Optimisation pour l'apprentissage et apprentissage pour l'optimisation

Milagros van Grieken

THÈSE

présentée en vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ PAUL SABATIER

Specialité : Mathématiques Appliquées

Milagros VAN GRIEKEN

Optimisation pour l'apprentissage et apprentissage pour l'optimisation

---

Soutenue devant le jury composé de :

BÈS Christian	Examinateur Professeur, Université Toulouse III
HIRIART-URRUTY Jean-Baptiste	Examinateur Professeur, Université Toulouse III
JAN Sophie	Co-Directrice de thèse Maître de conférence, Université Toulouse III
MASMOUDI Mohamed	Directeur de thèse Professeur, Université Toulouse III
MOHAMMADI Bijan	Rapporteur Proffeseur, Université Montpellier II
SCHOENAUER Marc	Rapporteur Directeur de recherche, INRIA Rocquencourt
TERPOLILLI Peppino	Examinateur Ingénieur de recherche, TOTAL-PAU

Laboratoire MIP (UMR 5640)
Université Paul Sabatier - 118 route de Narbonne - 31062 Toulouse Cedex 4

À Jose, merci d'être avec moi, de m'aimer et d'avoir été aussi patient à mon égard. Je t'aime, cette réussite est pour toi.

À mes parents, qui m'avez permis de devenir ce que je suis aujourd'hui. Vous m'avez appris le chemin de la vie et m'avez donné les meilleurs conseils aux meilleurs moments.

À mon frère Carlos, dont le soutien a très fortement contribué à la réussite de cette thèse.

À José et Mireya, pour vos encouragements et votre assistance morale, merci de faire que je me sente comme votre fille.

Remerciements

Je voudrais remercier toutes les personnes qui d'une façon ou d'une autre ont contribué à la réalisation de cette thèse.

Tout d'abord je voudrais remercier mes directeurs de thèse Mohamed Masmoudi et Sophie Jan qui m'ont fait confiance. Mohamed Masmoudi a bien voulu m'accepter en thèse. Il a eu beaucoup de patience et m'a appris beaucoup de choses qui me seront utiles sur le long chemin de la recherche. Je voudrais également le remercier pour toutes les suggestions et idées dont j'ai essayé de tirer le maximum de profit. Sophie Jan, avec ses sages conseils, a su me pousser à persévérer et à faire de cette thèse une réalité. Sans sa lecture attentive et ses corrections, je suppose que le sens de beaucoup de phrases n'aurait pas été le même. Merci pour ton amitié...

Je remercie Marc Schoenauer pour la rapidité avec laquelle il a lu mon manuscrit et l'intérêt qu'il a porté à mon travail.

Je remercie également Bijan Mohammadi d'avoir accepté d'être rapporteur de ce travail.

Je voudrais aussi remercier Crhistian Bès, Jean-Baptiste Hiriart-Urruty et Peppino Terpollili, qui me font l'honneur de participer à mon jury de thèse.

Je voudrais exprimer mon amitié à mes collègues de bureau, Olivier et Sandrine, pour tous les agréables moments passés ensemble, et j'espère qu'ils oublieront les mauvais.

Merci à Fabien, avec qui j'ai partagé l'évolution de cette thèse. Ses suggestions et contributions ont été d'une grande utilité.

En ce qui concerne la partie informatique de ce travail, je remercie Miloslav Grundmann pour son aide en C++.

Un grand merci à Jose, "perinolo", sans le soutien de qui, tout au long de notre chemin ensemble, je suis convaincu que jamais je n'aurais atteint le but que je m'étais proposée.

Un merci très spécial à Claudio Pinto, qui est non seulement un mentor pour moi mais aussi un vrai et grand ami. Merci d'être toujours là.

Merci à la famille Tello pour m'avoir accueilli comme si je faisais partie de cette famille...

Mes remerciements vont également à l'Universitè de Los Andes, pour m'avoir donné l'opportunité de venir compléter mes études en France, je dois à ses professeurs ma formation initiale.

Je tiens aussi à mentionner le plaisir que j'ai eu à travailler au sein du laboratoire MIP, et j'en remercie ici tous ses membres.

Enfin, un grand merci à tous ceux qui m'ont accompagnée pendant ces années de thèse. Copains, copines, collègues et famille, tous ont participé d'une manière ou d'une autre à la réalisation de ce travail.

Table des matières

Table des figures

Liste des tableaux

Introduction

Dans le cadre du projet Monastir regroupant les industriels Renault, PSA, Michelin, SNCF, nous avions pu voir les limites des méthodes de linéarisation dans le domaine du Crash. Cette expérience, partiellement rapportée en section 3.7.2 page 55, est à l'origine du travail que nous présentons dans cette thèse.

D'autre part, dans de nombreux problèmes d'optimisation pratiques, les critères à minimiser sont le résultat de l'exécution de codes de calculs longs auxquels on ne peut pas se permettre de faire appel trop souvent. Il est également assez fréquent que les dérivées de ces critères par rapport aux paramètres à optimiser soient inaccessibles. Dans d'autres cas, il se peut que le phénomène physique soit tellement instable que le gradient, bien que calculable, ne soit pas d'une grande utilité en raison de son caractère trop local. C'est essentiellement pour ces raisons que nous nous sommes intéressés à la construction de modèles qui fournissent une bonne approximation peu coûteuse en temps de calcul des critères à minimiser. Afin de pouvoir utiliser des techniques d'optimisation basées sur l'utilisation du gradient, nous cherchions également à bâtir des modèles qui disposent de dérivées d'ordre un. Nous sommes parvenus à satisfaire toutes ces exigences en nous servant des réseaux de neurones.

Les réseaux de neurones cherchent à imiter la structure (neurones, synapses, ...) du cerveau humain. En 1943, Warren McCulloch (neurophysiologiste) et Watter Pitts (mathématicien) [51] ont construit un réseau de neurones capable de traiter des nombres binaires.

En 1949, Donals Hebb décrit les processus d'apprentissage. Ces travaux sont à l'origine des fonctions d'apprentissage utilisées actuellement. En 1957, Frank Rosenblatt crée le premier modèle de réseau neuronal avec apprentissage supervisé. Ce modèle est très important, car il est capable de généraliser. Cela signifie qu'il a la capacité de restituer une réponse correcte pour des données non apprises. Un des inconvénients de ce modèle est qu'il n'a pas la capacité de résoudre des problèmes qui ne sont pas linéairement séparables. Les réseaux de neurones sont utilisés pour la première fois pour résoudre des problèmes réels en 1960 : ADALINE (ADAptative LInear Elements).

Les méthodes neuronales sont abandonnées par la communauté scientifique en 1969. Elles reviennent au goût du jour dans les années 80, avec, la même année (1985) la conception de l'algorithme de rétro-propagation du gradient (que l'on appelle méthode adjointe en mathématiques appliquées) et la publication du livre de John Hopfield sur les réseaux de neurones. En 1986, Rumel Hart et McClelland utilisent l'algorithme de rétro-propagation du gradient pour résoudre des problèmes hors de porté du perceptron : ils utilisent des réseaux multicouches.

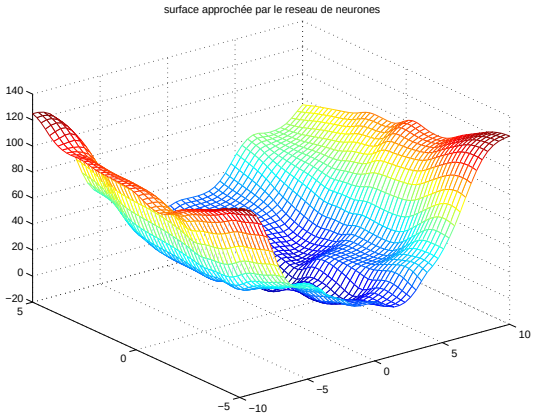
Les réseaux de neurones sont utilisés dans de nombreuses applications issues de différents

domaines (cf. chapitre 3). Bien que l'algorithme d'apprentissage supervisé nécessite lui-même la résolution d'un problème d'optimisation, l'utilisation directe de réseaux de neurones pour résoudre des problèmes d'optimisation globale n'est pas, à notre connaissance, encore très habituelle. On peut trouver des travaux sur l'optimisation en utilisant les réseaux de neurones depuis quelques années [60, 11, 57, 71, 68, 32]. On peut citer des travaux récents combinant méthodes neuronales et algorithmes génétiques [3, 14, 62, 30, 75, 55].

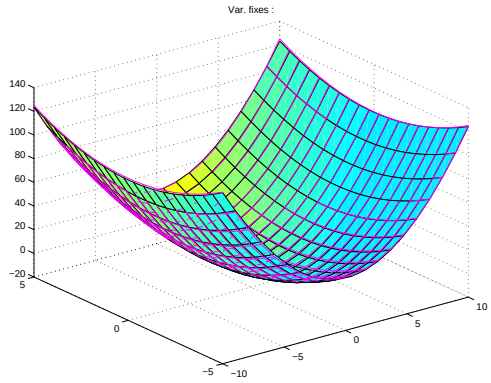
Afin d'illustrer immédiatement l'apport de notre travail, nous allons comparer un résultat d'approximation de la fonction parabole ($(x, y) \longmapsto x^2 + y^2$) par une approche neural classique. Nous présentons la meilleure approximation que nous avons obtenu avec la « toolbox » de Matlab.

On peut constater sur cette figure 1 que Matlab introduit des oscillations pour approcher une fonction lisse.

Au contraire, nous verrons à la section 4.2.1 (page 70) que la méthode que nous proposons nous permet d'approcher une fonction oscillante, dont on cherche le minimum global, par un réseau lisse.



Réseau neuronal classique



Notre méthode zéro-mémoire

FIG. 1 – Comparaison à Matlab

Dans le chapitre 1, nous allons motiver notre étude de l'utilisation des réseaux de neurones en optimisation. Dans un premier temps, nous expliquerons comment ce travail peut apporter une solution au problème de la perte de stabilité lors de la linéarisation d'un problème d'évolution non linéaire. Ensuite, on s'intéressera à l'optimisation de fonctions qui présentent plusieurs minima locaux (fonctions oscillantes). Enfin, on décrira brièvement l'intérêt de ce travail pour l'optimisation de fonctions de type "boite noire" [52].

Le chapitre 2 porte sur les plans d'expérience dont nous nous servons pour choisir d'une manière judicieuse les couples utilisés au cours de l'apprentissage du réseau. Après quelques rappels sur les plans d'expérience, nous décrirons comment nous les utilisons pour construire des couples d'apprentissage.

Au chapitre 3, nous commencerons par de brefs rappels sur les réseaux de neurones (sujet amplement développé dans la littérature depuis les années 80 : [1, 15, 16, 17, 33, 34, 35, 36,

45, 59, 69]). Nous décrirons ensuite une méthode d'apprentissage zéro-mémoire que nous avons développée en couplant des techniques telles que la méthode de Gauss-Newton, le gradient conjugué, les modes de la différentiation automatique, ... Nous détaillerons également quelques méthodes de régularisation utilisées pour pallier à d'éventuelles erreurs sur les données. Ces méthodes font intervenir : la méthode de Tikhonov, la stratégie d'arrêt de l'apprentissage, la taille du modèle et pour terminer l'utilisation de la méthode de Gauss-Newton. Cette approche de régularisation permet en plus d'éviter les minima locaux (qui posent un sérieux problème pour les méthodes classiques), en augmentant la taille du modèle pour assurer l'apprentissage et en le réduisant ensuite pour la régularisation. Enfin, nous illustrerons la méthode d'apprentissage par des résultats numériques, aussi bien sur des cas académiques, que sur des cas industriels. Tous les aspects présentés, sur l'exemple de réseaux neuronaux peuvent, a priori, être adaptés à tous problèmes inverses

Au dernier chapitre, nous verrons comment nous avons exploité les réseaux de neurones et leur gradient pour résoudre des problèmes d'optimisation des types présentés au chapitre 1.

Chapitre 1

Motivation à l'utilisation des réseaux de neurones

1.1 Dérivation d'un problème d'évolution non linéaire

Les problèmes modélisés par des équations d'évolution non linéaires comme l'équation de Burger, les équations de Navier-Stokes ou le problème du pendule possèdent des propriétés de stabilité. Ces dernières sont essentiellement basées sur une estimation de type énergie et découlent du fait que le problème non linéaire décrit un phénomène physique. En revanche, les problèmes linéarisés associés ne bénéficient pas nécessairement de ces mêmes bonnes propriétés. Comme la dérivée de la solution d'un problème d'évolution non linéaire par rapport à un paramètre de conception donné est justement solution de l'équation linéarisée, on ne peut la prendre en considération que sur un intervalle de temps fini T. Elle a donc un caractère trop local et se révèle peu pertinente pour les problèmes d'optimisation.

1.1.1 Équation de Burger

L'équation de Burger

$$u_t + \left(\frac{u^2}{2}\right)_x = 0$$

ou, sous sa forme quasi-linéaire,

$$u_t + uu_x = 0$$

est un modèle qui présente les mêmes caractères fondamentaux que le système de 3 équations (conservation de la masse, de la quantité de mouvement et de l'énergie) à 3 inconnues (par exemple la vitesse, la pression et l'entropie spécifique) de la dynamique des gaz unidimensionnelle [53].

Elle représente également le phénomène du "bang sonique" : loin d'un avion supersonique et en particulier près du sol, le bruit engendré par l'avion se concentre dans certaines zones où la pression est gouvernée par l'équation de Burger [18].

Le terme $\left(\frac{u^2}{2}\right)_x$ est un terme non linéaire de convection.

Si on ajoute un terme linéaire mais dissipatif à l'équation écrite ci-dessus

$$u_t + \left(\frac{u^2}{2}\right)_x = \nu u_{xx},$$

avec $\nu > 0$, on obtient une équation de type parabolique.

