit précisément du meilleur apprentissage retenu dans le tableau 8 pour le réseau à 2 ondelettes. D'autre part, les coefficients de tous les réseaux présentant cette performance sont identiques, ce qui prouve qu'il s'agit bien du même minimum de la fonction de coût. Cette comparaison est également effectuée, pour des réseaux à fonctions dorsales sigmoïdes, dans le paragraphe III.2.2.3.

III.2.2 Réseau prédicteur à fonctions dorsales.

Nous nous intéressons également à la modélisation de ce processus à l'aide de réseaux de fonctions sigmoïdes. D'autres travaux [Rivals95b] ont abordé cette modélisation. Nous la reprenons ici à l'aide de nos outils (notamment avec l'algorithme de Levenberg–Marquardt nécessitant le calcul du gradient dans le sens direct).

III.2.2.1 Apprentissage avec l'algorithme de BFGS.

Dans les mêmes conditions que les réseaux d'ondelettes (mêmes séquences d'apprentissage et d'évaluation de la performance, 50 apprentissages par architecture, et utilisation de l'algorithme de BFGS), nous obtenons les résultats présentés dans le tableau 9.

Nombre de sigmoïdes	Nombre de paramètres	EQMA	EQMP
1	9	0,20	0,3
2	14	0,13	0,17
3	19	0,15	0,14
4	24	0,085	0,16

Tableau 9. Résultats de la modélisation du processus réel avec réseaux de sigmoïdes ; apprentissage à l'aide de l'algorithme de BFGS.

La meilleure performance est donc obtenue avec un réseau de 3 neurones à fonctions sigmoïdes. Augmenter le nombre de neurones améliore l'EQMA mais pas la performance.

III.2.2.2 Apprentissage avec l'algorithme de Levenberg–Marquardt.

Le tableau suivant illustre les meilleures performances obtenues avec des réseaux de fonctions sigmoïdes et des apprentissages utilisant l'algorithme de Levenberg–Marquardt.

Nombre de sigmoïdes	Nombre de paramètres	EQMA	EQMP
1	9	0,23	0,38
2	14	0,11	0,20
3	19	0,092	0,15
4	24	0,086	0,15

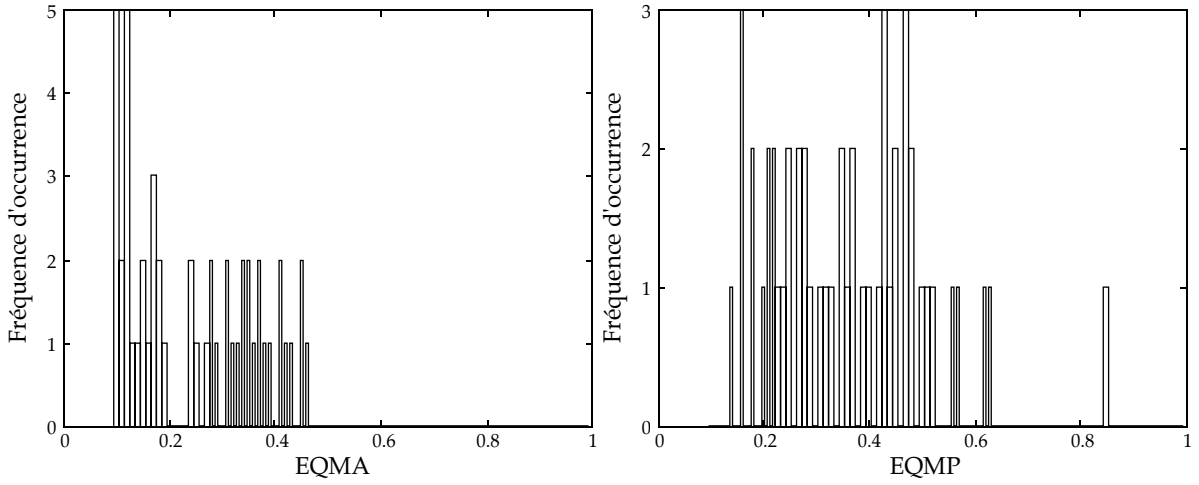
Tableau 10. Résultats de la modélisation du processus réel avec réseaux de sigmoïdes ; apprentissage à l'aide de l'algorithme de Levenberg–Marquardt.

Les architectures à 3 et 4 neurones cachés réalisent une performance égale. En retenant la plus parcimonieuse (réseau à trois neurones), on retrouve donc la même que dans le cas d'un apprentissage avec l'algorithme de BFGS. Les deux algorithmes permettent donc d'aboutir à des modèles de précision équivalente.

III.2.2.3 *Fréquence d'occurrence du meilleur résultat.*

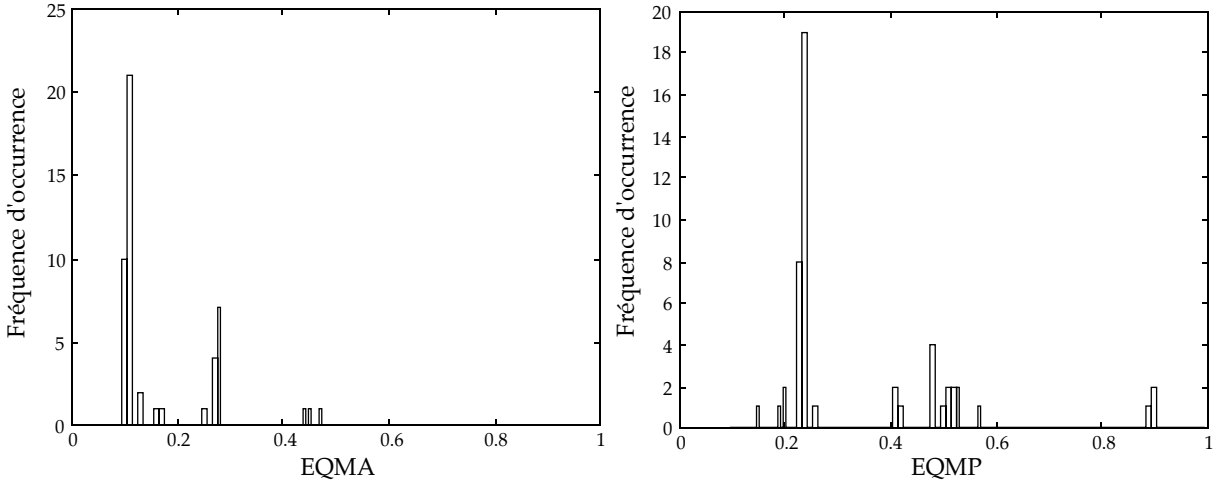
Dans le paragraphe III.2.2.3 de ce chapitre, nous avons comparé "l'efficacité" des deux algorithmes du point de vue de la fréquence d'occurrence de la meilleure solution (que l'on retient) et ceci dans le cas de modèles fondés sur des réseaux d'ondelettes. Nous effectuons à nouveau cette comparaison, avec cette fois les résultats obtenus sur les modèles fondés sur des réseaux de fonctions sigmoïdes.

La figure 8 illustre les histogrammes du nombre d'apparitions des différents critères d'apprentissages (a) et des performances (b) obtenus dans le cas de l'algorithme de BFGS. La figure 9 illustre ces deux histogrammes dans le cas de l'utilisation de l'algorithme de Levenberg–Marquardt.



(a) (b)

Figure 8. Histogrammes des EQMA (a) et des EQMP (b) de réseaux de fonctions dorsales avec apprentissage à l'aide de l'algorithme de BFGS.



(a) (b)

Figure 9. Histogrammes des EQMA (a) et des EQMP (b) de réseaux de fonctions dorsales avec apprentissage à l'aide de l'algorithme de Levenberg–Marquardt.

On fait ici une constatation semblable à celle que nous avons faite pour les réseaux d'ondelettes : une des valeurs de l'EQMP (= 0.24) ainsi que l'EQMA correspondante (= 0.11) sont obtenues beaucoup plus fréquemment que les autres lorsque l'on utilise l'algorithme de Levenberg-Marquardt. Néanmoins, ce n'est pas, cette fois, la valeur minimale de l'EQMP. D'autre part, une comparaison des coefficients des réseaux possédant cette performance montre qu'ils sont dans la plupart des cas identiques.

III.2.3 Conclusion de la modélisation entrée–sortie.

Nous avons effectué une modélisation entrée-sortie d'un processus réel à l'aide de réseaux de neurones à sigmoïdes et d'ondelettes. Nous avons opté pour

l'apprentissage de prédicteurs bouclés afin d'obtenir des modèles de simulation du processus. Cela revient à considérer un modèle hypothèse de type bruit de sortie (Output Error).

Du point de vue des performances obtenues, les deux types de réseaux utilisés dans les mêmes conditions aboutissent généralement à des modèles de précision très proche.

Une comparaison des deux algorithmes que nous utilisons sur l'exemple de la modélisation du processus réel, sur la base de la fréquence d'occurrence de la meilleure solution trouvée a montré que l'algorithme de Levenberg–Marquardt possède un avantage sur la méthode de BFGS. Cette tendance a été observée de façon similaire sur les réseaux d'ondelettes et sur les réseaux de fonctions sigmoïdes. Il n'est évidemment pas possible de généraliser ce résultat à partir de ce seul exemple, qui a pour caractéristique de porter sur un réseau comprenant un très petit nombre d'ondelettes, mais il pourrait être intéressant de mener une comparaison plus systématique des deux algorithmes sous cet angle.

III.3 Modélisation d'état.

Nous nous proposons à présent de modéliser ce processus avec une représentation d'état à variables d'état libres (non mesurées). Nous conservons un ordre 2 ($N_s = 2$) et une mémoire de 1 sur l'entrée de commande ($N_e = 1$).

L'état n'étant pas mesuré, le prédicteur ne peut être que bouclé. Ce prédicteur est optimal dans le cas ou le processus est non bruité, ou si l'on est en présence d'un bruit additif en sortie. Le prédicteur ainsi construit n'est pas un estimateur de l'état : son seul rôle est de modéliser le comportement entrée–sortie du processus.

Nous effectuons l'apprentissage de réseaux réalisant les fonctions suivantes :

$$\begin{cases} x_1(k+1) = \psi_{11}\big(x_1(k), x_2(k), u(k)\big) \\ x_2(k+1) = \psi_{12}\big(x_1(k), x_2(k), u(k)\big) \\ y(k+1) = \psi_{2}\big(x_1(k), x_2(k), u(k)\big) \end{cases} \tag{6}$$

Les fonctions ψ_{11}, ψ_{12} et ψ_2 sont des fonctions réalisées à l'aide d'un réseau d'ondelettes (figure 13 du chapitre III) ou d'un réseau de fonctions dorsales (figure 8 du chapitre II).

Les apprentissages ont été effectués à l'aide de l'algorithme de BFGS. En effet, nous avons vu dans le chapitre III que le calcul du gradient dans le sens direct, nécessaire pour la mise en œuvre de l'algorithme de Levenberg-Marquardt, est coûteux, d'un point de vue numérique, pour des réseaux d'état.