Le système qui suit est un assez bon modèle des équations de Navier-Stokes qui seront vues à la section suivante.

$$\begin{aligned} u_t + u\,u_x &= \nu u_{xx}, \qquad \text{dans } [0,L]\times]0,T] \\ u(x,t=0) &= u_0(x), \\ u(x=0,t) &= u_0(0) = 0, \\ u(x=L,t) &= u_0(L) = 0. \end{aligned} \tag{1.1}$$

En multipliant (1.1) par u et en intégrant par rapport à x, on obtient

$$\int_0^L u u_t = \nu \int_0^L u\,u_{xx} - \int_0^L u^2 u_x \tag{1.2}$$

ou encore

$$\frac{1}{2}(\int_0^L u^2)_t = \nu \int_0^L u\,u_{xx} - \int_0^L u^2 u_x. \tag{1.3}$$

Moyennant quelques intégrations par parties et l'utilisation des conditions aux limites, on a

$$\int_0^L u\,u_{xx} = -\int_0^L (u_x)^2 + [u\,u_x]_0^L = -\int_0^L (u_x)^2 \tag{1.4}$$

pour le terme de viscosité et

$$-\int_0^L u^2 u_x = 2\int_0^L u^2 u_x - [u^3]_0^L = 2\int_0^L u^2 u_x \tag{1.5}$$

pour le terme non linéaire. Cela veut dire que,

$$\int_0^L u^2 u_x = 0.$$

En reportant (1.4) dans (1.3), et en utilisant (1.5), on aboutit à

$$\frac{1}{2}(\int_0^L u^2)_t = -\nu \int_0^L (u_x)^2 < 0. \tag{1.6}$$

La dernière équation montre que la contribution du terme non linéaire est nulle et que l'énergie cinétique décroît.

Appelons U la dérivée de u par rapport à un paramètre quelconque. Pour simplifier les calculs, nous disons que ce paramètre est ν. Alors U est la solution de

$$U_t + U\,u_x + u\,U_x = \nu U_{xx} + u_{xx}. \tag{1.7}$$

Appliquons la même technique que pour traiter l'équation (1.1) : en multipliant par U et en intégrant par rapport à x, on a

$$\frac{1}{2}(\int_0^L U^2)_t = \nu \int_0^L U\,U_{xx} + \int_0^L U\,u_{xx} - \int_0^L U^2 u_x - \int_0^L u\,U\,U_x. \tag{1.8}$$

En faisant quelques intégrations par parties et en tenant compte des conditions aux limites, on obtient

$$\int_0^L U\,U_{xx} = [U\,U_x]_0^L - \int_0^L (U_x)^2 = -\int_0^L (U_x)^2, \tag{1.9}$$

$$\int_0^L U\,u_{xx} = [U\,u_x]_0^L - \int_0^L U_x u_x = -\int_0^L U_x u_x, \tag{1.10}$$

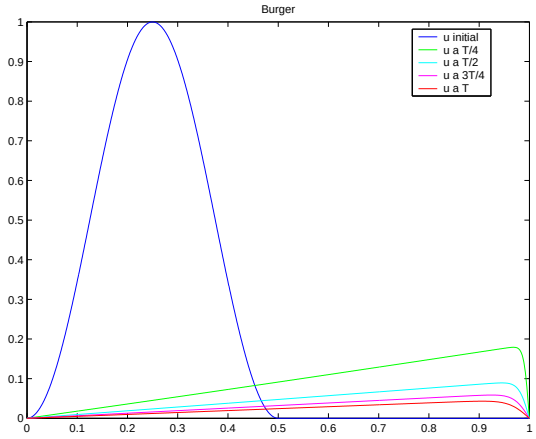
$$-\int_0^L U^2 u_x = [-U^2 u]_0^L + 2\int_0^L U\,U_x\,u = 2\int_0^L U\,U_x\,u. \tag{1.11}$$

En reportant dans (1.8), on obtient

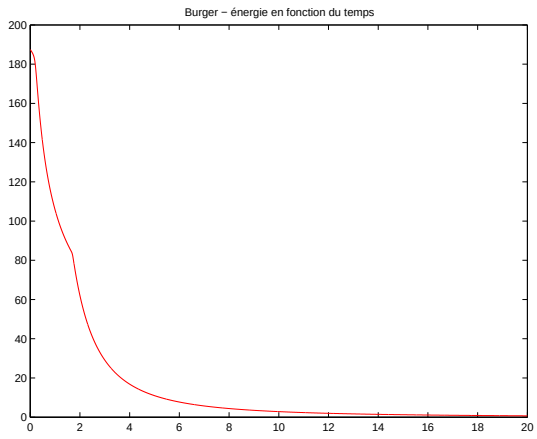
$$\frac{1}{2}(\int_0^L U^2)_t = -\nu \int_0^L (U_x)^2 - \int_0^L U_x u_x + \int_0^L U\,U_x\,u. \tag{1.12}$$

On perd les propriétés de stabilité pour de petites valeurs de ν, et on ne peut plus garantir la décroissance de l'énergie.

On illustre ce phénomène numériquement avec $u_0 = \dfrac{1-\cos(4\pi x)}{2}$ $0 \le x \le 0.5$ et $\nu = 10^{-3}$.



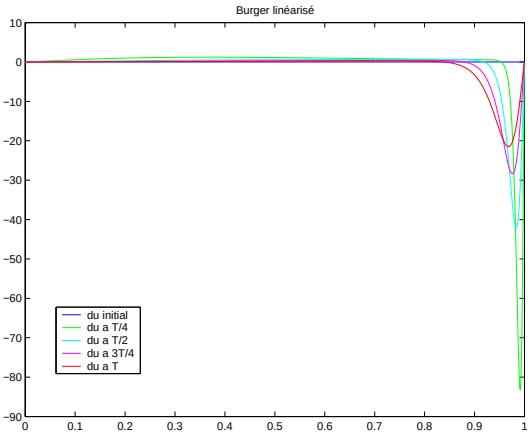
Profils de u pour t $= 0$, $T/4$, $T/2$, $3T/4$ et T



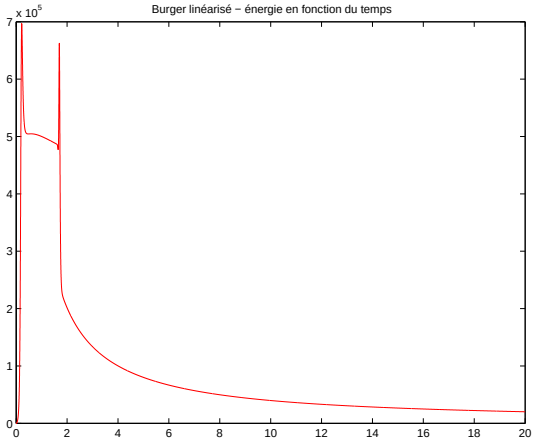
Énergie en fonction du temps

FIG. 1.1 – Équation de Burger

D'après la figure 1.1, on peut noter que l'énergie décroît en fonction du temps. En revanche, dans le cas linéarisé (figure 1.2), on ne peut pas faire la même affirmation : l'énergie "explose". On en déduit que le calcul des dérivées en utilisant l'équation linéarisée n'est pas exploitable.



Profils de U pour t $= 0, T/4, T/2, 3T/4$ et T



Énergie en fonction du temps

FIG. 1.2 – Équation de Burger linéarisée

1.1.2 Navier-Stokes

On considère maintenant le mouvement d'un fluide modélisé par les équations de Navier-Stokes données ci-dessous dans un domaine Ω borné de $\mathbb{R}^2$, dont la frontière Γ est suffisamment régulière.

$$\begin{aligned} u_t + (u.\nabla)u - \nu\Delta u + \nabla p &= 0, \qquad \text{dans } \Omega\times]0,T] && (1.13)\\ \operatorname{div}(u) &= 0, \qquad \text{dans } \Omega\times]0,T] && (1.14)\\ u(x,t) &= 0, \qquad \text{dans } \Gamma\times]0,T] && (1.15)\\ u(x,0) &= u_0(x), \qquad \text{dans } \Omega. && (1.16) \end{aligned}$$

Le vecteur u et le scalaire p représentent respectivement la vitesse du fluide et sa pression.

En multipliant par u et en intégrant par rapport à x, on aboutit à

$$\int_\Omega u\,.u_t = \nu\int_\Omega u\,.\Delta u - \int_\Omega (u\,.\nabla)u\,.u - \int_\Omega u\,.\nabla\, p \qquad (1.17)$$

ou encore, en intégrant par parties le terme $\int_\Omega u\,.\Delta u$ et en tenant compte des conditions aux limites,

$$\frac{1}{2}(\int_\Omega \mid u \mid^2)_t = -\nu\int_\Omega \mid \nabla u \mid^2 - \int_\Omega (u\,.\nabla)u\,.u - \int_\Omega u\,.\nabla p. \qquad (1.18)$$

On a aussi

$$\begin{aligned} -\int_\Omega (u\,.\nabla)u\,.u &= -\int_\Omega \sum_i u_i((\nabla u)u)_i = -\int_\Omega \sum_i u_i \sum_j \frac{\partial u_i}{\partial x_j} u_j \\ &= \int_\Omega \sum_{i,j} u_i (u_i \frac{\partial u_j}{\partial x_j} - u_j \frac{\partial u_i}{\partial x_j}) + \int_\Gamma \sum_{i,j} u_i^2 u_j \\ &= \int_\Omega \sum_i u_i^2 \mathrm{div}(u) + \int_\Omega \sum_i u_i \sum_j u_j \frac{\partial u_i}{\partial x_j} \\ &= \int_\Omega div(u) \mid u \mid^2 + \int_\Omega (u\,.\nabla)u\,.u, \end{aligned}$$

et

$$-\int_\Omega u\nabla p = -\int_\Omega \sum_i u_i \frac{\delta p}{\partial x_i} = \int_\Omega \sum_i \frac{\partial u_i}{\partial x_i} p - \int_\Gamma \sum_i u_i p = -\int_\Omega div(u)p$$

On en déduit, en utilisant (1.14),

$$\frac{1}{2}(\int_\Omega u^2)_t = -\nu \int_\Omega (\nabla u)^2 < 0. \tag{1.19}$$

Encore une fois, on montre que l'énergie cinétique totale décroît. En appliquant la même technique que pour l'équation de Burger, on constate à nouveau une perte de stabilité de la solution de l'équation linéarisée pour de petites valeurs de ν.

1.1.3 Pendule élastique

On considère un ressort de raideur k et de longueur nominale l_n mobile autour d'un axe de rotation. Une masse m est suspendue au ressort dont la longueur devient l, avec $l = \sqrt{x^2 + y^2}$ (x position horizontale, y position verticale).

La masse m est alors soumise à son poids $\overrightarrow{P} = m\overrightarrow{g}$ et à la force de rappel du ressort $\overrightarrow{F} = -k\overrightarrow{IM}$. On néglige la masse du ressort et tous les frottements [64].

Appliquons la relation fondamentale de la dynamique au système, il vient

$$m\overrightarrow{a} = \overrightarrow{F} + \overrightarrow{P} = -k\overrightarrow{IM} + m\overrightarrow{g} \tag{1.20}$$

où $\overrightarrow{a}$ est l'accélération et $\overrightarrow{IM} = \overrightarrow{OM} - \overrightarrow{OI}$

$$\overrightarrow{IM} = \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} \dfrac{l - l_n}{l} x \\ \dfrac{l - l_n}{l} y \end{pmatrix}.$$

En projetant sur les axes, la relation fondamentale de la dynamique donne

$$\begin{cases} mx_{tt} = \dfrac{k(l_n - l)}{l} x, \\ my_{tt} = \dfrac{k(l_n - l)}{l} y - mg. \end{cases}$$

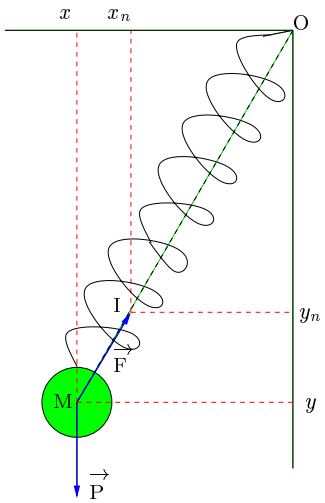


FIG. 1.3 – Pendule élastique

On suppose maintenant que la masse est unitaire et on se donne un paramètre γ qui joue le rôle d'une accélération dont l'objectif pourrait être de contrôler la position horizontale de M. Le système devient alors

$$\begin{cases} x_{tt} = \dfrac{k(l_n - l)}{l} x + \gamma, \\ y_{tt} = \dfrac{k(l_n - l)}{l} y - g. \end{cases} \tag{1.21}$$

Si on dérive (1.21) par rapport à γ, on obtient le système linéarisé

$$\begin{cases} X_{tt} = \dfrac{d}{d\gamma}\left(\dfrac{k(l_n - l)}{l}\right) x + \dfrac{k(l_n - l)}{l} X + 1, \\ Y_{tt} = \dfrac{d}{d\gamma}\left(\dfrac{k(l_n - l)}{l}\right) y + \dfrac{k(l_n - l)}{l} Y, \end{cases}$$

où X (resp. Y) représente la dérivée de x (resp. y) par rapport à γ et

$$\frac{d}{d\gamma}\left(\frac{k(l_n - l)}{l}\right) = -k\frac{l_n}{l^3}\left(xX + yY\right).$$

Le système linéarisé s'écrit alors

$$\begin{pmatrix} X_{tt} \\ Y_{tt} \end{pmatrix} = A \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1.22}$$

avec

$$A = \begin{pmatrix} \dfrac{k(l_n - l)}{l} - \dfrac{kl_n}{l^3}x^2 & -\dfrac{kl_n}{l^3}xy \\ -\dfrac{kl_n}{l^3}xy & \dfrac{k(l_n - l)}{l} - \dfrac{kl_n}{l^3}y^2 \end{pmatrix}.$$

Calculons les valeurs propres de la matrice A

$$\begin{aligned}
\det(A-\lambda I) &= \left(\frac{k(l_n-l)}{l}-k\frac{l_n}{l^3}x^2-\lambda\right)\left(\frac{k(l_n-l)}{l}-k\frac{l_n}{l^3}y^2-\lambda\right)-k^2\frac{l_n^2}{l^6}x^2y^2 \\
&= \left(\frac{k(l_n-l)}{l}-\lambda\right)^2-\left(\frac{k(l_n-l)}{l}-\lambda\right)\left(k\frac{l_n}{l^3}y^2+k\frac{l_n}{l^3}x^2\right) \\
&= \left(\frac{k(l_n-l)}{l}-\lambda\right)\left[\left(\frac{k(l_n-l)}{l}-\lambda\right)-k\frac{l_n}{l}\right] \\
&= \left(\frac{k(l_n-l)}{l}-\lambda\right)(-k-\lambda)\,.
\end{aligned}$$

Les deux valeurs propres de A sont donc $\lambda_1=-k$ et $\lambda_2=\dfrac{k(l_n-l)}{l}$. Si les deux valeurs propres sont négatives, la solution se comporte comme $e^{\pm i\sqrt{\lambda}t}$. En revanche, si A a une valeur propre positive λ, la solution se comporte comme $e^{\pm\sqrt{\lambda}t}$. Dès que $l<l_n$, $\lambda_2=\dfrac{k(l_n-l)}{l}>0$.