III.3.1 Réseau prédicteur d'état à fonctions d'ondelettes.

Nous effectuons l'apprentissage de plusieurs architectures de réseaux d'ondelettes, dans les même conditions que précédemment.

L'initialisation des réseaux d'ondelettes dans le cas de l'apprentissage d'un prédicteur d'état bouclé présente une difficulté particulière : en effet, les états n'étant pas connus a priori, le domaine des entrées d'état n'est pas connu. Or l'application de la procédure d'initialisation proposée pour les réseaux d'ondelettes fondés sur la transformée continue (chapitre III) nécessite la connaissance de ces domaines.

On peut néanmoins remarquer que, en début d'apprentissage, les valeurs des variables d'état en sortie du réseau sont au maximum égales aux pondérations, lesquelles sont uniformément distribuées dans l'intervalle $[-10^{-2}, 10^{-2}]$. Sous réserve que la sortie soit centrée en zéro, la procédure d'initialisation valable pour les réseaux de type entrée–sortie reste donc applicable pour des réseaux d'état.

Le tableau 11 illustre les résultats obtenus pour des réseaux constitués de 1 et 2 ondelettes.

Nombre d'ondelettes.	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	19	0,38	0,42
2	28	0,091	0,15

Tableau 11. Résultats de la modélisation du processus réel avec réseaux d'état de fonctions ondelettes.

La meilleure performance est donc obtenue avec un réseau de 2 ondelettes (comme pour le modèle de type entrée–sortie). Cette performance est très proche de celle obtenue avec un modèle entrée-sortie (voir tableau 7) mais nous ne sommes pas arrivés à améliorer cette dernière bien que les réseaux d'état constituent une représentation plus générale que les réseaux entrée-sortie. Ce phénomène peut s'expliquer par une taille insuffisante de la séquence d'apprentissage. En effet, dans les résultats obtenus en modélisation entrée-sortie, on remarque que les performances se dégradent souvent pour des réseaux de plus de 25 coefficients ; d'autre part, il faut au moins deux fonctions non linéaires pour modéliser correctement ce processus. Étant donné les réseaux d'état d'ondelettes que nous utilisons, ces deux conditions (un réseau de moins de 25 coefficients et constitué d'au moins deux fonctions) ne peuvent être remplies simultanément.

III.3.2 Réseau prédicteur d'état à fonctions dorsales.

Nous considérons à présent des modèles d'état constitués de réseaux de neurones à une couche de fonctions sigmoïdes et un neurone de sortie linéaire. On peut faire la même remarque concernant le nombre de paramètres. Un réseau d'état à deux fonctions sigmoïdes contient 26 coefficients ajustables. Dans le cas d'un réseau entrée–sortie, ce nombre de paramètres aboutit généralement à un surajustement comme le montrent les résultats présentés plus haut.

Le tableau 12 donne le meilleur résultat obtenu pour des architectures constitués de 1 à 3 fonctions sigmoïdes.

Nombre de sigmoïdes	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	19	0,24	0,34
2	26	0,091	0,18
3	33	0,058	0,18

Tableau 12. Résultats de la modélisation du processus réel avec réseaux d'état de fonctions sigmoïdes.

A partir d'un réseau de 2 neurones, on obtient un critère d'apprentissage meilleur que celui réalisé par un réseau d'entrée–sortie constitué par le même nombre de fonctions. En revanche, la performance n'est pas améliorée. Ceci tend à confirmer l'hypothèse concernant l'existence d'un phénomène de surajustement pour des réseaux de plus de 24 coefficients.

III.3.3 Réseau prédicteur d'état à fonctions dorsales dont la sortie est l'un des états.

On se propose d'utiliser pour la modélisation du processus réel des modèles d'état particuliers dont la sortie est considérée comme un état. Ce type de réseau a été introduit à la fin du chapitre III et le calcul du gradient de la fonction de coût dans le cas de réseaux d'ondelettes est présenté en annexe de ce mémoire.

Dans la suite, nous présentons les résultats obtenus en utilisant de tels réseaux fondés sur des neurones à fonctions dorsales sigmoïdes. Ils réalisent des modèles de la forme :

$$\begin{cases} y(k+1)=x_1(k+1) = \psi_1\big(x_1(k),\, x_2(k), u(k)\big) \\ x_2(k+1) = \psi_2\big(x_1(k),\, x_2(k), u(k)\big) \end{cases} \tag{7}$$

Le tableau 13 illustre les meilleurs résultats sur 50 apprentissages effectués pour chaque réseau de 1 à 3 neurones cachés. L'utilisation des réseaux contenant plus de neurones n'améliore pas la performance.

Nombre de sigmoïdes	Nombre de paramètres.	EQMA	Meilleure EQMP sur 50 apprentissages
1	14	0,2	0,28
2	20	0,12	0,15
3	26	0,071	0,117

Tableau 13. Résultats de la modélisation du processus réel avec réseaux d'état de fonctions sigmoïdes dont la sortie est un état.

Le réseau à 3 neurones présente la meilleure performance que nous ayons obtenue pour la modélisation de ce processus réel, pour tous types de réseaux et de modèles. D'autre part, ce réseau d'état à 3 fonctions sigmoïdes contient 26 paramètres ajustables soit moins qu'un réseau à deux états libres ayant le même nombre de neurones cachés. La figure suivante montre l'architecture de ce réseau :

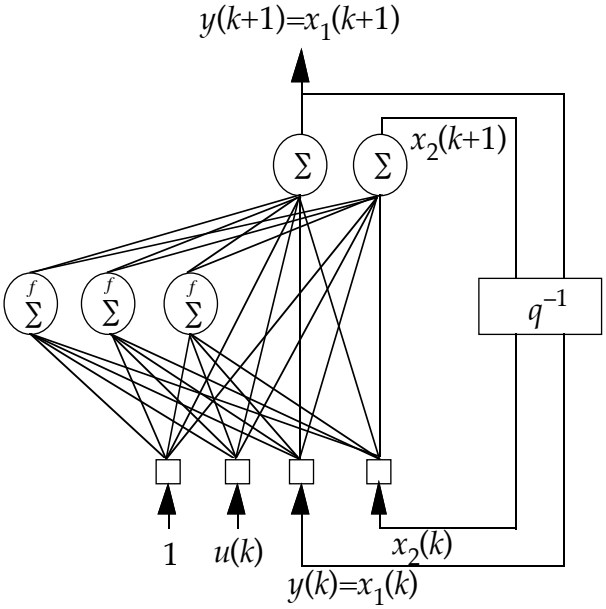


Figure 10. Réseau d'état à 3 fonctions sigmoïdes dont la sortie est un état.

III.3.4 Conclusion de la modélisation d'état.

Du point de vue de la représentation mathématique, les réseaux d'état constituent des modèles plus généraux que ceux de type entrée–sortie.

Nous avons modélisé un processus réel à l'aide de prédicteurs d'état fondés sur des réseaux d'ondelettes et de fonctions dorsales sigmoïdes. Les résultats obtenus montrent que, pour un même nombre de fonctions, un apprentissage avec un réseau d'état présente une meilleure précision que celui avec un réseau entrée–sortie. En revanche, les performances ne sont pas meilleures.

Sur la base des observations effectuées sur les résultats de la modélisation entrée–sortie, il est très probable que ceci est dû à un phénomène de surajustement. En effet, pour un même nombre de fonctions, les réseaux d'état présentent plus de paramètres ajustables que les réseaux entrée–sortie. Les résultats obtenus dans le paragraphe précédent avec un réseau d'état ayant moins de paramètres à nombre de neurones égal (ceci est possible en choisissant la sortie comme un état) sont cohérents avec cette hypothèse.

IV. CONCLUSION.

Nous avons étudié la modélisation de deux processus à l'aide de réseaux d'ondelettes (fondés sur la transformée continue) et de réseaux de neurones bouclés. Les résultats obtenus montrent que les deux types de réseaux, pour un même nombre de fonctions, permettent d'obtenir des modèles de performances très souvent équivalentes.

D'autre part, nous avons effectué une comparaison des deux algorithmes d'apprentissage utilisés dans ce mémoire. Cette comparaison est fondée sur la fréquence d'occurrence d'une solution parmi plusieurs apprentissages effectués pour une même architecture de réseau. Il apparaît que, pour les exemples étudiés dans ce mémoire, l'algorithme de Levenberg-Marquardt possède une meilleure robustesse vis-à-vis de l'initialisation aléatoire des paramètres des réseaux.

La modélisation à l'aide d'un réseau d'état dont la sortie est un état a permis d'améliorer les performances obtenues pour le processus réel avec des réseaux entrée–sortie et d'état. Le résultat obtenu avec un réseau de trois sigmoïdes et égal à celui réalisé dans [Rivals95b] avec un réseau de neurones d'état complètement connecté à deux sigmoïdes.

Conclusion

Le travail dont nous avons rendu compte dans le présent mémoire porte sur l'étude des réseaux d'ondelettes pour la modélisation de processus. Compte tenu des succès rencontrés au cours des dernières années par les réseaux de neurones, il était intéressant, dans la perspective des travaux antérieurs du Laboratoire, d'étudier les possibilités de mise en œuvre des réseaux d'ondelettes, tant pour la modélisation statique que pour la modélisation dynamique, et de comparer leurs performances avec celles des réseaux de neurones classiques utilisant des fonctions dorsales comme les sigmoïdes ou les gaussiennes. Nous avons proposé une procédure simple, pour l'apprentissage des réseaux de fonctions dorsales gaussiennes. Cette procédure, qui agit en cours d'apprentissage, permet une mise en œuvre efficace des ressources dont dispose un réseau, c'est-à-dire de ses neurones cachés.

Nous avons ensuite proposé une méthodologie de mise en œuvre des fonctions ondelettes pour la modélisation statique et dynamique de processus. Nous avons séparé le problème en deux parties, correspondant chacune à un type de transformée en ondelettes. En effet, les paramètres des ondelettes peuvent

- soit prendre n'importe quelle valeur réelle (approche fondée sur la transformée en ondelettes continue),
- soit être choisis sur une grille régulière (approche fondée sur la transformée en ondelettes discrète).

L'approche fondée sur la transformée continue.

Nous avons proposé une méthodologie de mise en œuvre de réseaux d'ondelettes bouclés et non bouclés, dans laquelle on peut considérer tous les paramètres des ondelettes comme des paramètres ajustables : l'apprentissage de ces réseaux peut donc être effectué par minimisation d'une fonction de coût à l'aide de techniques de gradient. Une procédure d'initialisation simple, nécessitant très peu de calculs, permet de prendre en considération la propriété de localité des fonctions.

Les résultats obtenus lors de l'utilisation de ces réseaux, pour la modélisation de quelques processus (simulés et réels) possédant un petit nombre d'entrées, ont montré qu'ils possèdent des propriétés de parcimonie équivalentes à celles des réseaux de neurones, si l'on considère le *nombre de fonctions* utilisées par le réseau pour atteindre la précision recherchée. En revanche, pour le même nombre de fonctions, un réseau d'ondelettes comporte plus de paramètres qu'un réseau de fonction dorsales. De plus, les expériences que nous avons effectuées concernant le problème maître-élève ont montré que la capacité des réseaux d'ondelettes à retrouver le réseau maître est très infé-

rieure à celle des réseaux de neurones dès que la dimension du problème est supérieure à 3 ou 4.

L'approche fondée sur la transformée discrète.
Cette approche permet la construction de réseaux tirant partie des propriétés spécifiques de ces bases de fonctions. Les paramètres de ces fonctions étant à valeurs discrètes, il n'est pas possible d'utiliser des techniques de gradient pour l'apprentissage. La démarche que nous avons adoptée consiste à construire des réseaux par sélection d'ondelettes parmi celles d'une bibliothèque établie à cet effet. Une telle démarche a été utilisée par d'autres auteurs pour des applications de modélisation et de commande, mais elle conduit à des réseaux peu parcimonieux. Nous avons proposé d'utiliser cette démarche pour l'initialisation des apprentissages des réseaux d'ondelettes fondés sur la transformée continue. La modélisation de processus simulés nous a permis de mettre en évidence l'apport de cette procédure d'initialisation.

L'optimisation non linéaire est l'outil fondamental pour l'apprentissage de réseaux de fonctions paramétrées. Afin de permettre l'utilisation d'une famille d'algorithmes plus étendue pour l'apprentissage de réseaux bouclés, entrée-sortie et d'état, nous avons présenté le calcul du gradient dans le sens direct. La mise en œuvre de réseaux d'ondelettes bouclés constitue un des apports originaux de notre travail, qui a fait l'objet d'une publication dans une revue internationale. Nous avons également pu comparer les performances de deux algorithmes du second ordre couramment utilisés pour l'optimisation de la fonction de coût lors de l'apprentissage de réseaux, bouclés ou non : l'algorithme de Levenberg-Marquardt et l'algorithme BFGS.

En résumé, deux conclusions ressortent de cette étude.

- Les réseaux d'ondelettes, bouclés ou non, fondés sur la transformée continue, peuvent constituer une alternative intéressante aux réseaux de neurones conventionnels, à fonction dorsale sigmoïdale, pour constituer des modèles, statiques ou dynamiques, de processus comportant un petit nombre d'entrées. Notre travail sur l'initialisation des coefficients et sur les algorithmes d'apprentissage du second ordre nous a permis de proposer des procédures de mise en œuvre de complexité analogue à celle des réseaux de neurones. En revanche, l'accroissement du nombre de paramètres en fonction du nombre d'entrées est plus rapide que pour des réseaux de sigmoïdes.
- Les réseaux d'ondelettes fondés sur la transformée discrète sont moins parcimonieux que les précédents ; en revanche, la méthode de sélection d'ondelettes

à paramètres discrets peut être mise à profit pour l'initialisation des translations et dilatations de réseaux d'ondelettes fondés sur la transformée continue.

Bibliographie

[Battiti92]
R. Battiti
"First and Second Order Methods for Learning : Between Steepest Descent Methods and Newton's Method."
Neural Computation, Vol. 4, No.2, pp. 141-166, 1992

[Behera95]
L. Behara, M. Gopal & S. Chaudhury
"Inversion of RBF Networks and Applications to Adaptive Controlof Nonlinear Systems."
IEE Proceedings, Control Theory Appl., Vol. 142, No. 6, pp. 617-624, 1995

[Bishop95]
C. Bishop
"Neural Networks for Pattern Recognition."
Clarendon Press – Oxford, New – York, 1995

[Baron97]
R. Baron
"*Contribution à l'Étude des Réseaux d'Ondelettes.*"
Thèse de Doctorat de l'École Normale Supérieure de Lyon, 1997

[Barron93]
A. R. Barron
"*Universal Approximation Bounds for Superpositions of a Sigmoidal Function.*"
IEEE Transactions on Information Theory IT-39, pp. 930-945, 1993

[Broom88]
D. S. Broomhead & D. Lowe
"Multivariable Functional Interpolation and Adaptive Networks."
Complex Systems, Vol. 2, pp. 321-355, 1988

[Cannon95]
M. Cannon & J. J. E. Slotine
"Space-Frequency Localized Basis Function Networks for Nonlinear System Estimation and Control."
Neurocomputing Vol. 9, No. 3, pp. 293-342, 1995

[Caprile90]
B. Caprile & F. Girosi
"A non deterministic minimization algorithm."
A.I. Memo 1254, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1990

[Chen89]
S. Chen, S.A. Billings, W. Luo
"Orthogonal Least Squares Methods and Their Application to Non–linear System Identification."
Int. Journal of Control, Vol. 50, No. 5, pp. 1873-1896, 1989

[Chen90]
S. Chen, A. Billings, C. F. N. Cowan & P. M. Grant
"Practical Identification of NARMAX models using Radial Basis Functions."
Int. Journal of Control, Vol. 52, No. 6, pp. 1327-1350, 1990

[Chentouf96]
R. Chentouf & C. Jutten
"Combining Sigmoids and Radial Basis Functions in Evolutive Neural Architectures."
In Proceedings of the European Symposium on Artificial Neural Networks (ESANN 96), Bruges, Belgium, Avril 1996

[Cohen96]
A. Cohen & J. Kovacheckevic
"Wavelets: The Mathematical Background."
Proceedings of the IEEE, Vol. 84, No. 4, pp. 514-522, 1996

[Cybenko89]
G. Cybenko
"Approximation by Superposition of a Sigmoidal Function."
Mathematics of control, signals and systems, Vol. 2, pp. 303-314, 1989

[Daubechies90]
I. Daubechies
"The Wavelet Transform, Time-Frequency Localization and Signal Analysis."
IEEE Transactions on information theory, Vol. 36, pp. 961-1005, 1990

[Daubechies92]
I. Daubechies
"Ten Lectures on Wavelets."
CBMS-NSF regional series in applied mathematics, SIAM, Philadelphia, 1992

[Dreyfus98]
G. Dreyfus & Y. Idan
"The canonical form of discrete-time non-linear models."
Neural Computation, Vol.10, No. 1, pp. 133-164, 1998

[Elanayar94]
S. Elanayar & Y. C. Shin
"Radial Basis Function Neural Network for Approximation and Estimation of Nonlinear Stochastic Dynamic Systems."
IEEE Transactions On Neural Networks, Vol. 5, No. 4, pp. 594-603, 1994

[Friedman81]
J. H. Friedman & W. Stuetzle
"Projection Pursuit Regression."
Journal of the American Statistical Association. Theory and Methods Section, Vol. 76, No. 376, pp. 817-823, Decembre 1981

[Funahashi89]
K. Funahashi
"On the Approximate Realization of Continuous Mappings by Neural Networks."
Neural Networks, Vol. 2, pp. 183-192, 1989

[Girosi95]
F. Girosi, M. Jones & T. Poggio
"Regularization Theory and Neural Networks Architectures."
Neural Computation, Vol. 7, No. 2, pp.219-269, 1995

[Hartman90]
E. J. Hartman & J. M. Kowalski
"Layered Neural Networks with Gaussian Hidden Units as Universal Approximations."
Neural Computation, Vol. 2, pp.210-215, 1990

[Hassibi93]
B. Hassibi & D. G. Stork
"Second Order Derivatives for Network Pruning: Optimal Brain Surgeon."
In Advances in Neural Information Processing Systems, Vol 5, S.J. Hanson, J.D. Cowan and C.L. Giles, Eds., pp.164-172, Morgan-Kaufmann, Avril 1993

[Hirose91]
Y. Hirose, K. Yamashita & S. Hijiya
"Back-Propagation Algorithm Wich Varies the Number of Hidden Units."
Neural Networks, Vol. 4, No. 1, pp.61-66, 1991

[Hornik89]
K. Hornik, M. Stinchcombe & H. White
"Multilayer feedforward networks are universal approximators."
Neural Networks 2, pp. 359-366, 1989

[Hornik94]
K. Hornik, M. Stinchcombe, H. White & P. Auer
"Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives."
Neural Computation, Vol. 6, No. 6, pp.1262-1275, 1994

[Huber85]
P. J. Huber
"Projection Pursuit."
The Annals of Statistics, Vol. 13, No. 2, pp.435-475, 1985

[Hwang94]
J-N. Hwang, S-R. Lay, M. Maechler, R. douglas Martin & J. Schimert
"Regression Modeling in Back-Propgation and Projection Pursuit Learning."
IEEE Transactions on Neural Networks, Vol. 5, No. 3, pp.342-353, 1994

[Jordan85]
M. I. Jordan,
"*The Learning of Representations for Sequential Performance*."
Thèse de Doctorat, University of California, San Diego, 1985

[Juditsky94]
A. Juditsky, Q. Zhang, B. Delyon, P. Y. Glorennec & A. Benveniste
"Wavelets in Identification: wavelets, splines, neurons, fuzzies: how good for identification?"
Rapport INRIA No. 2315, Septembre 1994

[Jutten95]
C. Jutten & R. Chentouf
"A New Scheme For Incremental Learning."
Neural Processing Letters, Vol. 2, No. 1, 1995

[Kuga95]
T. Kugarajah & Q. Zhang
"Mutidimensional Wavelet Frames."
IEEE Trans. on Neural Networks, Vol. 6, No. 6, pp. 1552-1556, November 1995

[LeCun90]
Y. Le Cun, J. S. Denker & S. A. Solla
"Optimal Brain Damage."
In Proceedings of the Neural Information Processing Systems-2, D. S. Touretzky (ed.), pp. 598-605, Morgan-Kaufmann 1990

[Lehtokangas95]
M. Lehtokangas, J. Saarinen, K. Kaski, P. Huuhtanen
"Initializing Weights of a Multilayer Perceptron Network by Using the Orthogonal Least Squares Algorithm."
Neural Computation, Vol. 7, No. 5, pp. 982-999, 1995

[Levenberg44]
K. Levenberg
"A Method for the Solution of Certain Non–linear Problems in Least Squares."
Quarterly Journal of Applied Mathematics II (2), pp. 164–168, 1944

[Levin92]
A.U. Levin
"Neural Networks in Dynamical Systems."
Thèse de Doctorat, Yale University, New Haven (CT), 1992

[Ljung87]
L. Ljung
"System Identification ; Theory for the User."
Prentice Hall, Englewood Cliffs, New Jersey 1987

[Mallat89]
S. Mallat
"A Theory for Multiresolution Signal Decomposition: The Wavelet Transform."
IEEE Trans. Pattern Anal. Machine Intell. Vol. 11, pp. 674-693, 1989

[Marquardt63]
D. W. Marquardt
"An Algorithm For Least-Squares Estimation of Nonlinear Parameters."
Journal of Soc. Indust. Appl. Math, Vol. 11, No. 2, pp. 431-441, June 1963

[Meyer85]
Y. Meyer
"Principe d'incertitude, bases hilbertiennes et algèbres d'opérateurs."
Séminaire Bourbaki, Numéro 662, 1985–1986