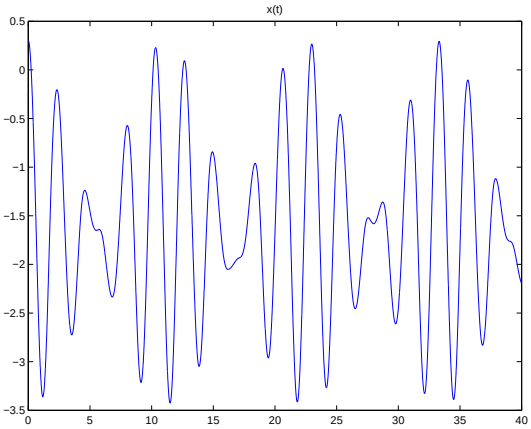
On illustre les propriétés établies ci-dessus pour un ressort de longueur nominale $l_n=1$ et de raideur $k=10$ en prenant comme conditions initiales

$$\begin{cases} x(t=0)=x_0, & x_t(t=0)=0, \\ y(t=0)=y_0, & y_t(t=0)=0. \end{cases}$$

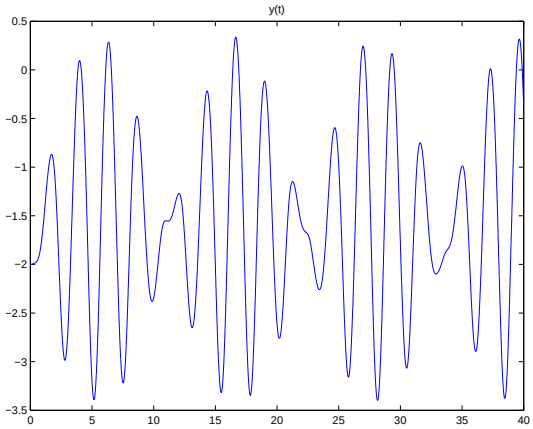
Dans la figure 1.5, on peut observer que si la longueur du ressort à un instant t est plus petite que sa longueur nominale, alors la valeur propre de la matrice A devient positive et les dérivées des positions par rapport à γ "explosent", c'est-à-dire que le système perd sa stabilité.

Par contre, si on regarde la figure 1.4, on peut remarquer que la longueur du ressort est toujours plus grande que sa longueur nominale, et le système linéarisé ne perd pas ses propriétés de stabilité.

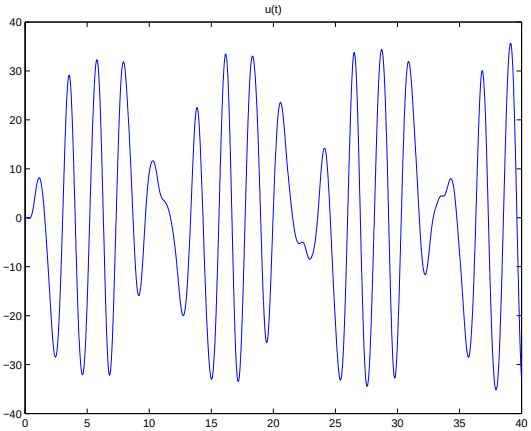
La figure 1.6 représente l'évolution de l'ordonnée $y(T)$ de l'extrémité du ressort à l'instant final considéré en fonction de y_0. On peut constater que la courbe est assez oscillante, en particulier, comme prévu, lorsque $y(0)$ est choisi loin du cercle d'équilibre (ligne rouge). Il est assez aisé de concevoir que le calcul des dérivées dans ce type de situation soit une tâche ardue.



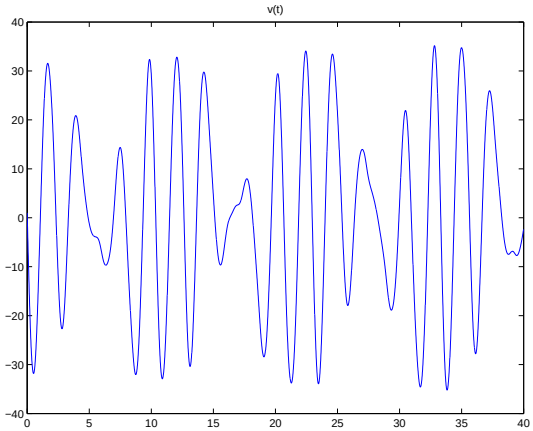
Position horizontale du ressort



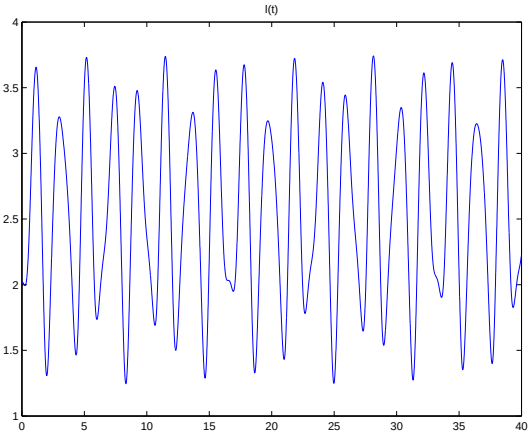
Position verticale du ressort



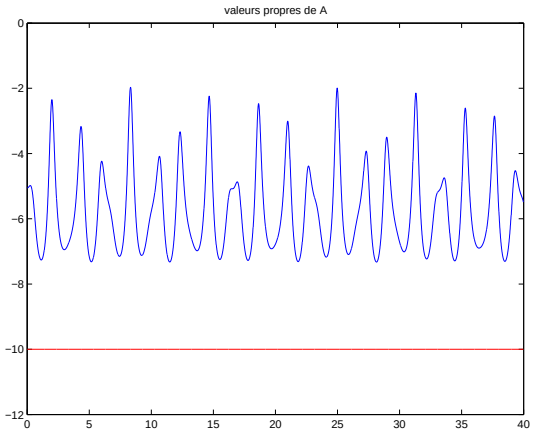
Dérivée de x par rapport à γ



Dérivée de y par rapport à γ

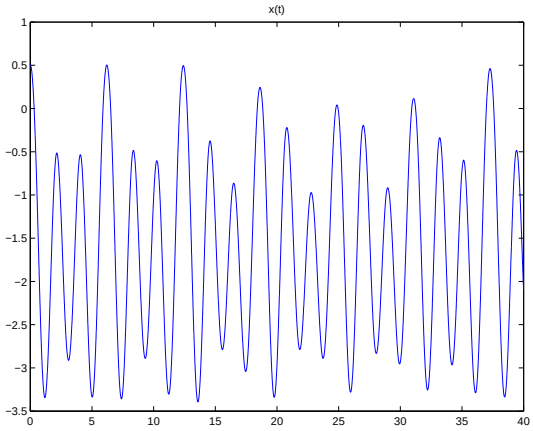


Longueur du ressort

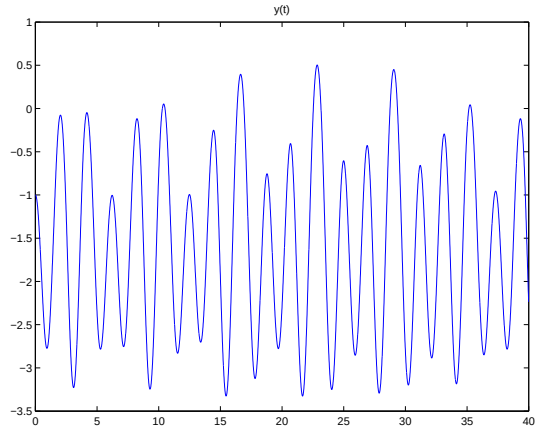


Valeurs propres de A

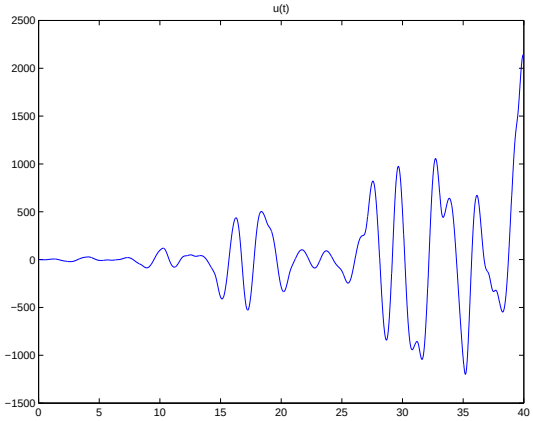
FIG. 1.4 – Système de pendule stable : $(x_0, y_0) = (0.3, -2)$



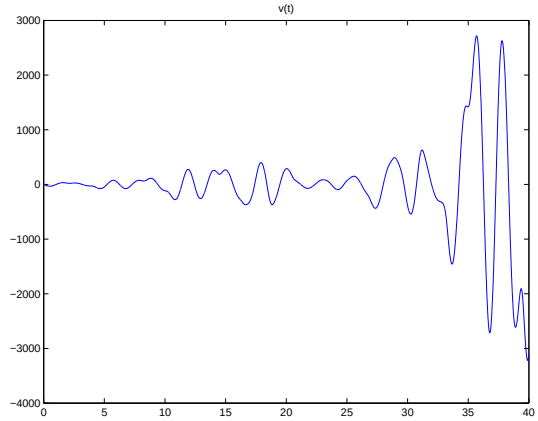
Position horizontale du ressort



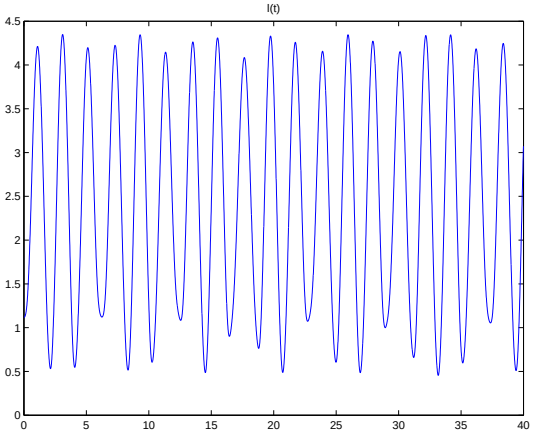
Position verticale du ressort



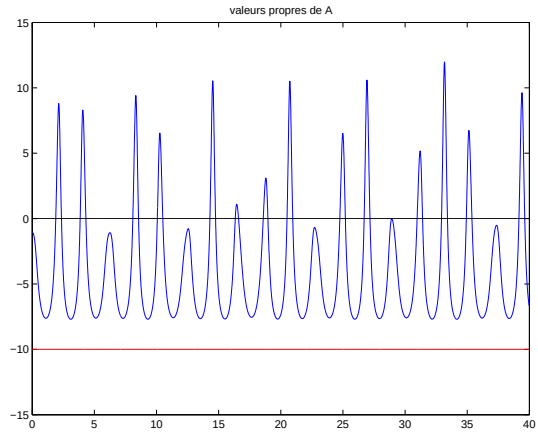
Dérivée de x par rapport à γ



Dérivée de y par rapport à γ



Longueur du ressort



Valeurs propres de A

FIG. 1.5 – Système de pendule instable : $(x_0, y_0) = (0.5, -1)$

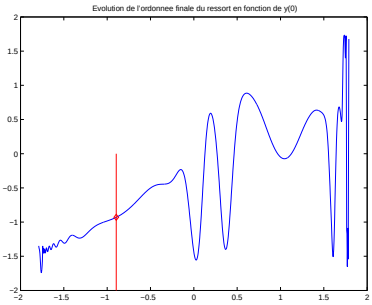


FIG. 1.6 – Evolution de $y(T)$ en fonction de l'ordonnée initiale $y0$

1.2 De l'intérêt des réseaux de neurones

Un moyen de pallier aux problèmes décrits dans la section 1.1 est d'utiliser les réseaux de neurones [33, 37, 4, 40] pour créer une surface de réponse associée au problème et optimiser le modèle ainsi obtenu.

Une autre application possible est l'optimisation de fonctions dont l'évaluation coûte cher et / ou dont on ne peut pas calculer le gradient.

Les réseaux neuronaux permettent également de chercher le minimum global de fonctions oscillantes telles que la fonction de Griewank ou de Rastrigin.

1.2.1 Pour l'optimisation globale

Calculer le minimum global de fonctions très oscillantes est une tâche difficile. Considérons, par exemple, la fonction de Rastrigin représentée sur la figure 1.7 page 14. Elle possède de nombreux minima locaux, mais elle n'a qu'un minimum global, quelle que soit la dimension du problème.

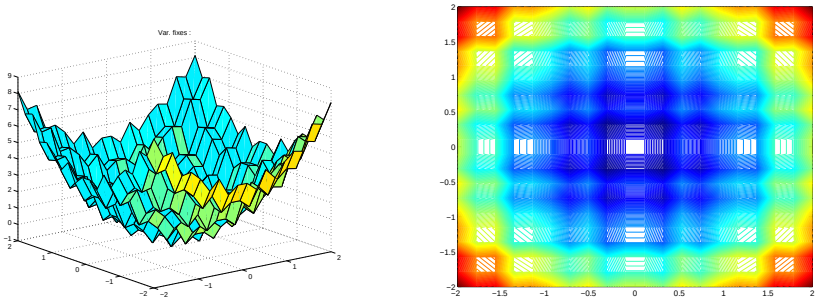


FIG. 1.7 – La fonction de Rastrigin en 2 dimensions et ses contours

Sur un tel exemple, les réseaux de neurones nous permettent d'obtenir une approximation très grossière de la fonction (section 3.7.1 page 49). On exploite ensuite le modèle ainsi obtenu pour obtenir le minimum global (section 4.2.1 page 70) sans être piégé dans les minima locaux.

1.2.2 Pour l'optimisation de fonctions de type « boîte noire »

Dans la plupart des problèmes pratiques ou industriels, la fonction à approcher (et optimiser) n'est pas explicite : on ne la connaît que par des mesures parfois coûteuses ou difficiles à réaliser. Dans ce cas, on construit une surface de réponse à partir des données dont on dispose. Pour ce faire, plusieurs méthodes existent, dont les méthodes neuronales. Un avantage non négligeable de ces dernières est qu'elles permettent un calcul de gradient pour un coût ajouté négligeable.

Considérons, par exemple, le problème de la modélisation d'un choc frontal d'un véhicule sur un mur (section 3.7.2 page 55). On cherche à optimiser un certain critère de déformation en fonction de paramètres tels que la position du pare-chocs ou sa raideur. Une technique naturelle consiste à utiliser une méthode d'optimisation basée sur le gradient du critère. Un calcul de ce dernier via un code de différentiation automatique conduit la démarche à l'échec pour les raisons mentionnées ci-dessus (section 1.1).