[Meyer90]
Y. Meyer
"Ondelettes et Opérateurs I : Ondelettes."
Editions Hermann, 1990

[Minoux83]
M. Minoux
"Programmation Mathématique : Théorie et Algorrithmes."
Editions Dunod, 1983

[Mohraz96]
K. Mohraz & Peter Protzel
"FlexNet: A Flexible Neural Network Construction Algorithm."
In Proceedings of the European Symposium on Artificial Neural Networks (ESANN 96), Bruges, Belgium 1996

[Mukhopa93]
S. Mukhopadhyay & K. S. Narendra
"Disturbance Rejection in Nonlinear Systems Using Neural Networks."
IEEE Trans. On Neural Networks Vol. 1, pp. 63-72, 1993

[Narendra90]
K. S. Narendra & K. Parthasarathy
"Identification and Control Of Dynamical Systems Using Neural Networks."
IEEE Trans. on Neural Networks Vol.1, pp. 4-27, 1990

[Nash80]
J. C. Nash
"Compact Numerical Methods for Computers : Linear Algebra and Function Minimization."
Adam–Hilger Ltd, Bristol, 1980

[Nerrand92]
O. Nerrand
"Réseaux de Neurones pour le Filtrage Adaptatif, l'Identification et la Commande de Processus."
Thèse de Doctorat de l'Université Paris VI, 1992

[Nerrand93a]
O. Nerrand, P. Roussel-Ragot, L. Personnaz & G. Dreyfus
"Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms."
Neural Computation, Vol. 5, pp 165-199, 1993

[Nerrand94]
O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz & G. Dreyfus
"Training Recurrent Neural Networks : Why and How? An Illustration in Process Modeling."
IEEE Trans. on Neural Networks, Vol. 5, No. 2, pp. 178-184, 1994

[Oussar98]
Y. Oussar, I. Rivals, L. Personnaz & G. Dreyfus
"Training Wavelet Networks for Nonlinear Dynamic Input-Output Modeling."
Neurocomputing, in press.

[Park91]
J. Park & I. W. Sandberg
"Universal Approximation Using Radial-Basis-Function Networks."
Neural Computation Vol. 3, No. 2, pp. 246-257, 1991

[Pati93]
Y. C. Pati & P. S. Krishnaprasad
"Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformations."
IEEE Trans. on Neural Networks Vol. 4, No. 1, pp. 73-85, 1993

[Powell85]
M. J. D. Powell
"Radial Basis Functions for Multi–variable Interpolation : A Review."
IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS Shrivenham, UK, 1985

[Pucar95]
P. Pucar & M. Millnert
"Smooth Hinging Hyperplanes - An Alternative to Neural Nets."
Proceedings of 3rd European Control Conference, Vol. 2, pp. 1173-1178, Italy, September 1995

[Reed93]
R. Reed
"Pruning Algorithms - A Survey."
IEEE Transactions on Neural Networks, Vol. 4, No. 5, pp. 740-747, 1993

[Rivals95a]
I. Rivals
"Modélisation et Commande de Processus par Réseaux de Neurones; Application au Pilotage d'un Véhicule Autonome."
Thèse de Doctorat de l'Université Paris 6 ,1995

[Rivals95b]
I. Rivals, L. Personnaz, G. Dreyfus & J.L. Ploix
"Modélisation, classification et commande par réseaux de neurones : principes fondamentaux, méthodologie de conception et illustrations industrielles."
Dans : *Les réseaux de neurones pour la modélisation et la commande de procédés*, J.P. Corriou, coordonnateur (Lavoisier Tec et Doc), 1995

[Rivals96]
I. Rivals & L. Personnaz
"Black Box Modeling With State Neural Networks."
In Neural Adaptive Control Technology I, R. Zbikowski and K. J. Hunt eds., World Scientific, 1995

[Rumelhart86]
D. E. Rumelhart, and J. L. McClelland
"*Parallel Distributed Processing*,."
MIT Press, Cambridge, MA, 1986

[Sanner92]
R. Sanner & J. J. E Slotine
"Gaussian Networks for Direct Adaptive Control."
IEEE Transactions on Neural Networks, Vol. 3, No. 6, pp. 837-863, 1992

[Sanner95]
R. Sanner & J. J. E Slotine
"Stable Adaptive Control of Robot Manipulators Using Neural Networks."
Neural Computation, Vol. 7, No. 4, pp. 753-790, 1995

[Söberg95]
J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, Et Al
"Nonlinear Black–Box Modeling in System Identification: a Unified Overview."
Automatica, Vol. 31, No. 12, pp. 1691-1724, 1995

[Sontag93]
"*Neural Networks for Control.*"
In "*Essays on Control: perspectives in the theory and its applications*."
H. L. Trentelman & J. C. Willems Editions, Birkhäuser, Boston 1993

[Stoppi97]
H. Stoppiglia
"Méthodes statistiques de sélection de modèles neuronaux; applications financières et bancaires."
Thèse de Doctorat de l'Université Paris 6, 1997

[Torré95]
B. Torrésani
"Analyse Continue par Ondelettes."
InterEditions / CNRS Editions, Paris 1995

[Urbani95]
D. Urbani
"Méthodes Statistiques de Sélection d'Architectures Neuronales: Application à la Conception de Modèles de Processus Dynamiques."
Thèse de Doctorat de l'Université Paris 6, 1995

[Walter94]
E. Walter & L. Pronzato
"Identification de modèles paramétriques à partir de données expérimentales."
Editions Masson, Paris 1994

[Yang96]
S. Yang & C. Tseng
"An Orthogonal Neural Network for Function Approximation."
IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics. Vol. 26, No. 5, pp. 779-785, 1996

[Zhang92]
Q. Zhang & A. Benveniste
"Wavelet Networks."
IEEE Trans. on Neural Networks Vol. 3, No. 6, pp. 889-898, 1992

[Zhang93]
Q. Zhang
"Regression Selection and Wavelet Network Construction."
Rapport interne de l'INRIA N. 709, Projet AS, Avril 1993

[Zhang95]
J. Zhang, G. G. Walter, Y. Miao & W. N. Wayne Lee
"Wavelet Neural Networks For Function Learning."
IEEE Trans. on Signal Processing, Vol. 43, no. 6, pp. 1485-1497, 1995

[Zhang97]
Q. Zhang
"Using Wavelet Network in Nonparametric Estimation."
IEEE Trans. on Neural Networks, Vol. 8, No. 2, pp. 227-236, 1997

Annexe A

Training Wavelet Networks for Nonlinear Dynamic Input-Output Modeling

Article accepté pour publication dans Neurocomputing

Neurocomputing, in press.

Training Wavelet Networks for Nonlinear Dynamic Input-Output Modeling.

Y. Oussar, I. Rivals, L. Personnaz, G. Dreyfus
Laboratoire d'Électronique
École Supérieure de Physique et Chimie Industrielles
10, rue Vauquelin
F - 75231 PARIS Cedex 05, FRANCE.
Phone: 33 1 40 79 45 41 Fax: 33 1 40 79 44 25
E-mail: Yacine.Oussar@espci.fr

Abstract

In the framework of nonlinear process modeling, we propose training algorithms for feedback wavelet networks used as nonlinear dynamic models. An original initialization procedure is presented, that takes the locality of the wavelet functions into account. Results obtained for the modeling of several processes are presented; a comparison with networks of neurons with sigmoidal functions is performed.

Keywords: Training, Wavelet networks, Nonlinear dynamic modeling, Neural networks, Feedback networks, Recurrent networks.

1. INTRODUCTION.

During the past few years, the nonlinear dynamic modeling of processes by neural networks has been extensively studied. Both input-output [7] [8] and state-space [5] [14] models were investigated. In standard neural networks, the non-linearities are approximated by superposition of sigmoidal functions. These networks are universal approximators [2] and have been shown to be parsimonious [3].

Wavelets are alternative universal approximators; wavelet networks have been investigated in [17] in the framework of *static* modeling; in the present paper, we propose a training algorithm for feedback wavelet networks used as nonlinear *dynamic* models of processes. We first present the wavelets that we use and their properties. In section 3, feedforward wavelet networks for static modeling are presented. In section 4, the training systems and algorithms for dynamic input-output modeling with wavelet networks, making use of the results of section 3, are described. For illustration purposes, the modeling of several processes by wavelet networks and by neural networks with sigmoidal functions is presented in section 5.

2. FROM ORTHOGONAL WAVELET DECOMPOSITION TO WAVELET NETWORKS.

The theory of wavelets was first proposed in the field of multiresolution analysis; among others, it has been applied to image and signal processing [6]. A family of wavelets is constructed by translations and dilations performed on a single fixed function called the *mother wavelet*. A wavelet ϕ_j is derived from its mother wavelet ϕ by

$$\phi_j(z) = \phi\left(\frac{x - m_j}{d_j}\right) \qquad (1)$$

where its translation factor m_j and its dilation factor d_j are real numbers ($d_j > 0$). We are concerned with modeling problems, i.e. with the fitting of a data set by a finite sum of wavelets. There are several ways to determine the wavelets for this purpose:

- From orthogonal wavelet decomposition theory, it is known that, with a suitable choice of ϕ, and if m_j and d_j are integers satisfying some conditions, the family $\{\phi_j\}$ forms an orthogonal wavelet basis. A weighted sum of such functions with appropriately chosen m_j and d_j can thus be used; in this way, only the weights have to be computed [18].
- Another way to design a wavelet network is to determine the m_j and d_j according to a space-frequency analysis of the data; this leads to a set of wavelets which are not necessarily orthogonal [10] [1].
- Alternatively, one can consider a weighted sum of wavelets functions whose parameters m_j and d_j are adjustable real numbers, which are to be trained together with the weights.

In the latter approach, wavelets are considered as a family of parameterized nonlinear functions which can be used for nonlinear regression; their parameters are estimated through "training". The present paper introduces training algorithms for *feedback* wavelet networks used for dynamic modeling, which are similar in spirit to training algorithms used for feedback neural networks.

Choice of a mother wavelet

In the present paper, we choose the first derivative of a gaussian function, $\phi(x) = \pm x \exp\left(\pm \frac{1}{2} x^2\right)$ as a mother wavelet. It may be regarded as a differentiable version of the Haar mother wavelet, just as the sigmoid is a differentiable version of a step function, and it has the universal approximation property [17]. This mother wavelet has also been used in reference [17]. More complex wavelet functions, such as the second derivative of the gaussian (as in [1]) may be used, but they will not be considered here.

The wavelet network.