Chapitre 2

Les méthodes de plan d'expérience

L'utilisation de réseaux de neurones nécessite une phase d'apprentissage. Ce dernier est basé sur la connaissance de couples d'apprentissage. La qualité de l'apprentissage du réseau dépend en grande partie de la bonne répartition de ces couples dans l'espace de recherche. Dans la majorité des cas, pour des raisons de coût ou de temps de calcul, le nombre de couples d'apprentissage n'est pas très élevé. Il est donc important qu'ils soient judicieusement répartis dans l'espace de recherche. C'est pourquoi nous nous sommes intéressés aux plans d'experience.

Les méthodes de plan d'experience [13] dépassent largement le cadre du calcul numérique et elles sont bien antérieures au développement récent des outils de calcul. Elles ont tout d'abord été utilisées pour réduire le nombre d'expériences réelles à effectuer (réalisation de prototypes, sondages géologiques, ...). Dans la littérature, les paramètres sont appelés facteurs et les différentes valeurs discrètes prises par un facteur sont les niveaux.

Nous commençons par une présentation des plans d'expériences inspirée par une littérature abondante sur ce sujet [73, 12, 61, 66, 6, 67, 25, 26, 27]. Malheureusement, à notre connaissance, il n'existe pas de document accessible présentant une approche systématique pour la construction de plans d'expérience. Nous essayons, dans ce chapitre, de contribuer à combler ce vide.

2.1 Plan factoriel pour deux facteurs à deux niveaux

Prenons par exemple l'étude de la résistance (critère de Von Mises) d'une structure coque dépendant de deux facteurs, l'épaisseur $e \in \{3, 5\}$ et le module d'Young $E \in \{3\times10^4, 4\times10^4\}$.

Les 4 traitements ou combinaisons de niveaux des 2 facteurs, épaisseur et module d'Young, sont

	e codé (A)	e réel	E codé (B)	E réel
1	-1	3	-1	3×10^4
2	$+1$	5	-1	3×10^4
3	-1	3	$+1$	5×10^4
4	$+1$	5	$+1$	5×10^4

où les niveaux des facteurs dits codés sont exprimés dans la même unité : niveau bas à -1

et niveau haut à +1. Ce changement d'unité permet l'étude simultanée de facteurs définis à des échelles différentes et facilite l'approche mathématique de ce type de problème.

Ces 4 traitements peuvent être représentés graphiquement par les sommets du carré dans l'espace défini par les facteurs codés.

Un tel plan est désigné comme plan factoriel 2^2 ou plus généralement plan factoriel a^b où a est le nombre de facteurs et b est le nombre de niveaux.

Nous supposons que la réponse Y est reliée aux niveaux des facteurs codés, notés A et B, par :

$$Y = f(A, B) + \varepsilon$$

où $f(A, B)$ est appelée la réponse théorique et représente la résistance de la structure pour les niveaux considérés de A et B et ε est l'erreur. Pour les 4 traitements, les réponses théoriques sont présentées dans le tableau

A	B	$f(A,B)$
-1	-1	$f(-1,-1)$
1	-1	$f(1,-1)$
-1	1	$f(-1,1)$
1	1	$f(1,1)$

et peuvent être estimées en réalisant des expériences. Néanmoins, elles ne permettent pas de comparer les influences respectives de A et B sur Y.

Pour ce faire, on définit les **effets factoriels** de A et de B de la manière suivante :

– Effet de A : $\frac{1}{2}\Big($ (moyenne de f pour $A = +1$) - (moyenne de f pour $A = -1$) $\Big)$

$$\begin{aligned} e(A) &= \frac{f(1,-1)+f(1,1)}{4} - \frac{f(-1,-1)+f(-1,1)}{4} \\ &= \frac{f(1,-1)+f(1,1)-f(-1,-1)-f(-1,1)}{4} \end{aligned}$$

– Effet de B : $\frac{1}{2}\Big($ (moyenne de f pour $B = +1$) - (moyenne de f pour $B = -1$) $\Big)$

$$\begin{aligned} e(B) &= \frac{f(1,1)+f(-1,1)}{4} - \frac{f(1,-1)+f(-1,-1)}{4} \\ &= \frac{f(1,1)+f(-1,1)-f(1,-1)-f(-1,-1)}{4} \end{aligned}$$

Notant aussi

$$e(A-) = \frac{f(1,-1)-f(-1,-1)}{2} \quad \text{l'effet de } A \text{ lorsque } B = -1$$
$$e(A+) = \frac{f(1,1)-f(-1,1)}{2} \quad \text{l'effet de } A \text{ lorsque } B = 1,$$

on a $e(A+) - e(A-) = 0$ si l'effet de A ne dépend pas de B.

Cette différence $e(A+) - e(A-)$ caractérise la dépendance ou l'indépendance des effets.

On définit donc **l'effet d'interaction** entre A et B par

$$\begin{aligned} e(AB) &= \frac{1}{2}\Big(\frac{f(1,-1)-f(-1,-1)}{2} - \frac{f(1,1)-f(-1,1)}{2}\Big) \\ &= \frac{f(1,-1)-f(-1,-1)-f(1,1)+f(-1,1)}{4} \end{aligned}$$

et la **moyenne générale** notée $e(1)$ par

$$e(1) = \frac{f(-1,-1)+f(1-1)+f(-1,1)+f(1,1)}{4}.$$

Les 4 effets factoriels définis à partir des 4 réponses théoriques sont donc

la moyenne générale $e(1) = \frac{1}{4}(f(-1,-1)+f(1,-1)+f(-1,1)+f(1,1))$,

l'effet de A $e(A) = \frac{1}{4}(f(1,-1)+f(1,1)-f(-1,-1)-f(-1,1))$,

l'effet de B $e(B) = \frac{1}{4}(f(1,1)+f(-1,1)-f(1,-1)-f(-1,-1))$,

l'interaction entre A et B $e(AB) = \frac{1}{4}(f(1,-1)-f(-1,-1)-f(1,1)+f(-1,1))$.

On peut écrire la réponse sous la forme

$$f(A,B) = e(1) + e(A)\times A + e(B)\times B + e(AB)\times A\times B.$$

Si l'on écrit le plan d'expérience sous la forme

1	A	B	AB	$f(A,B)$
1	-1	-1	1	$f(-1,-1)$
1	1	-1	-1	$f(1,-1)$
1	-1	1	-1	$f(-1,1)$
1	1	1	1	$f(1,1)$

les effets précédents sont obtenus en effectuant le produit scalaire des 4 colonnes par la colonne $f(A,B)$ et en divisant le produit scalaire par 4. Ceci peut être écrit sous forme matricielle en considérant la **matrice des effets**

$$H = \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

qui représente les colonnes $1, A, B, AB$ du tableau précédent. On peut alors écrire

$$\begin{pmatrix} e(1) \\ e(A) \\ e(B) \\ e(AB) \end{pmatrix} = \frac{1}{4}H^T \begin{pmatrix} f(-1,-1) \\ f(1,-1) \\ f(-1,1) \\ f(1,1) \end{pmatrix}$$

ou encore

$$e = \frac{1}{4}H^T f.$$

Comme la matrice H est orthogonale ($HH^T = 4I$), on a

$$f = He.$$

D'une manière générale, une matrice H de rang n dont les termes ne prennent que les valeurs ± 1 telle que $HH^T = n\,Id_n$ est appelée matrice de Hadamard. Une telle matrice n'existe que pour $n = 2$ et pour n multiple de 4.

2.2 Plans factoriels fractionnaires pour facteurs à deux niveaux

Considérons un plan factoriel pour 7 facteurs à 2 niveaux. Le plan complet comprend $2^7 = 128$ expériences. Il permet d'estimer 128 effets qui se décomposent en

1	moyenne,
7	effets principaux,
21	interactions entre 2 facteurs,
35	interactions entre 3 facteurs,
35	interactions entre 4 facteurs,
21	interactions entre 5 facteurs,
7	interactions entre 6 facteurs,
1	interaction entre 7 facteurs.

Mais, si l'on peut estimer ces effets, ils ne sont pas tous importants. Il existe une certaine hiérarchie entre eux : les effets principaux tendent à être supérieurs aux interactions de 2 facteurs, qui sont elles plus grandes (en valeur absolue) que les interactions de 3 facteurs, ...

Il est souvent vrai qu'à un certain niveau, les interactions d'ordre le plus élevé sont négligeables et peuvent donc être éliminées. De plus, quand le nombre de facteurs grandit, il arrive souvent que certains d'entre eux n'aient pas d'effet.

Si p n'est pas petit, il existe une redondance dans un plan 2^p qui correspond à un excès du nombre des interactions et parfois du nombre de facteurs envisagés. Les plans factoriels fractionnaires exploitent cette redondance.

Les principes de construction de fractions de plans factoriels 2^p sont présentés ci-dessous pour des cas simples avec $p = 2, 3, 4$ et 5 facteurs.

2.2.1 Cas de deux facteurs à deux niveaux

Dans un plan factoriel complet 2^2 pour les facteurs A et B, le tableau des 4 expériences qui permet le calcul des effets factoriels s'écrit :

1	A	B	AB	$f(A,B)$
1	-1	-1	1	$f(-1,-1)$
1	1	-1	-1	$f(1,-1)$
1	-1	1	-1	$f(-1,1)$
1	1	1	1	$f(1,1)$

Examinons le cas d'école suivant : on ne peut expérimenter que 2 expériences parmi les 4. On choisit de sélectionner le premier et le quatrième pour lesquels $AB = 1$.

1	A	B	AB	$f(A,B)$
1	-1	-1	1	$f(-1,-1)$
1	1	1	1	$f(1,1)$

La réalisation de ces deux expériences permet d'estimer f en 2 points :

$$\begin{aligned} f(-1,-1) &= e(1) + e(AB) - (e(A) + e(B)) \\ f(1,1) &= e(1) + e(AB) + (e(A) + e(B)) \end{aligned}$$

et en résolvant ce système, on pourra calculer $e(1)+e(AB)$ d'une part et $e(A)+e(B)$ d'autre part.

Ces effets, groupés par paquets de 2, indissociables l'un de l'autre, sont dits confondus. Le nombre d'expériences a été divisé par 2 mais les effets que l'on calcule ne sont plus purs, ils sont confondus 2 par 2.

Le cas d'école à 2 facteurs n'est pas très intéressant pratiquement, car on ne peut plus estimer les effets principaux des 2 facteurs.

Si on note X la matrice

$$X = \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

les effets confondus correspondent à des colonnes linéairement dépendantes.

La relation $AB = 1$ qui a permis de sélectionner les expériences du plan factoriel complet est appelée **relation de définition du demi-plan**. On dit que l'effet de AB est **confondu** avec la moyenne générale. La relation $A = B$ peut être retrouvée formellement en faisant le produit d'un facteur donné (par exemple B) par le terme dont l'effet est confondu avec la moyenne générale : en utilisant le fait que $B^2 = 1$, on a

$$AB = 1 \Longrightarrow AB^2 = B \Longrightarrow A = B.$$

2.2.2 Trois facteurs à deux niveaux

Dans le plan complet 2^3 pour les facteurs A, B, C, le tableau permettant le calcul des effets est

1	A	B	C	AB	AC	BC	ABC	f(A,B,C)
1	-1	-1	-1	1	1	1	-1	f(-1,-1,-1)
1	1	-1	-1	-1	-1	1	1	f(1,-1,-1)
1	-1	1	-1	-1	1	-1	1	f(-1,1,-1)
1	1	1	-1	1	-1	-1	-1	f(1,1,-1)
1	-1	-1	1	1	-1	-1	1	f(-1,-1,1)
1	1	-1	1	-1	1	-1	-1	f(1,-1,1)
1	-1	1	1	-1	-1	1	-1	f(-1,1,1)
1	1	1	1	1	1	1	1	f(1,1,1)

Pour réaliser seulement la moitié des 8 essais, choisissons les expériences 2, 3, 5 et 8 pour lesquels $ABC = 1$.

1	A	B	C	AB	AC	BC	ABC	f(A,B,C)
1	1	-1	-1	-1	-1	1	1	f(1,-1,-1)
1	-1	1	-1	-1	1	-1	1	f(-1,1,-1)
1	-1	-1	1	1	-1	-1	1	f(-1,-1,1)
1	1	1	1	1	1	1	1	f(1,1,1)

La détermination de f en ces 4 points permet de résoudre le système suivant :

$$\begin{aligned} f(1,-1,-1) &= e(1)+e(ABC)+(e(A)+e(BC))-(e(B)+e(AC))-(e(C)+e(AB)) \\ f(-1,1,-1) &= e(1)+e(ABC)-(e(A)+e(BC))+(e(B)+e(AC))-(e(C)+e(AB)) \\ f(-1,-1,1) &= e(1)+e(ABC)-(e(A)+e(BC))-(e(B)+e(AC))+(e(C)+e(AB)) \\ f(1,1,1) &= e(1)+e(ABC)+(e(A)+e(BC))+(e(B)+e(AC))+(e(C)+e(AB)) \end{aligned}$$

dont les inconnues sont les paquets d'effets confondus

$$\begin{gathered} e(1)+e(ABC), \\ e(A)+e(BC), \\ e(B)+e(AC), \\ e(C)+e(AB). \end{gathered}$$

Comme dans le cas de 2 facteurs, les effets confondus entre eux correspondent aux colonnes linéairement dépendantes de la matrice

$$X = \begin{pmatrix} 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Contrairement au plan précédent, les effets principaux $e(A)$, $e(B)$, $e(C)$ ne sont plus confondus entre eux mais avec les interactions de 2 facteurs $e(BC)$, $e(AC)$, $e(AB)$. Le demi-plan ainsi obtenu permet de calculer les effets $e(A), e(B), e(C)$ si les interactions $e(BC), e(AC), e(AB)$ sont négligeables.

La relation de définition $ABC = 1$ peut être multipliée par tout autre facteur pour obtenir les autres relations $BC = A, AC = B, AB = C$.

Ici, la confusion des effets principaux avec les interactions de 2 facteurs peut être gênante car les interactions d'ordre 2 ont de faibles chances d'être négligeables. Cette propriété du plan choisi s'exprime avec la notion de résolution : le plan décrit est de résolution III car la relation de définition confond une interaction de 3 facteurs avec la moyenne. Cette résolution III signifie aussi que les effets principaux sont confondus avec les interactions de 2 facteurs.