In the case of a problem with N_i inputs, multidimensional wavelets must be considered. The simplest, most frequent choice ([1], [6], [17], [18]) is that of separable wavelets, i.e. the product of N_i monodimensional wavelets of each input:

$$\Phi_j(x) = \prod_{k=1}^{N_i} \phi(z_{jk}) \quad \text{with} \quad z_{jk} = \frac{x_k - m_{jk}}{d_{jk}} \qquad (2)$$

where m_j and d_j are the translation and dilation vectors. We consider wavelet networks of the form:

$$y = \psi(x) = \sum_{j=1}^{N_w} c_j \Phi_j(x) + a_0 + \sum_{k=1}^{N_i} a_k x_k \ . \qquad (3)$$

(3) can be viewed as a network with N_i inputs, a layer of N_w wavelets of dimension N_i, a bias term, and a linear output neuron. When linear terms are expected to play an important role in the model, it is customary to have additional direct connections from inputs to outputs, since there is no point in using wavelets for reconstructing linear terms. Such a network is shown in Figure 1.

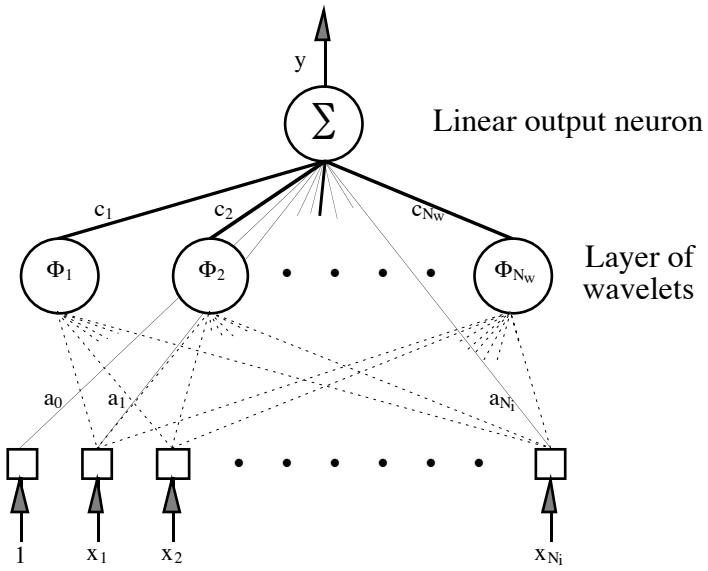


Figure 1. A feedforward wavelet network.

3. STATIC MODELING USING FEEDFORWARD WAVELET NETWORKS.

Static modeling with wavelet networks has been investigated by other authors in [17]. In order to make the paper self-contained, we devote the present section to introducing notations and to recalling basic equations which will be used in Section 4 for dynamic modeling.

We consider a process with N_i inputs and a scalar output y_p. Steady-state measurements of the inputs and outputs of the process build up a training set of N examples $(\boldsymbol{x}^n, y_p^n)$, $\boldsymbol{x}^n = [x_1^n, \ldots, x_{N_i}^n]^T$ being the input vector for example n and y_p^n the corresponding measured process output. In the domain defined by the training set, the static behavior of the process is assumed to be described by:

$$y_p^n = f(\boldsymbol{x}^n) + w^n \qquad n = 1 \text{ to } N \qquad (4)$$

where f is an unknown nonlinear function, and $\{w^n\}$ denotes a set of independent identically distributed random variables with zero mean and variance σ_w^2.

We associate the following wavelet network to the assumed model (4):

$$y^n = \psi(\boldsymbol{x}^n, \theta) \qquad n = 1 \text{ to } N \qquad (5)$$

where y^n is the model output value related to example n, the nonlinear function ψ is given by

relation (3), and θ is the set of adjustable parameters:

$$\theta = \{m_{jk},\, d_{jk},\, c_j,\, a_k,\, a_0\} \quad \text{with} \quad j = 1\ ,\ \ldots\ ,\ N_w \quad \text{and} \quad k = 1\ ,\ \ldots\ ,\ N_i \qquad (6)$$

θ is to be estimated by training so that ψ approximates the unknown function f on the domain defined by the training set.

3.1. Training feedforward wavelet networks.

As usual, the training is based on the minimization of the following quadratic cost function:

$$J(\theta) = \frac{1}{2} \sum_{n=1}^{N} (y_p^n - y^n)^2 = \frac{1}{2} \sum_{n=1}^{N} (e^n)^2 \qquad (7)$$

The minimization is performed by iterative gradient-based methods.

The partial derivative of the cost function with respect to θ is:

$$\frac{\partial J}{\partial \theta} = - \sum_{n=1}^{N} e^n \frac{\partial y^n}{\partial \theta} \qquad (8)$$

where $\frac{\partial y^n}{\partial \theta}$ is a short notation for $\left.\frac{\partial y}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{x}=\boldsymbol{x}^n}$. The components of the latter vector are:

- parameter a_0:

$$\frac{\partial y^n}{\partial a_0} = 1 \qquad (9)$$

- direct connection parameters:

$$\frac{\partial y^n}{\partial a_k} = x_k^n \quad k = 1\ ,\ \ldots,\ N_i \qquad (10)$$

- weights:

$$\frac{\partial y^n}{\partial c_j} = \boldsymbol{\Phi}_j(\boldsymbol{x}^n) \quad j = 1\ ,\ \ldots\ ,\ N_w \qquad (11)$$

- translations:

$$\frac{\partial y^n}{\partial m_{jk}} = \pm \frac{c_j}{d_{jk}} \left.\frac{\partial \boldsymbol{\Phi}_j}{\partial z_{jk}}\right|_{\boldsymbol{x}=\boldsymbol{x}^n} \quad k = 1\ ,\ \ldots\ ,\ N_i \text{ and } j = 1\ ,\ \ldots\ ,\ N_w \qquad (12)$$

with

$$\left.\frac{\partial \boldsymbol{\Phi}_j}{\partial z_{jk}}\right|_{\boldsymbol{x}=\boldsymbol{x}^n} = \phi(z_{j1}^n)\ \phi(z_{j2}^n) \ldots \phi'(z_{jk}^n) \ldots \phi(z_{jN_i}^n) \qquad (13)$$

where $\phi'(z_{jk}^n)$ is the value of the derivative of the scalar mother wavelet at point z_{jk}^n:

$$\phi'(z_{jk}^n) = \left.\frac{d\phi(z)}{dz}\right|_{z=z_{jk}^n} \qquad (14)$$

- dilations:

$$\frac{\partial y^n}{\partial d_{jk}} = \pm \frac{c_j}{d_{jk}} z_{jk}^n \left.\frac{\partial \boldsymbol{\Phi}_j}{\partial z_{jk}}\right|_{\boldsymbol{x}=\boldsymbol{x}^n} \quad k = 1\ ,\ \ldots\ ,\ N_i \text{ and } j = 1\ ,\ \ldots\ ,\ N_w \qquad (15)$$

At each iteration, the parameters are modified using the gradient (8), according to:

$$\Delta\theta = -M \frac{\partial J}{\partial \theta} \qquad (16)$$

where M is some definite positive matrix ($M = \mu\, Id$, $\mu>0$ in the case of a simple gradient descent, or $M = \mu\, \widetilde{H^{-1}}$, $\mu>0$ where $\widetilde{H^{-1}}$ is an approximation, updated iteratively, of the inverse Hessian, for quasi-Newton methods).

3.2. Initialization of the network parameters.

Initializing the wavelet network parameters is an important issue. Similarly to Radial Basis Function networks (and in contrast to neural networks using sigmoidal functions), a random initialization of all the parameters to small values (as usually done with neural networks) is not desirable since this may make some wavelets too local (small dilations) and make the components of the gradient of the cost function very small in areas of interest. In general, one wants to take advantage of the input space domains where the wavelets are not zero.

Therefore, we propose an initialization for the mother wavelet $\phi(x) = \pm x \exp\left(\pm \frac{1}{2} x^2\right)$ based on the input domains defined by the examples of the training sequence. We denote by $[\alpha_k, \beta_k]$ the domain containing the values of the k-th component of the input vectors of the examples. We initialize the vector m of wavelet j at the center of the parallelepiped defined by the N_i intervals $\{[\alpha_k, \beta_k]\}$: $m_{jk} = \frac{1}{2}(\alpha_k + \beta_k)$. The dilation parameters are initialized to the value $0.2\,(\beta_k - \alpha_k)$ in order to guarantee that the wavelets extend initially over the whole input domain. The choice of the a_k ($k = 1, \ldots, N_i$) and c_j ($j = 1, \ldots, N_w$) is less critical: these parameters are initialized to small random values.

3.3. Stopping conditions for training.

The algorithm is stopped when one of several conditions is satisfied: the Euclidean norm of the gradient, or of the variation of the gradient, or of the variation of the parameters, reaches a lower bound, or the number of iterations reaches a fixed maximum, whichever is satisfied first.

The final performance of the wavelet network model depends on whether: (i) the assumptions made about the model (relation 4) are appropriate, (ii) the training set is large enough, (iii) the family contains a function which is an approximation of f with the desired accuracy in the domain defined by the training set, (iv) an efficient (i.e. second-order) training algorithm is used.

4. DYNAMIC MODELING USING WAVELET NETWORKS.

We propose to extend the use of wavelet networks to the dynamic modeling of single-input-single-output (SISO) processes. The training set consists of two sequences of length N: the input sequence $\{u(n)\}$ and the measured process output $\{y_p(n)\}$. As in the static case, the aim is to approximate f by a wavelet network.

Depending on the assumptions about the noise, either feedforward or feedback predictors may

be required [9]. For example, if it is assumed that the noise acting on the process is state noise (see for instance equation (35) of section 5.2), i.e. if a Nonlinear AutoRegressive with eXogeneous inputs (NARX, or Equation Error) model

$$y_p(n) = f\left(y_p(n \pm 1)), y_p(n \pm 2), ..., y_p(n \pm N_s), u(n \pm 1), ..., u(n \pm N_e)\right) + w_n \qquad (17)$$

is assumed to be valid, then *the optimal associated predictor is a feedforward one,* whose inputs are past outputs *of the process* y_p and the external inputs u:

$$y(n) = f\left(y_p(n\text{-}1), y_p(n\text{-}2) , ... , y_p(n\text{-}N_s), u(n\text{-}1) , ... , u(n\text{-}N_e)\right) \ . \qquad (18)$$

f is a unknown nonlinear function, which is to be approximated by a wavelet network ψ given by (3).

Conversely, if it is assumed that the noise is output noise, i.e. if an Output Error model

$$\begin{aligned} s(n) &= f\left(s(n \pm 1)), s(n \pm 2), ..., s(n \pm N_s), u(n \pm 1), ..., u(n \pm N_e)\right) \\ y_p(n) &= s(n) + w(n) \end{aligned} \qquad (19)$$

is assumed to be valid, then *the optimal associated predictor is a feedback one,* whose inputs are past outputs *of the model* y and the external inputs u:

$$y(n) = f\left(y(n\text{-}1), y(n\text{-}2) , ... , y(n\text{-}N_s), u(n\text{-}1) , ... , u(n\text{-}N_e)\right) \qquad (20)$$

In the absence of noise, either feedforward or feedback predictors can be used. If the goal is the design of a simulation model, i.e. of a model that can compute the output more than one time step ahead, a feedback predictor should be trained [9].

In all cases, θ is to be estimated so that ψ approximates the unknown function f on the domain defined by the training set.