2.2.3 Quatre facteurs à deux niveaux

Le plan complet 2^4 pour les facteurs A, B, C, D comprend $2^4 = 16$ expériences. Si on reproduit la technique précédente pour choisir $2^3 = 8$ points parmi les 16 du plan complet,

on va sélectionner celles pour lesquelles $ABCD = 1$. Cette relation de définition permet d'établir que $D = ABC$. On peut donc construire les $2^3 = 8$ expériences du plan factoriel pour les 4 facteurs A, B, C et D à partir du plan factoriel complet pour les facteurs A, B, C en calculant la colonne D via le produit terme à terme des colonnes A, B et C.

1	A	B	C	$D = ABC$
1	-1	-1	-1	-1
1	1	-1	-1	1
1	-1	1	-1	1
1	1	1	-1	-1
1	-1	-1	1	1
1	1	-1	1	-1
1	-1	1	1	-1
1	1	1	1	1

A, B, C sont nommés **facteurs de base du plan fractionnaire**. En multipliant la relation de définition par les termes d'ordre 1 (effets principaux), on obtient $D = ABC$, $A = BCD$, $B = ACD, C = ABD$ et si on multiplie par les termes d'ordre 2, on a $CD = AB$, $AC = BD$, $BC = AD$.

Finalement, la relation de définition s'écrivant $ABCD = 1$, l'effet confondu avec la moyenne est une interaction de 4 facteurs : le plan est de résolution IV (quatre)... Cela signifie aussi que les effets principaux (d'ordre 1) sont confondus avec des interactions de 3 facteurs (1+3=4) et que les interactions de 2 facteurs sont confondues entre elles (2+2=4).

L'avantage qualitatif de ce plan par rapport à ceux qui ont été vus précédemment est que les effets principaux ne sont pas confondus avec les interactions d'ordre 2. Ces dernières sont confondues par paires.

2.2.4 Cinq facteurs à deux niveaux

Pour construire deux demi-plans d'un plan factoriel 2^5, on partage en deux le tableau des effets au moyen d'une relation de définition. L'un des demi-plans factoriels est construit en utilisant $ABCDE = 1$, tandis que l'autre est défini par $ABCDE = -1$. Pour chacun d'eux, la moyenne générale est confondue avec une interaction d'ordre 5.

Pour couper le plan en 4, on ne confond plus 1 mais 2 effets avec la moyenne générale. Ceci correspond à l'égalité de 2 colonnes avec la colonne de la moyenne générale. Dans le cas où on confond les effets de l'interaction des 3 facteurs ABC et de l'interaction des 3 facteurs CDE avec la moyenne générale, les expériences retenues vérifient $ABC = CDE = 1$ et sont

au nombre de 8 :

1	A	B	C	D	E	ABC	CDE
1	-1	-1	1	-1	-1	1	1
1	1	-1	-1	1	-1	1	1
1	-1	1	-1	1	-1	1	1
1	1	1	1	-1	-1	1	1
1	-1	-1	1	1	1	1	1
1	1	-1	-1	-1	1	1	1
1	-1	1	-1	-1	1	1	1
1	1	1	1	1	1	1	1

La construction de ce quart de plan est en fait réalisée en notant que $ABC = CDE = 1$ implique $C = AB$, $D = CE = ABE$. Le plan est donc établi en calculant C et D à partir d'un plan complet 2^3 défini pour les facteurs de base A, B et E.

On aurait pu aussi choisir les expériences pour lesquels $ABC = -CDE = 1$, ou bien $-ABC = CDE = 1$, ou bien $ABC = CDE - 1$.

La confusion des effets de ABC et de CDE avec la moyenne générale définit donc 4 fractions de plan différentes.

A partir de la relation $ABC = -CDE = 1$, en effectuant le produit de ABC et de CDE, on déduit que $-1 = ABC^2DE = ABDE$. L'effet de $ABDE$ est donc également confondu avec la moyenne générale.

Les effets confondus entre eux sont donc ceux des 4 termes suivants

$$1, ABC, CDE, ABDE.$$

On en déduit les effets confondus avec un effet quelconque, par exemple celui de A, par multiplication avec chacun des 4 termes :

$$e(A) = e(A^2BC) = e(ACDE) = e(A^2BDE),$$

ou encore

$$e(A) = e(BC) = e(ACDE) = e(BDE).$$

La somme d'effets qui pourra être estimée est donc $e(A)+e(BC)+e(ACDE)+e(BDE)$. Cette somme donnera une estimation de $e(A)$ à condition que les interactions $e(BC)$, $e(ACDE)$ et $e(BDE)$ soient négligeables.

Les sommes comprenant les 4 autres effets principaux et pouvant être estimées sont

$$\begin{aligned} &e(B) + e(AC) + e(BCDE) + e(ADE),\\ &e(C) + e(AB) + e(DE) + e(ABCDE),\\ &e(D) + e(ABCD) + e(CE) + e(ABE),\\ &e(E) + e(ABCE) + e(CD) + e(ABD). \end{aligned}$$

Les effets principaux $e(A), e(B), e(C), e(D), e(E)$ seront donc estimables si les interactions présentes dans ces sommes sont négligeables. A partir des 8 expériences du quart de plan, trois autres sommes d'effets comprenant des interactions pourront aussi être estimées.

Les termes d'ordre le plus faible dont l'effet est confondu avec la moyenne générale sont des interactions de 3 facteurs : on dit que le plan 2^{5-2} construit ici est **de résolution III**.

2.2.5 Bilan

Le nombre de facteurs considérés est p. On n'étudie qu'une fraction $1/2^q$ de l'ensemble des 2^p expériences. Seulement 2^{p-q} expériences sont donc réalisées.

Les 2^p effets factoriels : moyenne, effets principaux, interactions jusqu'à l'ordre p ne pourront tous être estimés car il y a seulement 2^{p-q} observations.

Mais ces effets sont regroupés en 2^{p-q} groupes de 2^q effets. Les 2^q effets de chacun des groupes ne sont pas dissociables : ils sont confondus. On n'estimera qu'une combinaison de ceux-ci. Si l'un des effets de cette combinaison est important et que les autres sont négligeables, on peut considérer que l'effet important est estimable.

Ces plans sont définis par l'ensemble des q effets confondus avec la moyenne générale. Des règles algébriques simples permettent de construire le plan et de trouver les effets confondus.

En formant les produits des q termes confondus avec la moyenne générale, on obtient un ensemble de 2^q effets confondus. Pour obtenir le plan, on construit un plan factoriel complet pour $p-q$ facteurs dits facteurs de base et on en déduit les niveaux des q autres facteurs en utilisant les relations de confusion.

Si l'interaction d'ordre le plus faible appartenant à cet ensemble est une interaction de 3 facteurs, le plan est dit de résolution III ; si c'est une interaction de 4 facteurs, le plan est de résolution IV.

La notation classique d'une fraction $1/2^q$ d'un plan 2^p de résolution R est 2^{p-q}_R.

Avec $p = 5$ facteurs, on a construit une fraction $\frac{1}{4}$ du plan complet 2^5 : $q = 2$, $2^{5-2} = 8$ expériences sont retenus. Les 32 effets factoriels sont confondus par paquets de $2^2 = 4$ effets. Le plan est défini par $ABC = CDE = 1$ relation qui est complétée (par produit des termes entre eux) en $ABC = CDE = ABDE = 1$. Cette dernière relation permet de trouver les effets confondus avec un effet quelconque (par exemple AB) en la multipliant par le terme considéré :

$$AB \times ABC = AB \times CDE = AB \times ABDE = AB \times 1$$
$$C = ABCDE = DE = AB.$$

La combinaison $e(C) + e(ABCDE) + e(DE) + e(AB)$ fait partie des 8 combinaisons estimables. Les facteurs de base sont A, B, E à partir desquels on construit $C = AB, D = ABE$. Le plan appartient à l'ensemble des plans notés 2^{5-2}_{III}.

2.3 Construction récursive de plans factoriels fractionnaires

On s'intéresse ici à développer une méthode de construction des points d'un plan factoriel fractionnaire 2^{p-q}. Par souci de clarté, on notera $Q = p - q$. On cherche donc 2^Q sommets de l'hypercube $[-1, +1]^p$. Nous détaillons ici une manière récursive de construire ces points.

On divise l'espace de recherche en deux sous-espaces de dimension $(p-1)$:

$$(-1) \times [-1, +1]^{p-1} \qquad (+1) \times [-1, +1]^{p-1}.$$

Dans chacun de ces 2 sous-espaces, on cherche 2^{Q-1} points en construisant deux plans factoriels fractionnaires $2^{(p-1)-q}$. On répète cette procédure, de manière récursive, tant que le nombre de points à construire est supérieur à un. A ce niveau, on prend, au hasard, un point dans l'hypercube $[-1,1]^q$.

On note $pf(p,Q,flag)$ le plan factoriel fractionnaire (2^Q points) de p facteurs à 2 niveaux ($flag \in \{-1,1\}$). La méthode récursive que nous proposons est résumée dans la figure 2.1.

Procédure $plan = pf(p,Q,flag)$
si (Q = 0),
plan = $\{(\underbrace{flag, flag, ..., flag}_{p fois})\}$
sinon
$p = p-1$
$Q = Q-1$
$N = 2^Q$
appel à $pf(p,Q,flag)$ qui retourne $\{d_1, d_2, ..., d_N\} \subset \mathbb{R}^p$
appel à $pf(p,Q,-flag)$ qui retourne $\{g_1, g_2, ..., g_N\} \subset \mathbb{R}^p$
$plan = \{(1,d_1),(1,d_2),...,(1,d_N),(-1,g_1),(-1,g_2),...,(-1,g_N)\} \subset \mathbb{R}^{p+1}$
fin du si

FIG. 2.1 – Construction récursive d'un plan factoriel fractionnaire : algorithme $pf(p,Q,flag)$

Par exemple, pour un plan factoriel fractionnaire (2^{2-1} points) de $p = 2$ facteurs à deux niveaux, il y a deux solutions. Une des solutions est donnée par $flag = 1$ et l'autre par $flag = -1$ (cf. fig. 2.2).

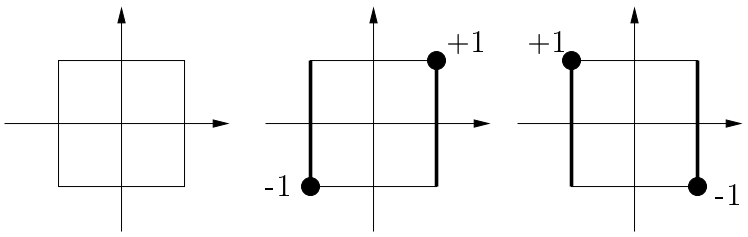


FIG. 2.2 – Plan factoriel fractionnaire 2^{2-1} ($p = 2$, $q = 1$, $Q = 1$)

Un exemple du calcul récursif du plan factoriel fractionnaire 2^{3-1} est montré dans la fig. 2.3 pour trouver quatre points ($Q = 2$) dans $[-1,1]^3$.

2.4 Construction de plans factoriels fractionnaires orthogonaux

La procédure décrite ci-dessus fonctionne très bien si la dimension p du problème n'est pas grande et si la différence entre p et Q n'est pas plus grande que 2 ($q = p - Q \leq 2$).

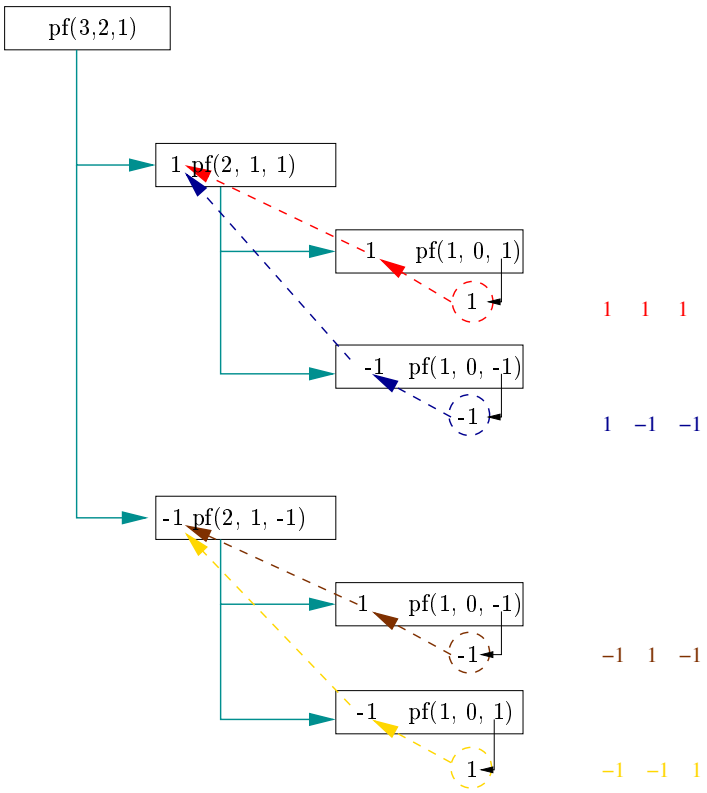


FIG. 2.3 – Plan factoriel fractionnaire 2^{3-2} ($p = 3$, $q = 2$, $Q = 1$)

Sinon, on risque de créer des plans factoriels qui sont linéairement dépendants. On constate en effet sur la figure 2.4 que les 2 dernières colonnes de la matrice à 4 lignes et 5 colonnes construite en utilisant $pf(5, 2, 1)$ sont identiques.

Pour éviter ce problème, on commence par construire un plan factoriel complet de Q facteurs à 2 niveaux. Considérons l'ensemble des points ainsi obtenus comme un tableau de 2^Q lignes et Q colonnes. L'objectif est ensuite de compléter ce tableau, en rajoutant des colonnes linéairement indépendantes, entre elles et avec celles déjà construites, jusqu'à l'obtention d'un nouveau tableau à 2^Q lignes et p colonnes. Cet algorithme est décrit dans la figure 2.5.

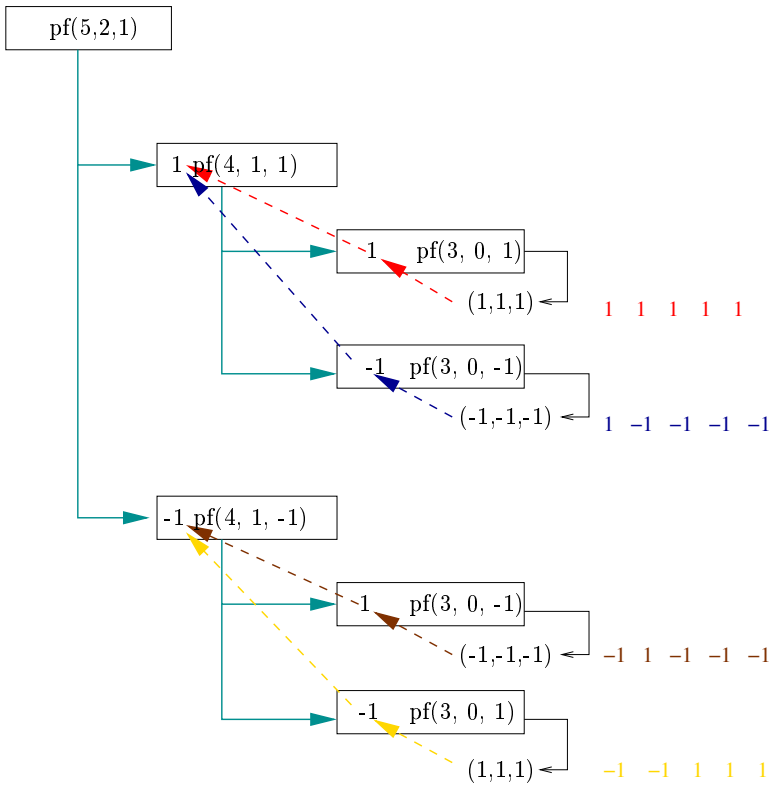


FIG. 2.4 – Plan factoriel fractionnaire 2^{5-3} $(p = 5,\ q = 3,\ Q = 2)$: non orthogonal

Procédure $plan = pfo(p, Q, flag)$
Calculer le plan factoriel complet $pf(Q, Q, flag)$.
Notons $(c_1, ..., c_Q)$ les colonnes du tableau ainsi construit.
Calculer le nombre des colonnes à ajouter : $nbcol = p - Q$.
Pour $k = 1$ jusqu'à $k = nbcol$, répéter
(a) sélectionner au hasard deux colonnes distinctes c_i et c_j,
(b) notons nc le produit terme à terme de c_i et c_j,
(c) si la nouvelle colonne nc est linéairement indépendante de $(c_1, c_2, ..., c_{Q+(k-1)})$,
$c_{Q+k} = nc$,
sinon
retourner à (a).
fin pour.

FIG. 2.5 – Plan factoriel fractionnaire orthogonal : algorithme $pfo(p, Q, flag)$

2.5 Application à la génération de points bien répartis dans un domaine donné

L'approximation et l'optimisation d'une fonction de p variables par réseaux de neurones nécessite la connaissance d'un ensemble de points initiaux afin de construire un premier modèle. Ce modèle initial doit être pertinent, sans pour autant exiger un nombre important de points, car les fonctions à approcher peuvent être très coûteuses en temps de calcul.

La façon la plus facile de choisir les points initiaux est d'utiliser une série pseudo-aléatoire. Mais une telle méthode ne permet jamais d'obtenir des informations pour les valeurs extrêmes du domaine de recherche. Dans ce cas, dans les zones extrémales, le réseau fonctionne en extrapolation.

Une interpolation étant toujours préférable, tant pour l'apprentissage que pour l'optimisation, les points d'initialisation du modèle sont répartis de manière à donner au modèle une information dans les coins de l'ensemble de définition. Cependant, comme il existe une infinité de fonctions quadratiques qui ont la même valeur en tous les sommets d'un hypercube, il est indispensable de considérer des points à l'intérieur du domaine de recherche pour interpoler une fonction.

Dans un espace de grande dimension p, il n'est pas possible d'utiliser tous les sommets de l'hypercube comme points initiaux : en effet, leur nombre grandit exponentiellement avec la dimension du problème.

Pour toutes ces raisons, nous choisissons trois types de points :
- des sommets de l'hypercube externe (dont les coordonnées sont dans $\{-1, 1\}^p$),
- des sommets d'un hypercube interne (dont les coordonnées sont dans $\{-1/3, 1/3\}^p$),
- des points tirés aléatoirement dans $[-1, 1]^p$.

Dans certains cas, le nombre de points initiaux est limité et ne permet pas d'utiliser les trois types de points décrits ci-dessus. Nous procédons alors comme dans la figure 2.6. Le choix de l'entier Q est guidé par la volonté de construire une matrice de taille $2^Q \times p$ avec des colonnes linéairement indépendantes.

Soit N_{init} le nombre de points initiaux.
On cherche Q tel que $p \leq 2^Q$.
Si $N_{init} >= 2^Q$, **alors**
    on choisit 2^Q sommets de l'espace de recherche : $externes = pfo(p, Q, 1)$,
    si $N_{init} >= 2^{Q+1}$, **alors**
        on complète par 2^Q sommets d'un hypercube interne :
        $internes = externes/3$,
        et par $N_{init} - 2^{Q+1}$ points pseudo-aléatoires,
    sinon
        on complète par $N_{init} - 2^Q$ points pseudo-aléatoires.
    fin du si
sinon si $N_{init} >= 2^{Q-1}$, **alors**
    on choisit 2^{Q-1} sommets de l'espace de recherche : $externes = pfo(p, Q-1, 1)$,
    si $N_{init} >= 2^Q$, **alors**
        on complète par 2^{Q-1} sommets d'un hypercube interne :
        $internes = pfo(p, Q-1, -1)/3$,
        et par $N_{init} - 2^Q$ points pseudo-aléatoires,
    sinon
        on complète par $N_{init} - 2^{Q-1}$ points pseudo-aléatoires.
    fin du si
sinon
    si on souhaite utiliser des plans d'expériences, **alors**
        il faut augmenter N_{init},
    sinon
        on se contente de tirer N_{init} points pseudo-aléatoires.
    fin du si
fin du si

FIG. 2.6 – Algorithme pour le choix de points bien répartis

Chapitre 3

Optimisation pour l'apprentissage

Dans le cadre de l'approximation de fonctions, il existe plusieurs méthodes :
- l'approche linéaire, où les fonctions de base sont fixées à l'avance, d'une manière indépendente du phénomène à approcher (ex. : polynômes, splines, ...),
- l'approche non linéaire, où les fonctions de base sont construites en fonction des données à apprendre (ex. : réseaux de neurones).

Il convient de citer une approche intermédiaire : une base hilbertienne de L^2 est construite à l'avance (Fourier, ondelettes), mais on ne considère qu'un sous-ensemble de cette base. On retrouve alors
- une approche linéaire, dans le cas où la série est tronquée à un ordre fixé,
- une approche non-linéaire, si on écarte les fonctions de base associées aux petits coefficients.

Nous avons choisi d'utiliser les réseaux de neurones [4], et nous allons montrer les avantages de cette méthode.

Les réseaux de neurones sont des systèmes artificiels capables de simuler certaines capacités des systèmes naturels [72]. Le cerveau, par exemple, effectue des calculs d'une façon complètement non conventionnelle et complexe. Les réseaux de neurones doivent leur efficacité, d'une part, à leur structure parallèle et, d'autre part, à leur capacité d'apprentissage et de généralisation.

Les réseaux de neurones [16] ont été beaucoup étudiés et utilisés dans divers domaines depuis des années : la reconnaissance de formes, la détection d'anomalies, la prédiction de données, ...

L'utilisation des réseaux de neurones offre, entre autres [33] la possibilité d'approcher des phénomènes réels complexes et non linéaires.

Dans de nombreux cas, en particulier dans les contextes industriels, le nombre de neurones peut devenir énorme. En conséquence, la taille du problème, et donc les temps de calcul, grandissent démesurément. Dans ce chapitre, nous décrivons une technique d'apprentissage "zéro-memoire".

3.1 Les neurones biologiques

Le neurone biologique est la pierre élémentaire du cerveau. Il existe environ 10 000 types différents de neurones [74]. On compte au total quelques 10^{11} neurones, chacun pouvant recevoir, par l'intermédiaire de ses dendrites (Fig. 3.1), les informations provenant d'environ 200 000 entrées. Le cerveau est formé par un ensemble de neurones, connectés entre eux par des liens appelés synapses. Les poids de ces liaisons conditionnent le mécanisme de mémorisation et d'oubli. Le cerveau humain reçoit des signaux d'entrée de plusieurs sources ; les signaux sont traités pour créer une réponse. Le cerveau a des millions de neurones qui sont interconnectés pour élaborer des "Réseaux de Neurones". Ces réseaux exécutent les millions d'instructions nécessaires pour avoir une vie normale. Deux éléments du neurone biologique sont particulièrement intéressants pour nous : les dendrites et les synapses.

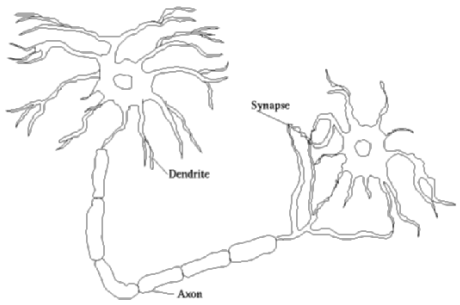


Fig. 3.1 – Un neurone biologique

Les dendrites sont des extensions du neurone qui lui permettent de se connecter à d'autres neurones, tandis que les synapses sont des portes qui acceptent des connexions provenant d'autres neurones. Un neurone biologique peut donc à la fois se connecter à d'autres neurones comme accepter des connexions en provenance d'autres neurones. Ainsi nous avons les bases d'un réseau. Le processus électro-chimique de transfert de l'information entre les neurones est complexe. Le signal se propage le long des axones jusqu'aux synapses. La synapse a la particularité de pouvoir moduler la diffusion des neuro-médiateurs et peut avoir un effet aussi bien excitateur qu'inhibiteur. Ainsi un faible influx nerveux peut engendrer la transmission de l'information. Les dendrites ne modifient pas le signal reçu en entrée. Si le signal résultant est suffisant, il y a création d'un signal à transmettre. Un neurone naturel reçoit des neurones voisins une certaine quantité d'information sous forme d'impulsions électriques par l'intermédiaire de ses dendrites. L'information, une fois traitée par le neurone, est transmise à d'autres neurones par l'intermédiaire des synapses. Cette information n'est transmise qu'à partir d'un certain seuil.

3.2 Structure d'un neurone artificiel

Le concept de réseau de neurones est inspiré du comportement du cerveau humain. Le réseau artificiel est constitué d'un ensemble de cellules appelées neurones. La figure 3.2 montre la structure basique d'un neurone. On peut identifier les éléments suiavnts [33] :

- un ensemble de e entrées $(X_1, X_2, ..., X_e)$ qui représentent les paramètres indépendents du problème (signaux d'entrée),
- un ensemble de e poids synaptiques (w_i) (un poids w_i représente la connexion entre l'entrée x_i et le neurone),
- un biais b,
- un opérateur de sommation (les entrées sont pondérés par les poids),
- une fonction d'activation.

A chaque neurone, on associe une valeur réelle z, appelée état du neurone, qui est calculée au moyen de la formule

$$z = f\left(\sum_{i=1}^{e} W_i X_i + b\right).$$

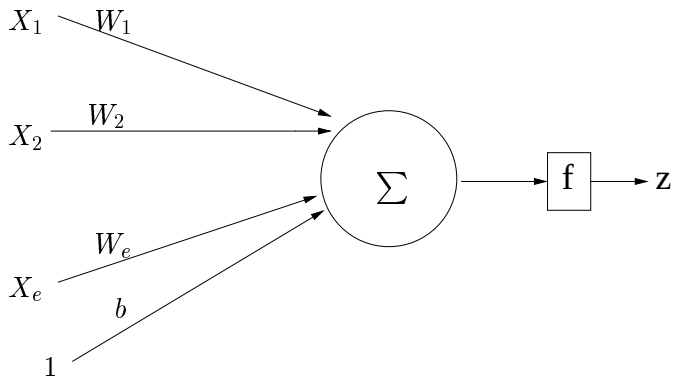


FIG. 3.2 – La structure d'un neurone artificiel

3.3 Architectures neuronales

Un réseau est constitué par un ensemble de neurones interconnectés par des poids et organisé comme une succession de couches. Un réseau a au moins deux couches (une couche d'entrées et une couche de sorties) [40]. Les neurones d'entrée sont les valeurs des paramètres indépendants du problème et la couche de sortie est constituée par les paramètres dépendants. Les neurones qui se trouvent entre la couche d'entrée et la couche de sortie sont appelés neurones cachés et l'ensemble de ces neurones forme la (ou les) couche(s) cachée(s).

La façon dont sont disposés les neurones dans un réseau et la manière dont ils sont connectés entre eux donnent lieu à différents types d'architectures. Nous en mentionnons ici trois genres différents.

Réseau à deux couches

Dans ce type de réseau, il n'existe aucune couche cachée. Les neurones de la couche d'entrée sont directement connectés (via les poids synaptiques) avec les neurones de la couche de sortie. Ce genre de réseau est essentiellement utilisé dans des problèmes qui sont linéairement séparables tels que les problèmes de mémoire associative.

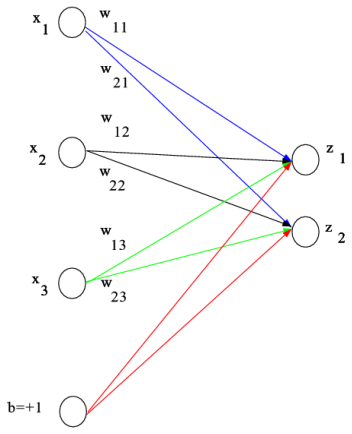


FIG. 3.3 – Structure d'un réseau monocouche

Il est connu [33, 44] qu'un réseau se limitant à deux couches (une couche pour les cellules d'entrée et une couche pour les cellules de sortie) peut se révéler insuffisant pour l'apprentissage de fonctions simples telles que le *ou exclusif*

$$\begin{array}{rccc} g: & \{0,1\}\times\{0,1\} & \longrightarrow & \{0,1\} \\ & (x_1,x_2) & \longmapsto & x_1+x_2-2x_1x_2. \end{array}$$

En effet, un réseau à deux couches ne peut effectuer qu'une séparation linéaire. Cela s'explique par le fait que (si f est monotone) l'ensemble $X = \{x \in \mathbb{R}^e : f(\sum w_j x_j) \leq a\}$ est le demi-espace délimité par l'hyperplan $\sum w_j x_j = f^{-1}(a)$. Par ailleurs, g prend deux valeurs 1 et 0. On peut voir facilement que l'on ne peut pas construire une droite qui sépare les deux ensembles $g^{-1}(1)$ et $g^{-1}(0)$: les deux enveloppes convexes de ces deux ensembles ne sont pas disjointes.