We define the copy n $(n = 1,..., N)$ as the wavelet network configuration giving $y(n)$ at its output in the case of a feedforward predictor, and as the feedforward part of the network canonical form in the case of a feedback predictor [8]. In order to keep the notations equivalent with the previous section we note: $y^n = y(n)$.

4.1. Training feedforward wavelet predictors.

In this case, the N copies are independent, and the training is similar to that of a static model. Therefore, the input vector of copy n can be viewed as the vector $\boldsymbol{x}^n$ defined in section 3 and $\{y_p(n)\}$ as the process output defined as y_p^n. More precisely, the inputs of copy n can be renamed as:

- external inputs: $x_k^n = u(n\text{-}k)$ with $k = 1, ... , N_e$
- state inputs: $x_k^n = y_p(n\text{-}k\text{+}N_e)$ with $k = N_e\text{+}1 , ... , N_e\text{+}N_s$

Since the state inputs of the copies are forced to the corresponding desired values, the predictor is said to be trained in a *directed* [8], or *teacher-forced* [4] fashion.

4.2. Training feedback wavelet predictors.

In this case, the N copies are not independent: the N output values $y^n = y(n)$ of the network may be considered as being computed by a large feedforward network made of N cascaded copies of the feedforward part of the canonical form of the feedback network [8]: the state

inputs of copy n are equal to the state outputs of copy n-1. The inputs and outputs of copy n are renamed as:

- external inputs: $x_k^n = u(n-k)$ with $k = 1, \ldots, N_e$.
- state inputs: $x_k^n = y(n-k+N_e)$ with $k = N_e+1, \ldots, N_e+N_s$.
- state outputs: $x_k^n = y(n-k+N_e+N_s+1)$ with $k = N_e+N_s+1, \ldots, N_e+2N_s$.

$x_{N_e+N_s+1}^n = y(n) = y^n$ is the n-th value of the output of the network.

$\theta^n = \{m_{jk}^n, d_{jk}^n, c_j^n, a_k^n, a_0^n\}$ with $j = 1, \ldots, N_w$ and $k = 1, \ldots, N_e+N_s$

is the set of parameters of copy n. The feedback predictor network and copy n are shown on figure 2.

Since the state inputs of the first copy only are forced to desired values, the predictor is said to be trained in a *semi-directed* fashion [8], (also known as *backpropagation through time* [15]: the gradient of the cost function is computed by a single backpropagation through the N copies).

The gradient of $J(\theta) = \frac{1}{2}\sum_{n=1}^{N}(y_p^n - y^n)^2 = \frac{1}{2}\sum_{n=1}^{N}(e^n)^2$ with respect to θ can be expressed as the sum of the gradient with respect to each of the N copies θ^n of θ:

$$\frac{\partial J}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial J}{\partial \theta^n} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n}\frac{\partial y^n}{\partial \theta^n} \tag{21}$$

The analytical expressions of $\frac{\partial y^n}{\partial m_{jk}^n}, \frac{\partial y^n}{\partial d_{jk}^n}, \frac{\partial y^n}{\partial c_j^n}, \frac{\partial y^n}{\partial a_k^n}, \frac{\partial y^n}{\partial a_0^n}$ which are the components of $\frac{\partial y^n}{\partial \theta^n}$ are identical to those given (without superscript n for θ) in relations (9) – (15), for the training of feedforward nets.

The set of partial derivatives $\left\{\frac{\partial J}{\partial y^n}\right\}$ can be computed by backpropagation through the feedforward network consisting of the N cascaded copies.

We introduce the intermediate variables $\{q_k^n\}$, q_k^n being the partial derivative of -J with respect to x_k^n, the state variable x_k of the n-th copy:

$$q_k^n = -\frac{\partial J}{\partial x_k^n} \tag{22}$$

Copy N:

- output:

$$q_{\text{out}}^N = q_{N_e+N_s+1}^N = e^N \tag{23}$$

- other output state variables:

$$q_k^N = 0 \quad \textit{with} \quad k = N_e+N_s+2, \ldots, N_e+2N_s \tag{24}$$

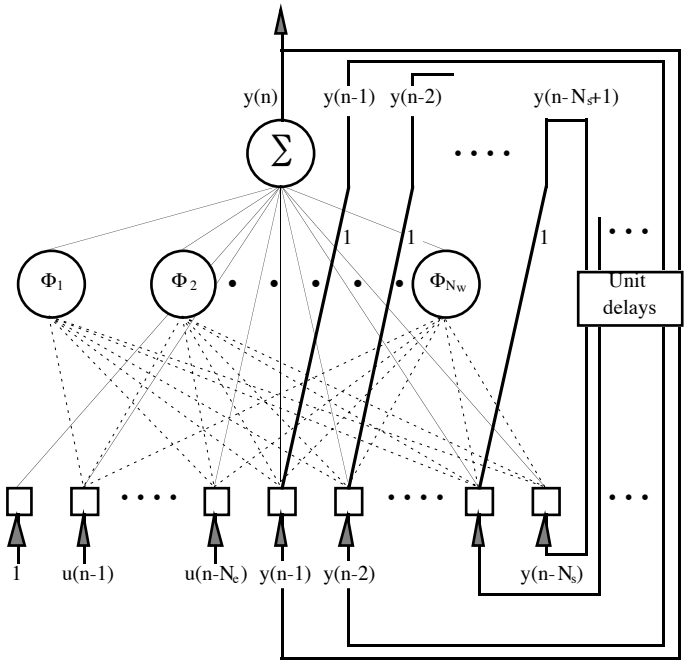
- for the N_s state inputs :

$$q_k^N = \left(a_k^N + \sum_{j=1}^{N_w} \frac{c_j^N}{d_{jk}^N}\frac{\partial \Phi_j}{\partial z_{jk}^N}\right) q_{\text{out}}^N \quad \text{with} \quad k = N_e+1, \ldots, N_e+N_s \tag{25}$$

Copies n = N-1 to 2:

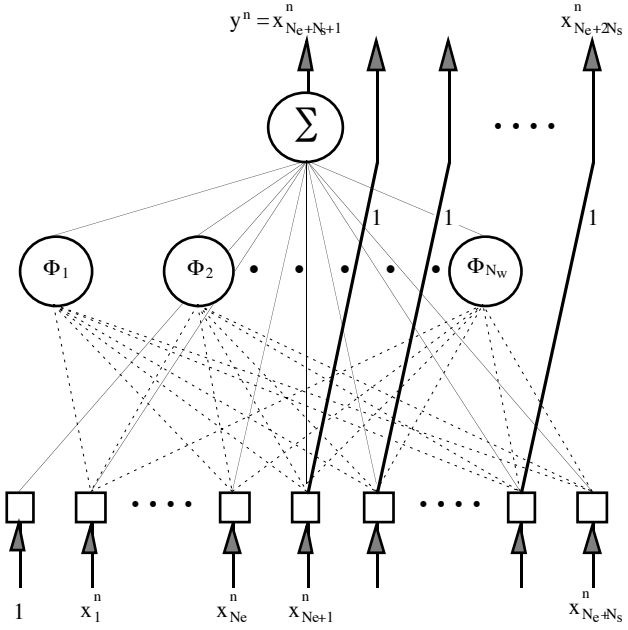
- output:

$$q_{\text{out}}^n = e^n + q_{N_e+1}^{n+1} \tag{26}$$



(a)

N_s output state variables



N_e external inputs N_s input state variables

(b)

Figure 2. (a) feedback predictor network; (b) n-th copy for training.

- other output state variables:

$$q_k^n = q_{k-N_s}^{n+1} \quad \text{with} \quad k = N_e+N_s+2 \, , \, \ldots \, , \, N_e+2\,N_s \tag{27}$$

- the N_s-1 first state inputs :

$$q_k^n = q_{k+N_s+1}^n + \left(a_k^n + \sum_{j=1}^{N_w} \frac{c_j^n}{d_{jk}^n} \frac{\partial \Phi_j}{\partial z_{jk}^n} \right) q_{\text{out}}^n \text{ with } k = N_e+1 , \ldots , N_e+N_s-1 \qquad (28)$$

- the last state input :

$$q_{N_e+N_s}^n = \left(a_{N_e+N_s}^n + \sum_{j=1}^{N_w} \frac{c_j^n}{d_{jk}^n} \frac{\partial \Phi_j}{\partial z_{jk}^n} \right) q_{out}^n \qquad (29)$$

Copy 1:

- output:

$$q_{\text{out}}^1 = e^1 + q_{N_e+1}^2 \qquad (30)$$

5. SIMULATION RESULTS.

In this section we make use of the above algorithms for training input-output wavelet networks on data gathered from simulated and from real processes, and we make use of the algorithms presented in [8] for training input-output neural networks with one hidden layer of sigmoidal neurons on the same data.

The wavelet networks are input-output models as defined by (18) or (20), where the unknown function *f* is approximated by wavelet networks whose mother wavelet is described in section 2 (derivative of a gaussian).

The neural networks used have one hidden layer of sigmoidal units and direct connections from the inputs:

$$y(\boldsymbol{x}) = \sum_{j=1}^{N_\sigma} c_j \tanh\left(v_j(\boldsymbol{x})\right) + a_0 + \sum_{k=1}^{N_i} a_k x_k \quad \text{with} \quad v_j(\boldsymbol{x}) = \sum_{k=1}^{N_i} w_{jk} x_k \qquad (31)$$

We denote by Training Mean Square Error (TMSE) the mean square error on the training set:

$$\text{TMSE} = \frac{1}{N} \sum_{n=1}^{N} \left(y_p(n) - y^n\right)^2 = \frac{2}{N} J \qquad (32)$$

The performance of the model is estimated by the Performance Mean Square Error (PMSE), computed on a test sequence.

The training procedure starts with a simple gradient method (500 iterations) which is followed by a quasi-Newton method (BFGS with line search by Nash [11]).