Réseau avec couche(s) cachée(s)

Ce type de réseau est une généralisation du précédent. Il a une ou plusieurs couches dites cachées. Les neurones qui appartiennent à ces couches s'appellent *neurones cachés*. Chaque neurone est connecté à tous les neurones de la couche suivante (il n'y a pas de cycles). L'information circule de l'entrée du réseau vers sa sortie.

Réseaux récurrents

C'est un réseau dans lequel un neurone peut être connecté avec lui-même ou avec des neurones des couches précédentes [2].

3.4 Evaluation d'un réseau avec couche(s) cachée(s)

Chaque neurone dans le réseau reçoit en entrée un ensemble de signaux, leur applique l'opérateur de sommation pondérée et transmet le résultat aux neurones suivants via la

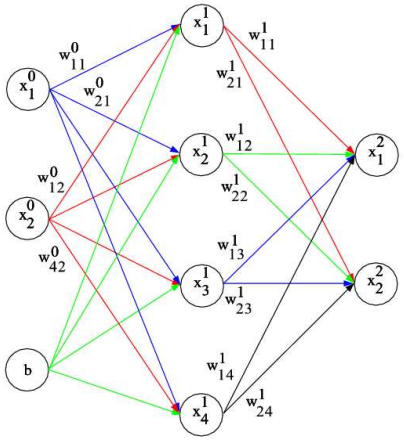


FIG. 3.4 – Structure d'un réseau multicouches

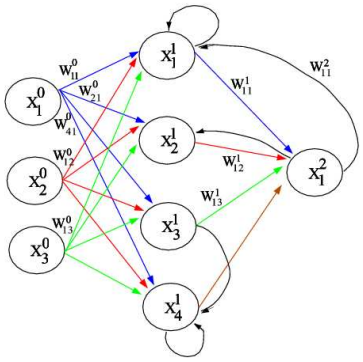


FIG. 3.5 – Structure d'un réseau récurrent

fonction d'activation.

L'état des neurones qui appartiennent à la première couche cachée peut s'écrire

$$z = f\left(\sum_{j=1}^{e} w_j x_j + b\right) \tag{3.1}$$

ou, de façon vectorielle,

$$z = f(w^T x + b) \tag{3.2}$$

L'état des autres neurones est décrit par :

$$z = f\left(\sum_{j=1}^{m} w_j x_j\right) \tag{3.3}$$

ou, de façon vectorielle,

$$z = f(w^T x) \tag{3.4}$$

où $w = [w_1, w_2, ..., w_e]^T$ représente le vecteur des connexions entre neurones, $x = [x_1, x_2, ..., x_e]^T$ l'ensemble des signaux d'entrée et f représente la fonction d'activation qui définit l'état du neurone. La fonction d'activation la plus utilisée dans le domaine des réseaux de neurones est la *fonction sigmoide* définie par

$$f(x) = \frac{1}{1 + e^{-(x-\theta)/\tau}}. \tag{3.5}$$

et représentée sur la figure 3.6 pour différentes valeurs des paramètres τ et θ.

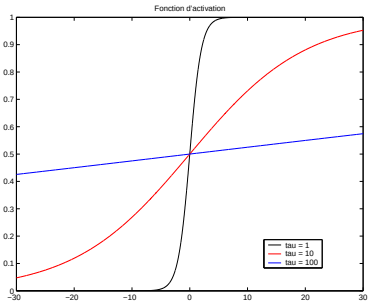


FIG. 3.6 – Fonction sigmoide pour $\theta = 0$ ($\tau = 1$ en noir, $\tau = 10$ en rouge, $\tau = 100$ en bleu)

3.4.1 La solution retenue

D'après un résultat dû à Kolmogorov [37, 38, 39], on peut approcher n'importe quelle fonction régulière avec un réseau à trois couches. (Naturellement, la taille de la couche centrale peut augmenter rapidement avec la précision de l'approximation.) Nous avons donc choisi de nous limiter à de tels réseaux.

Dans le premier étage (entre la couche d'entrée et la couche cachée), nous utilisons la fonction d'activation suivante

$$f(x) = \begin{cases} \dfrac{1}{1 + e^{-x/\tau}} & \text{si } |x| \leq \mu, \\ (x - \mu)\left(\dfrac{e^{-\mu/\tau}}{\tau}\right)\left(\dfrac{1}{1 + e^{-\mu/\tau}}\right)^2 + \left(\dfrac{1}{1 + e^{-\mu/\tau}}\right) & \text{si } x > \mu, \\ (x + \mu)\left(\dfrac{e^{\mu/\tau}}{\tau}\right)\left(\dfrac{1}{1 + e^{\mu/\tau}}\right)^2 + \left(\dfrac{1}{1 + e^{\mu/\tau}}\right) & \text{si } x < -\mu, \end{cases} \tag{3.6}$$

où $\mu = 3\tau$ et $\theta = 0$. Il s'agit simplement de la fonction sigmoïde habituelle modifiée de telle sorte que les poids ne deviennent pas trop grands (cf. figure 3.7). Ce premier étage nous sert à construire une base de fonctions non orthogonale.

Dans le deuxième étage, nous utilisons une fonction d'activation linéaire qui nous permet de réaliser l'approximation de la sortie souhaitée par les fonctions de base créées dans le premier étage.

La structure des réseaux que nous avons choisie est illustrée par la figure 3.8.

Le nombre optimal de neurones dans la couche cachée, est difficile à déterminer. C'est pendant la phase d'apprentissage (cf. 3.5 page 37) que l'on peut se rendre compte de l'aptitude du réseau à "apprendre".

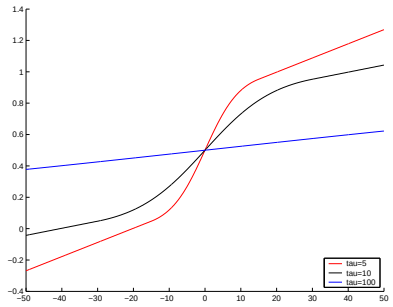


FIG. 3.7 – Fonction sigmoide modifiée pour $\theta = 0$ ($\tau = 5$ en rouge, $\tau = 10$ en noir, $\tau = 100$ en bleu)

3.5 L'apprentissage

Le processus d'apprentissage d'un réseau consiste à ajuster les poids de connexion entre les neurones.

Les différents types d'apprentissage

On peut classer les algorithmes d'apprentissage en deux catégories (cf. figure 3.9 issue de [70]) : supervisés et non supervisés. Dans les algorithmes non supervisés, on ne connaît pas la sortie que doit approcher le réseau (sortie souhaitée). Le réseau s'organise alors en regroupant selon les mêmes caractéristiques les différents signaux d'entrée. Dans les algorithmes supervisés, on connaît la sortie souhaitée et on peut faire la correction de la sortie du réseau par rapport à la sortie ciblée. On observe la sortie donnée par le réseau et on calcule la différence entre elle et la sortie souhaitée. Ensuite, les poids des connexions sont modifiés afin d'atténuer cette différence. L'apprentissage supervisé peut lui-même être de 2 types : l'apprentissage par renfort et l'apprentissage par correction. Dans le premier, l'information est du type *booléenne*. Il s'agit donc de classification. Dans l'apprentissage par correction, on connaît l'ordre de grandeur de l'erreur, et on peut modifier les poids de connexion de telle sorte que cette erreur soit la plus petite possible.

L'apprentissage : un problème d'optimisation

Notons R la réponse du réseau à trois couches considéré, e le nombre d'entrées (sans compter le biais), m le nombre de neurones cachés, s le nombre de sorties et p le nombre de poids de connexion à ajuster. En se référant à la figure 3.4, il est facile d'établir que $p = (e + 1 + s)m$.

La réponse R dépend de l'entrée du système x (l'état des cellules d'entrée) et des poids de connexions $\mathbf{W} \in \mathbb{R}^p$

$$\begin{array}{rccc} R: & \mathbb{R}^e \times \mathbb{R}^p & \longrightarrow & \mathbb{R}^s \\ & (x, \mathbf{W}) & \longrightarrow & R(x, \mathbf{W}). \end{array}$$

Ainsi on peut écrire $\mathbf{W} = \begin{pmatrix} W^0 \\ W^1 \end{pmatrix}$, $W^0 \in \mathbb{R}^{m\times(e+1)}$ et $W^1 \in \mathbb{R}^{s\times m}$. Les deux vecteurs

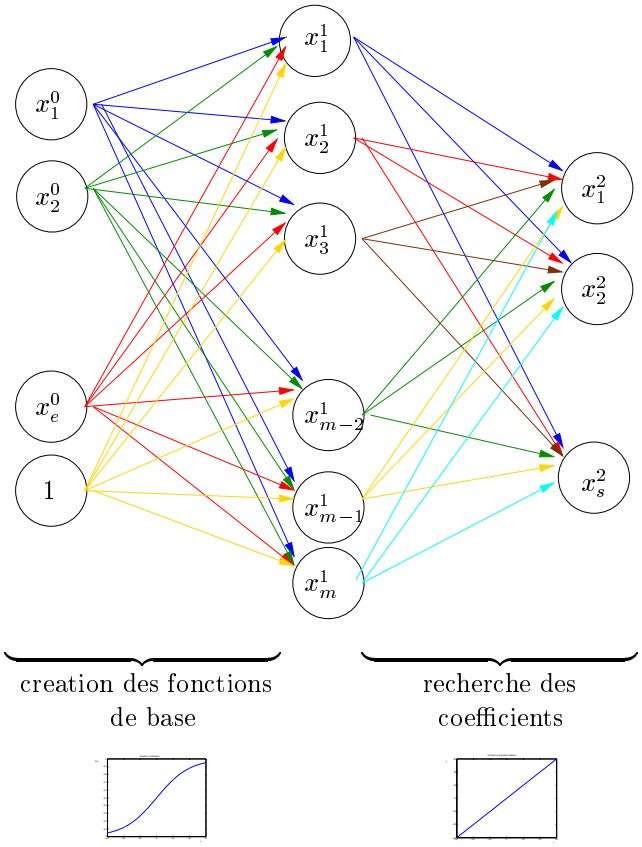


FIG. 3.8 – Structure du réseau et fonctions d'activation retenues

W^0 et W^1 ne jouent pas les mêmes rôles. Le vecteur W^0 permet la définition des fonctions de base et W^1 est le vecteur des coefficients dans cette base. Cette remarque jouera un rôle fondamental dans la régularisation du problème d'apprentissage.

La phase d'apprentissage (supervisé) consiste à minimiser l'écart, au sens des moindres carrés, entre la sortie souhaitée $G(x)$ et la sortie du réseau $R(x, \mathbf{W})$. Il s'agit donc de résoudre le problème

$$\min_{\mathbf{W}\in\mathbb{R}^p} J(\mathbf{W}) = \frac{1}{2}\sum_{x\in\Omega} \| R(x,\mathbf{W}) - G(x) \|^2, \tag{3.7}$$

dans lequel Ω est l'ensemble des couples d'apprentissage. Un élément $(x, G(x))$, $x \in \Omega$ est appelé couple d'apprentissage.

Le fait d'avoir choisi une fonction d'activation f différentiable permet d'utiliser les méthodes classiques d'optimisation. Le gradient de J est généralement calculé par la méthode classique de "rétropropagation du gradient".

L'efficacité de l'apprentissage dépend de la structure du réseau (nombre de neurones cachés). Si le nombre de neurones est insuffisant, l'apprentissage se révèle impossible. Au contraire, si le nombre de neurones dans la couche cachée est trop grand, le réseau apprend

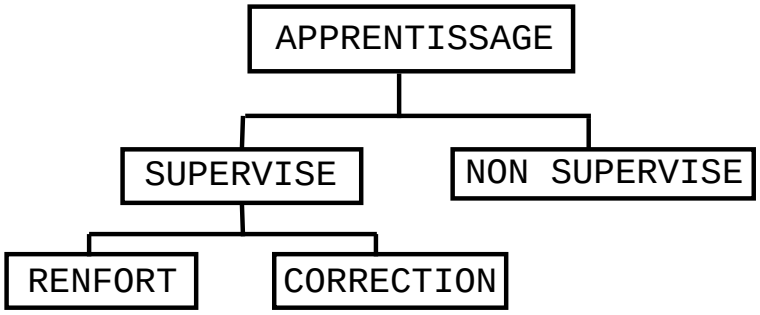


FIG. 3.9 – Les différents types d'algorithmes d'apprentissage

l'information par cœur. Dans ce cas, le réseau restitue correctement les réponses relatives aux couples déjà appris, mais s'avère incapable de donner une réponse correcte pour un nouveau cas [46]. On dit que le réseau ne généralise pas. Cela rappelle les problèmes classiques d'interpolation : les polynômes de bas degré ne donnent pas une bonne approximation et les polynômes de degré élevé oscillent.

Généralisation

L'ensemble de couples Ω est utilisé pour le processus d'apprentissage. Cet ensemble est généralement scindé en deux lots. Une fois que le réseau a appris l'ensemble des couples d'apprentissage du premier lot (en ajustant ses poids), on vérifie sur l'autre lot que la réponse du réseau n'est pas trop éloignée de la sortie souhaitée. Ce processus (capacité de produire des sorties correctes pour des couples non appris) s'appelle la généralisation [5]. Il est un bon indicateur de la qualité de l'apprentissage. En effet, si on parvient à bien apprendre les couples du premier lot, sans être capable de généraliser, on peut conclure que l'information fournie par le réseau n'est pas fiable.

En pratique, il est délicat de trouver le bon compromis entre apprentissage satisfaisant et bonne généralisation.

3.6 Une méthode d'apprentissage zéro mémoire

Afin de simplifier l'exposé de la méthode, nous considérons, dans cette section, que le nombre de sorties dans le réseau est $s = 1$.

3.6.1 Gauss-Newton ou Levenberg-Marquardt

Nous utilisons la méthode de Gauss-Newton pour résoudre le problème d'optimisation (3.7). Pour simplifier les écritures, nous notons

$$\begin{array}{rccl} E: & \mathbb{R}^p & \longrightarrow & \mathbb{R}^c \\ & \mathbf{W} & \longmapsto & \Big(R(x_1,\mathbf{W})-G(x_1), R(x_2,\mathbf{W})-G(x_2), ..., R(x_p,\mathbf{W})-G(x_p)\Big)^T \end{array}$$

où c est le nombre de couples d'apprentissage contenus dans Ω. Ainsi le problème (3.7) s'écrit

$$\min_{\mathbf{W}\in\mathbb{R}^p} J(\mathbf{W}) = \frac{1}{2} \parallel E(\mathbf{W}) \parallel^2 . \tag{3.8}$$

Notons encore DE la jacobienne de E. C'est une matrice à c lignes et p colonnes. La première façon de voir la méthode de Gauss-Newton est d'appliquer la méthode de Newton au problème (3.8) en négligeant les termes où interviennent les derivées de E d'ordre supérieur.