5.1. Modeling of a simulated process without noise.

The process considered here is simulated with a second order nonlinear equation. This process has been used to illustrate a selection procedure for neural models [16]. The output of the process is given by:

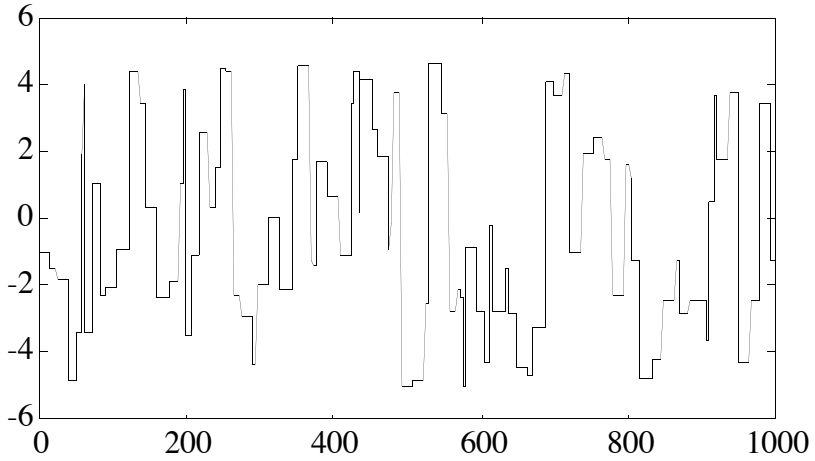
$$y_p(n) = f\left(y_p(n\text{-}1), y_p(n\text{-}2), u(n\text{-}1)\right) = \frac{24 + y_p(n\text{-}1)}{30} y_p(n\text{-}1) - 0.8 \frac{u(n\text{-}1)^2}{1 + u(n\text{-}1)^2} y_p(n\text{-}2) + 0.5u(n\text{-}1) \qquad (33)$$

Since noise is absent, either feedforward or feedback predictors can be used. In order to obtain a simulation model of the process, we chose to train a feedback predictor:

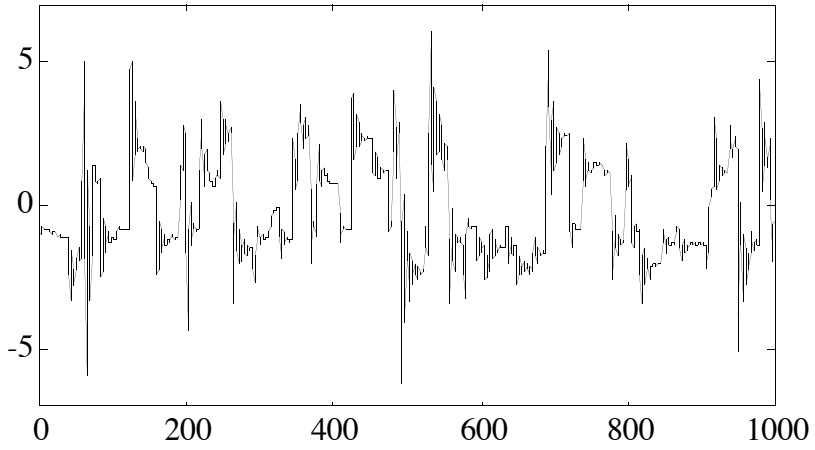
$$y(n) = \psi\left(y(n\text{-}1), y(n\text{-}2), u(n\text{-}1), \theta\right) \qquad (34)$$

A training and a test sequence of 1000 samples each were generated. The input sequence for

both training and test consists of pulses with random amplitude in the range [-5,5] and with random duration between 1 and 20 sampling periods. Figures 3a and 3b show the training sequence.



(a)



(b)

Figure 3: (a) Training input sequence; (b) Training output sequence.

Several feedback wavelet networks were trained, with fifty different initializations for each network. The results corresponding to the minimal PMSE's are given in table 1. Additional wavelets do not improve the performance.

Number of wavelets	Number of parameters	TMSE	PMSE
1	11	$7.6\ 10^{-2}$	$1.5\ 10^{-1}$
2	18	$2.0\ 10^{-2}$	$3.6\ 10^{-2}$
3	25	$2.2\ 10^{-3}$	$6.7\ 10^{-3}$
4	32	$2.8\ 10^{-4}$	$1.3\ 10^{-3}$
5	39	$5.2\ 10^{-5}$	$2.9\ 10^{-4}$
6	46	$3.8\ 10^{-6}$	$2.9\ 10^{-5}$

Table 1. Wavelet modeling results for the noiseless simulated process.

Several feedback neural networks were trained, with fifty different initializations for each network. The results corresponding to the minimal PMSE's are given in table 2. Additional hidden neurons do not improve the performance.

Number of sigmoids	Number of parameters	TMSE	PMSE
1	9	$1.1\ 10^{-1}$	$1.8\ 10^{-1}$
2	14	$7.1\ 10^{-2}$	$1.0\ 10^{-1}$
3	19	$1.1\ 10^{-3}$	$8.4\ 10^{-3}$
4	24	$3.9\ 10^{-4}$	$2.3\ 10^{-3}$
5	29	$4.5\ 10^{-6}$	$1.8\ 10^{-5}$
6	34	$4.2\ 10^{-6}$	$1.6\ 10^{-5}$

Table 2. Neural modeling results for the noiseless simulated process.

In this example, the two types of networks perform with roughly the same accuracy.

5.2. Modeling of a simulated process with noise.

The previous trainings were performed with noiseless data. In this section, we study the case where a zero-mean noise acts on the process. As described in section 4, we consider two cases: NARX models and Output Error models.

In the first one, the state variables of the model used for simulating the process are the output of the process at times n and n-1, and the noise is added to the state variables. It is a NARX model given by the following equation:

$$y_p(n) = f\left(y_p(n\text{-}1),\ y_p(n\text{-}2),\ u(n\text{-}1)\right) + w(n) \tag{35}$$

where f is the function introduced in the previous section.

In the second case, the state variables of the model used for simulating the process are not subject to noise, but noise is added to the output variable: it is an Output Error model given by the following equations:

$$\begin{cases} s(n) = f\left(s(n\text{-}1),\ s(n\text{-}2),\ u(n\text{-}1)\right) \\ y_p(n) = s(n) + w(n) \end{cases} \tag{36}$$

where $s(n)$ and $s(n\text{-}1)$ are the state variables.

Since we are interested in black-box modeling, we generate training and test data from (35) or (36). The input sequences used are identical to those shown in the previous section. The processes are simulated with a noise of variance $\sigma_w^2 = 10^{-2}$. Once the training and test sequences are generated, we pretend not to know equations (35) and (36). Since we must make a decision as to whether we train a feedforward predictor or a feedback predictor, we have to make an assumption about the effect of noise on the process (output noise or state noise). The results presented below have been obtained by making the right assumption: for modeling the data generated by equation (35), we have trained a feedforward wavelet predictor, and, for modeling

the data generated by equation (36), we have used a feedback predictor (the adverse effect of making the wrong assumption about the noise has been demonstrated in [8]).

Since we are modeling a process with noise, the goal is the following: find the smallest network such that the error on the test set and the error on the training be as close as possible to the variance of the noise. Because the process is simulated, we know the variance of the noise, so that we know whether this goal is achieved.
As in the case of the process without noise, several networks with an increasing number of wavelets were trained. The optimal N_w, for which the PMSE is smallest (no overfitting occurs), is 5; the results presented on table 3 show that the variance of the noise is indeed reached.

	TMSE	PMSE
NARX Model	$9.6\ 10^{-3}$	$1.0\ 10^{-2}$
Output Error Model	$1.0\ 10^{-2}$	$1.2\ 10^{-2}$

Table 3. Wavelet modeling results for noisy simulated processes, when the right assumption about the effect of noise is made.

5.3. Modeling of a real process.

The process to be modeled is the hydraulic actuator of a robot arm. The external input u is the position of a valve and the output y_p is the oil pressure. A sequence of 1024 points is available. We consider the first half of the data sequence as a training sequence. We use a feedback predictor with N_e=1 and N_s=2 so that:

$$y(n) = \psi(y(n\text{-}1), y(n\text{-}2), u(n\text{-}1), \theta) \qquad (37)$$

Predictors having increasing numbers of wavelets were trained, with 50 initializations for each predictor. The best PMSE is obtained with a network of 2 wavelets (18 parameters); the corresponding values of the TMSE and PMSE are:

$$\text{TMSE} = 0.11 \quad \text{PMSE} = 0.13$$

Figure 4 shows the responses of the process and of the wavelet network on the test sequence.
Table 4 shows the results obtained on the same problem with other input-output models. The neural network model whose performance is reported has three hidden neurons (best PMSE of 50 trainings with different initializations).

Input-output model	PMSE	Numbers of parameters	Reference
Hinging hyperplanes	0.34	14	[12]
Neural Network	0.14	19	This paper
Wavelet network	0.13	18	This paper

Table 4. A comparison of different input-output models of the hydraulic actuator.

In this modeling problem, wavelet and neural networks perform equivalently. However, these

results are still not as satisfactory as those obtained in [13] with a state-space model using a neural network with sigmoid functions; state-space modeling with wavelet networks will not be considered in the present paper.

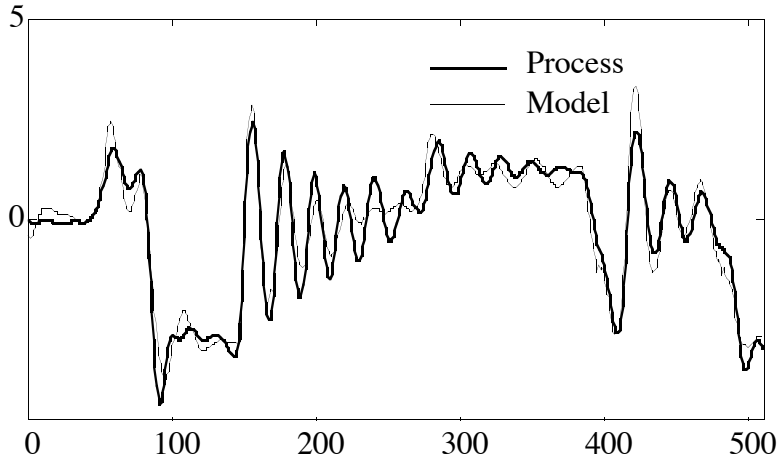


Figure 4. Model and process outputs on the test sequence.

6. CONCLUSION.

In this paper, we extend the use of wavelet networks for function approximation to dynamic nonlinear input-output modeling of processes. We show how to train such networks by a classic minimization of a cost function through second order gradient descent implemented in a backpropagation scheme, with appropriate initialization of the translation and dilation parameters. The training procedure is illustrated on the modeling of simulated and real processes. A comparison with classic sigmoidal neural networks leads to the conclusion that the two types of networks can perform equivalently in terms of accuracy and parsimony for nonlinear input-output modeling of processes with a small number of inputs, provided the technical precautions outlined above (proper initialization and efficient training algorithms) are taken.

References.

[1] M. Cannon and J.-J. E. Slotine, *Space-Frequency Localized Basis Function Networks for Nonlinear System Estimation and Control,* Neurocomputing 9 (3) (1995) 293-342.

[2] G. Cybenko, *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of control, signals and systems, 2 (1989) 303-314.

[3] K. Hornik, M. Stinchcombe, H. White and P. Auer, *Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives*, Neural Computation, 6 (6) (1994) 1262-1275.

[4] M. I. Jordan, *The Learning of Representations for Sequential Performance*, Doctoral Dissertation, University of California, San Diego, 1985.

[5] A. U. Levin, *Neural networks in dynamical systems; a system theoretic approach*, PhD Thesis, Yale University, New Haven, CT, 1992.

[6] S. Mallat, *A Theory for Multiresolution Signal Decomposition: The Wavelet Transform*, IEEE Trans. Pattern Anal. Machine Intell. 11 (7) (1989) 674-693.

[7] K. S. Narendra and K. Parthasarathy, *Identification and Control Of Dynamical Systems Using Neural Networks*, IEEE Trans. on Neural Networks, 1 (1) (1990) 4-27.

[8] O. Nerrand, P. Roussel-Ragot. L. Personnaz, G. Dreyfus, *Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms*, Neural Computation, 5 (2) (1993) 165-199.

[9] O. Nerrand , P. Roussel-Ragot, D. Urbani, L. Personnaz, G. Dreyfus, *Training recurrent neural networks: why and how? An Illustration in Process Modeling*, IEEE Trans. on Neural Networks 5 (2) (1994) 178-184.

[10] Y. C. Pati and P. S. Krishnaparasad, *Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformations,* IEEE Trans. on Neural Networks 4 (1) (1993) 73-85.

[11] E. Polak, *Computational Methods in Optimization: A Unified Approach* (Academic Press, New–York, 1971).

[12] P. Pucar and M. Millnert, *Smooth Hinging Hyperplanes - An Alternative to Neural Nets*, in: Proceedings of 3rd European Control Conference, Vol. 2 (Rome, 1995) 1173-1178.

[13] I. Rivals, L. Personnaz, G. Dreyfus, J.L. Ploix, *Modélisation, Classification et Commande par Réseaux de Neurones : Principes Fondamentaux, Méthodologie de Conception et Illustrations Industrielles,* in: J.P. Corriou, ed., Les réseaux de Neurones pour la Modélisation et la Commande de Procédés (Lavoisier Tec et Doc, 1995) 1-37.

[14] I. Rivals and L. Personnaz, *Black Box Modeling With State-Space Neural Networks*, in: R. Zbikowski and K. J. Hunt eds., Neural Adaptive Control Technology I (World Scientific, Singapore, 1996) 237-264.

[15] D. E. Rumelhart, and J. L. McClelland, *Parallel Distributed Processing*, (MIT Press, Cambridge, MA, 1986).

[16], D. Urbani, P. Roussel-Ragot. L. Personnaz and G. Dreyfus, *The Selection of Neural Models of Non-linear Dynamical Systems by Statistical Tests*, in: Proceedings of the IEEE Conference on Neural Networks for Signal Processing IV, (Greece ,1994) 229-237.

[17] Q. Zhang and A. Benveniste, *Wavelet Networks*, IEEE Trans. on Neural Networks 3 (6) (1992) 889-898.

[18] J. Zhang, G. G. Walter, Y. Miao and W. N. Wayne Lee, *Wavelet Neural Networks For Function Learning*, IEEE Trans. on Signal Processing 43 (6) (1995) 1485-1497.

Annexe B

Présentation du calcul du gradient de la fonction de coût *J* dans le cas d'un réseau d'ondelettes d'état avec possibilité de choisir la sortie comme variable d'état

Nous présentons ici le calcul du gradient de J pour un réseau d'état où la sortie peut être choisie comme une des variables d'état.
Le nombre total des états sera N_s avec :

$$N_s = N_{sy} + N_{ss} \tag{1}$$

Si la sortie est une variable d'état alors on a : N_{sy} =1. Sinon il vaut zéro. N_{ss} est donc le nombre des variables d'état différentes de la sortie.

1. Notations.

Pour pouvoir indicer les neurones d'état, on introduit une variable logique associée à N_{sy} ,définie de la façon suivante :

$$A_{sy} = \begin{cases} 1 \; si \; N_{sy}=0 \\ 0 \; si \; N_{sy}=1 \end{cases} \tag{2}$$

Dans un cas général, les neurones d'état seront indicés de $N_e+N_s+N_w+A_{sy}+1$ à $N_e+N_s+N_w+N_{ss}+1$. Les paramètres du réseau sont donc :

- les translations m_{jk} et les dilatations d_{jk} avec k=1 , ... , N_e+N_s et j=1 , ... , N_w ;
- les pondérations et les coefficients directs : on note c_{kj} le paramètre associé à la connexion entre la fonction (ou le neurone d'entrée) j et le neurone de sortie (ou le neurone d'état) k. Pour les pondérations nous avons $j= N_e+N_s+1$, ... , $N_e+N_s+N_w$ et $k=N_e+N_s+N_w+1$, ... ,$N_e+N_s+N_w+N_{ss}+1$; pour les coefficients directs nous avons j=1 , ... , N_e+N_s et $k=N_e+N_s+N_w+1$, ... ,$N_e+N_s+N_w+N_{ss}+1$;
- un terme constant sur le neurone linéaire de sortie, noté c_0 ;

Le nombre de composantes du vecteur θ est alors $2N_w(N_e+N_s)+(N_{ss}+1)(N_e+N_s+N_w)+1$.

La sortie y ainsi que l'expression des variables d'état en sortie sont identiques à celles données par les relations (59) et (60) du chapitre III.
Pour chaque copie du réseau (n=2, ... , N), les variables d'état en entrée sont calculées à partir de la relation suivante :

$$x_k^n = x_{k+N_s+N_w+A_{sy}}^{n\pm1} \quad \text{avec } k = N_e+1 , \ldots , N_e+N_{sy}+N_{ss} \tag{3}$$

Le cas particulier de la première copie est discuté au paragraphe VI.4.1 du chapitre III.

2. Calcul du gradient de *J* par rapport aux états par rétropropagation.

Pour <u>la copie *N*,</u> nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^N} = \pm e^N \tag{4}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... ,$N_e+N_s+N_w+N_{ss}+1$:

$$\frac{\partial J}{\partial x_k^N} = 0 \tag{5}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... ,N_e+N_s :

$$\frac{\partial J}{\partial x_k^N} = \frac{\partial J}{\partial y^N}\frac{\partial y^N}{\partial x_k^N} = \pm e^N\left(c_{\alpha,k} + \sum_{j=1}^{N_w}\frac{c_{\alpha,N_e+N_s+j}}{d_{jk}}\frac{\partial \Phi_j(x)}{\partial z_{jk}}\right) \tag{6}$$

avec $\alpha = N_e+N_s+N_w+1$.

Pour <u>les copies de n=*N*–1 à 2</u>, nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^n} = \frac{\partial J}{\partial x^n_{N_e+N_s+N_w+1}} = \begin{cases} \pm e^n \;\; si\; N_{sy}=0 \\ \pm e^n + \dfrac{\partial J}{\partial x^{n+1}_{N_e+1}} \;\; sinon \end{cases} \tag{7}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... ,$N_e+N_s+N_w+N_{ss}+1$:

$$\frac{\partial J}{\partial x_k^n} = \frac{\partial J}{\partial S^{n+1}_{k\pm N_s\pm N_w\pm A_{sy}}} \tag{8}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... ,N_e+N_s :

$$\frac{\partial J}{\partial x_k^n} = \frac{\partial J}{\partial y^n}\frac{\partial y^n}{\partial x_k^n} = \pm e^n\left(c_{\alpha,k} + \sum_{j=1}^{N_w}\frac{c_{\alpha,N_e+N_s+j}}{d_{jk}}\frac{\partial \Phi_j(x)}{\partial z_{jk}}\right) + \sum_{j=N_e+N_s+N_w+2}^{N_e+N_s+N_w+N_{ss}+1} c_{j,k}\frac{\partial J}{\partial x_j^n} \tag{9}$$

Pour la <u>copie n=1</u>, nous avons :

Pour la sortie :

$$\frac{\partial J}{\partial y^1} = \frac{\partial J}{\partial x^1_{N_e+N_s+N_w+1}} = \left\{ \begin{array}{c} \pm e^1 \text{ si } N_{sy} \\ \pm e^1 + \dfrac{\partial J}{\partial x^2_{N_e+N_s+N_w 1}} \text{ sinon} \end{array} \right\} \tag{10}$$

Pour les variables d'état en sortie, $k=N_e+N_s+N_w+2$, ... ,$N_e+N_s+N_w+N_{ss}+1$:

$$\frac{\partial J}{\partial x^1_k} = \frac{\partial J}{\partial x^2_{k\pm N_s\pm N_w\pm A_{sy}}} \tag{11}$$

Pour les variables d'état en entrée, $k=N_e+1$, ... ,N_e+N_s : le calcul des $\dfrac{\partial J}{\partial x^1_k}$ n'est pas utile.

3. Calcul du gradient de *J* par rapport aux paramètres du réseau.

Pour les coefficients directs sur la sortie :

$$\frac{\partial J}{\partial c_{\alpha j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial c^n_{\alpha j}} = \pm \sum_{n=1}^{N} e^n x^n_j \text{ avec } j = 1 , \ldots , N_e+N_s \text{ et } \alpha = N_e+N_s+N_w+1 \tag{12}$$

Pour les coefficients directs sur les états :

$$\frac{\partial J}{\partial c_{k,j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} \frac{\partial x^n_k}{\partial c^n_{k,j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} x^n_j \tag{13}$$

$$\text{avec } j = 1 , \ldots , N_e+N_s \text{ et } k = N_e+N_s+N_w+2 , \ldots , N_e+N_s+N_w+N_{ss}+1$$

Pour les pondérations sur la sortie :

$$\frac{\partial J}{\partial c_{\alpha ,N_e+N_s+j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial c^n_{\alpha ,N_e+N_s+j}} = \pm \sum_{n=1}^{N} e^n \Phi_j(x^n) \tag{14}$$

$$\text{avec } j = 1 , \ldots , N_w \text{ et } a = N_e+N_s+N_w+1$$

Pour les pondérations sur les états :

$$\frac{\partial J}{\partial c_{k ,N_e+N_s+j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} \frac{\partial x^n_k}{\partial c^n_{k ,N_e+N_s+j}} = \sum_{n=1}^{N} \frac{\partial J}{\partial x^n_k} \Phi_j(x^n) \tag{15}$$

$$\text{avec } j = 1 , \ldots , N_w \text{ et } k = N_e+N_s+N_w+2 , \ldots , N_e+N_s+N_w+N_{ss}+1$$

Pour le terme constant sur le neurone de sortie :

$$\frac{\partial J}{\partial c_0} = \sum_{n=1}^{N} \frac{\partial J}{\partial y^n} \frac{\partial y^n}{\partial c^n_0} = \pm \sum_{n=1}^{N} e^n \tag{16}$$

Pour les translations :

$$\frac{\partial J}{\partial m_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial m_{jk}^{n}} = \sum_{n=1}^{N} \frac{1}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \left(c_{\alpha, N_e+N_s+j}\, e^n \pm \sum_{l=N_e+N_s+N_w+2}^{N_e+N_s+N_w+N_{ss}+1} c_{l, N_e+N_s+j} \frac{\partial J}{\partial x_l^n} \right) \quad (17)$$

Pour les dilatations :

$$\frac{\partial J}{\partial d_{jk}} = \sum_{n=1}^{N} \frac{\partial J}{\partial d_{jk}^{n}} = \sum_{n=1}^{N} \frac{z_{jk}^n}{d_{jk}} \frac{\partial \Phi_j(x^n)}{\partial z_{jk}} \left(c_{\alpha, N_e+N_s+j}\, e^n \pm \sum_{l=N_e+N_s+N_w+2}^{N_e+N_s+N_w+N_{ss}+1} c_{l, N_e+N_s+j} \frac{\partial J}{\partial x_l^n} \right) \quad (18)$$