La direction de Newton d est la solution du système linéaire

$$\nabla^2 J(\mathbf{W}_k)d = -\nabla J(\mathbf{W}_k). \tag{3.9}$$

Dans notre cas, on a :

$$\nabla J(\mathbf{W}) = DE(\mathbf{W})^T \; E(\mathbf{W})$$

et

$$\nabla^2 J(\mathbf{W}) = DE(\mathbf{W})^T DE(\mathbf{W}) + \sum_{i=1}^{c} E_i(\mathbf{W})\nabla^2 E_i(\mathbf{W}).$$

Comme le second terme de $\nabla^2 J(\mathbf{W})$ peut être difficile à calculer, on le néglige. Cette approximation se justifie, au voisinage de la solution, lorsque E devient petit. On aboutit alors à la méthode, dite de Gauss-Newton,

$$DE(\mathbf{W}_k)^T DE(\mathbf{W}_k)d = -DE(\mathbf{W}_k)^T E(\mathbf{W}_k). \tag{3.10}$$

Dans la majorité des cas, le nombre c des couples d'apprentissage est bien plus petit que le nombre total p de poids dans le réseau. De ce fait, la matrice $DE(\mathbf{W}_k)^T DE(\mathbf{W}_k)$ peut être singulière. On utilise alors l'algorithme de Levenberg-Marquardt [42] : le système linéaire (3.10) est transformé en

$$(DE(\mathbf{W})^T DE(\mathbf{W}) + \alpha I)d = -DE(\mathbf{W})^T E(\mathbf{W}) \tag{3.11}$$

où α est un réel strictement positif. La valeur de α ne change pas la solution finale du problème (3.8), mais il permet d'accélérer la convergence et de rendre le système linéaire (3.10) plus stable.

Le système linéaire (3.11) est résolu via la méthode du gradient conjugué. C'est une méthode itérative qui permet de résoudre des systèmes linéaires $Ax = b$ lorsque la matrice A est symétrique définie positive. Developpée en 1952 par Hestenes et Stiefel [9], cette méthode ne fait pas appel à des paramètres définis par l'utilisateur. Elle ne nécessite que la connaissance du produit de la matrice A par un vecteur.

3.6.2 Différentiation automatique

Le produit matrice-vecteur $DE(\mathbf{W})^T DE(\mathbf{W})d$ est ici effectué en exploitant les techniques de différentiation automatique [22, 21, 28, 63]. On commence par calculer $z = DE(\mathbf{W})d$. Ce produit peut être réécrit comme ci-dessous

$$DE(\mathbf{W})d = \lim_{\varepsilon\to 0} \frac{E(\mathbf{W}+\varepsilon d) - E(\mathbf{W})}{\varepsilon} = \partial_\varepsilon E(\mathbf{W}+\varepsilon d)|_{\varepsilon=0}.$$

Il s'agit alors de dériver une fonction vectorielle E par rapport à un seul paramètre [29]. On utilise pour ce faire le mode direct qui consiste à dériver un code de sa première instruction à la dernière.

Il reste ensuite à calculer

$$DE(\mathbf{W})^T z = \sum_{i=1}^{c} DE_i(\mathbf{W}) z_i.$$

Comme la somme des dérivées est égale à la dérivée de la somme, nous avons

$$DE(\mathbf{W})^T z = D(\sum_{i=1}^{c} E_i(\mathbf{W}) z_i) = D(z^T E(\mathbf{W})).$$

Il s'agit donc de dériver une fonction scalaire $z^T E(\mathbf{W})$ par rapport à un grand vecteur $\mathbf{W}$. La dérivation en mode inverse est la plus appropriée. Elle consiste grossièrement à dériver de la dernière instruction à la première en utilisant les règles de dérivation des fonctions composées.

L'utilisation de la méthode du gradient conjugué et cette combinaison des modes direct et inverse de la différentiation automatique permettent de faire l'économie de calculs très coûteux tels que

- le calcul de la jacobienne DE,
- le calcul du produit matriciel $DE^T DE$,
- le stockage de cette matrice
- et sa factorisation.

Ce gain est d'autant plus crucial que le problème est de grande taille.

3.6.3 Régularisation

L'estimation d'une fonction g à partir de données en nombre limité peut devenir un problème mal posé [43], c'est-à-dire que la solution obtenue par minimisation de l'écart quadratique moyen existe mais n'est pas unique. L'idée fondamentale de la régularisation est de transformer un problème mal posé en un problème bien posé. Pour ce faire, on impose des contraintes supplémentaires à la fonction [72, 24].

On commence par faire un apprentissage "par cœur" : on augmente la taille de $\mathbf{W}$ (plus exactement le nombre m de cellules cachées) afin d'avoir un résidu proche de zéro après apprentissage. On s'assure ainsi que le réseau est assez riche pour approcher la vraie fonction. Dans ce cas, la généralisation n'est pas assurée : la réponse du réseau pour des données non apprises est peu satisfaisante. De nombreuses techniques de régularisation existent. Nous allons discuter quatre méthodes.

Régularisation de Tikhonov

Pour éviter les oscillations du réseau, on cherche à lisser les fonctions de base en conservant les poids du premier étage petits. La solution régularisée est celle qui minimise la combinaison

pondérée de la norme du résidu et de la norme des poids du premier étage :

$$\frac{1}{2}\Big(\parallel E(\mathbf{W}) \parallel^2 + \beta \parallel \mathbf{W}^0 \parallel^2 \Big) \tag{3.12}$$

où β est le paramètre de régularisation [58] et $\mathbf{W}^0$ les poids de connexion entre la couche d'entrée et la couche cachée. Le système linéaire associé à (3.12) est de la forme :

$$\left(DE(\mathbf{W})^T DE(\mathbf{W}) + \alpha I + \beta \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}\right) \begin{pmatrix} d_0 \\ d_1 \end{pmatrix} = -DE(\mathbf{W})^T E(\mathbf{W}) - \beta \begin{pmatrix} \mathbf{W}^0 \\ 0 \end{pmatrix}. \tag{3.13}$$

Le deuxième étage est laissé sans régularisation.

Avec une grande valeur de β, le processus d'optimisation privilégie la réduction de la norme des poids de connexions W^0 au détriment de la norme du résidu. En conséquence, les fonctions de base construites dans le premier étage sont étirées (cf. figure 3.6 ; de petits poids W^0 correspondent à un τ grand).

Dans notre cas, pour choisir le paramètre β, l'ensemble Ω des couples d'apprentissage est découpé en trois lots : un premier lot L1 composé de 80 % des couples d'apprentissage, un second lot L2 constitué de 12 % des couples d'apprentissage et un troisième lot L3 comprenant le reste des couples.

Le lot L3, sur lequel aucun traitement n'est jamais effectué, sert de lot de validation.

L'apprentissage est alors réalisé en deux phases :

Phase 1 :

a) apprentissage du lot L1 avec $\beta = 0$. Tant que $\parallel E(\mathbf{W}) \parallel \leq tol$, on augmente le nombre de neurones cachés et on réapprend. Ainsi, on augmente le nombre de fonctions de base nécessaires pour bien approcher le phénomène.

b) Une fois trouvé le nombre de neurones cachés approprié, on apprend le lot L1 en choisissant le β qui minimise le résidu sur le lot L2.

Phase 2 : apprentissage des lots L1 et L2 en utilisant le β trouvé dans la phase 1 et en démarrant des poids obtenus à cette même phase.

Les pourcentages donnés n'ont qu'un caractère indicatif.

Régularisation par réduction du nombre de neurones cachés

Cette technique consiste à réduire le nombre de cellules dans la couche cachée pour ameliorer la généralisation. Malheureusement, il arrive souvent que l'on passe d'une situation où l'on apprend sans généraliser à une situation où l'on n'apprend plus. Il n'est pas toujours facile de trouver un compromis.

La technique décrite dans le paragraphe précédent permet d'obtenir, avec un nombre de neurones cachés donné, une valeur optimale du paramètre β.

Si ce nombre de neurones cachés est trop grand, le phénomène de sur-apprentissage peut se produire malgré la régularisation. De plus, le temps de calcul augmente avec le nombre de neurones cachées. Pour ces deux raisons, nous effectuons en pratique de nouveaux tests avec moins de neurones cachés.

Ainsi, en combinant la régularisation de Tikhonov et le choix du nombre de neurones cachés, on parvient à trouver le nombre minimum de neurones cachés qui permet un bon compromis entre temps de calcul et bonne approximation de la fonction.

Régularisation par arrêt prématuré des itérations

Cette méthode est connue sous le nom de "stopped training method" [10, 65] ou "Landweber iteration" [43]. Elle est basée sur l'observation suivante (sur-apprentissage) : trop apprendre à partir d'un nombre limité de couples d'apprentissage peut dégrader la généralisation. En pratique, l'erreur sur les couples du lot L1 décroît à chaque itération puis se stabilise, alors que l'erreur dans le lot L2 passe par un minimum avant de croître. Il faut donc arrêter l'apprentissage autour de ce minimum du résidu du lot L2.

Régularisation par Gauss-Newton

En choisissant un W^0 initial de petite norme, nous initialisons le réseau avec des fonctions de base étirées (régulières). Comme le processus de Gauss-Newton calcule des perturbations de norme minimale [48], les poids de connexion du premier étage W^0 restent petits et les fonctions de base bien régulières.

La combinaison de ces techniques de régularisation permet d'éviter les minima locaux pour l'apprentissage (qui posent sérieux problèmes pour les méthodes classiques).

3.7 Résultats numériques

Dans cette section, nous allons illustrer les possibilités offertes par la méthode décrite ci-dessus sur deux types d'exemple : des exemples synthétiques basés sur l'approximation de fonctions classiques et des cas réels issus de divers domaines tels que l'industrie automobile ou le domaine pétrolier.

Pour chaque résultat numérique, nous indiquerons

- **la structure du réseau** : *nombre de neurones d'entrée e - nombre de neurones cachés m - nombre de neurones de sortie s.*
- **la répartition des couples d'apprentissage (section 3.6.3 page 41)** : *taille du lot d'apprentissage L1 - taille du lot de généralisation L2 - taille du lot de vérification L3.*

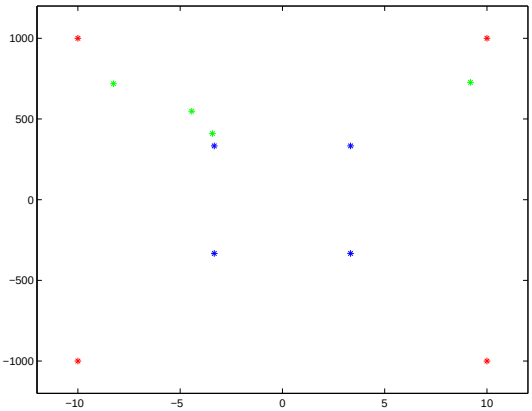
3.7.1 Exemples synthétiques

Nous allons illustrer la capacité de réseaux de neurones à approcher des fonctions classiques [47].

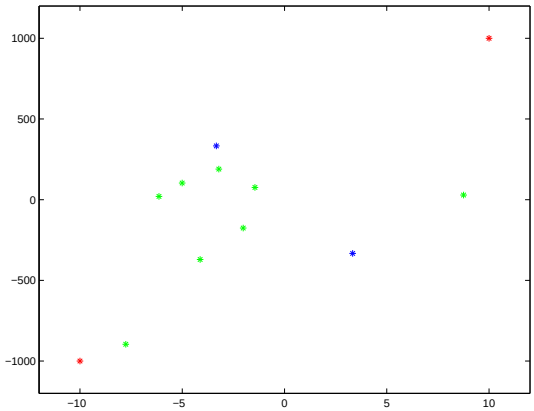
Notons f la fonction à approcher. Elle est définie sur $\mathbb{R}^e$ à valeurs dans $\mathbb{R}^s$. Les couples d'apprentissage s'écrivent alors $(x, f(x))$, où $x \in D \subset \mathbb{R}^e$. Les c valeurs de x peuvent être crées de 4 manières différentes :

- **option=1** : elles sont lues dans un fichier de données.
- **option=2** : elles sont choisies de façon aléatoire dans D.
- **option=3** : elles sont construites en utilisant deux plans d'expériences (section 2.4 page 26) ; un premier plan extérieur et un second intérieur. Le reste des valeurs de x est choisi de façon aléatoire.
- **option=4** : elles sont choisies comme ci-dessus, mais les deux plans d'expériences sont ici croisés.

La figure 3.10 permet d'illustrer la génération de couples au moyen de plans d'expériences. Les étoiles rouges représentent les points du plan factoriel externe, les étoiles bleues symbolisent les points du plan factoriel interne et les étoiles vertes sont les couples tirés aléatoirement.



Points générés par l'option 3



Points générés par l'option 4

FIG. 3.10 – Points générés à l'aide de plans factoriels

Dans de nombreux cas, en particulier dans les exemples réels, la fonction à approcher est très coûteuse à évaluer. On cherche alors à réduire le nombre d'appels à cette fonction. Lorsque le nombre d'entrées e est grand, on ne s'autorise donc pas à se donner autant de couples d'apprentissage que de sommets de l'hypercube (pour $e = 10$, il faudrait construire $2^{10} = 1024$ couples !!!). L'option 4 nous sert essentiellement dans ces cas là (lorsque nous sommes limités par le nombre de couples d'apprentissage).

Ou exclusif

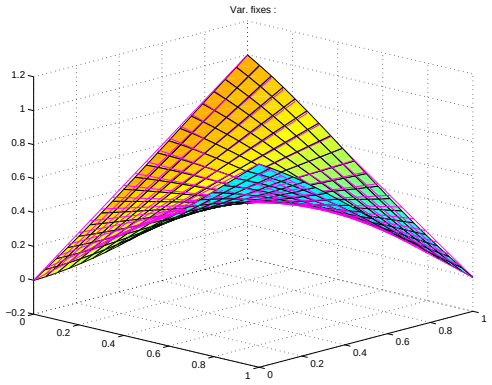
Il s'agit d'une fonction très simple qui n'est pas linéairement séparable. Pour cette raison, elle fait partie des exemples classiques utilisés pour tester les réseaux de neurones.

Données

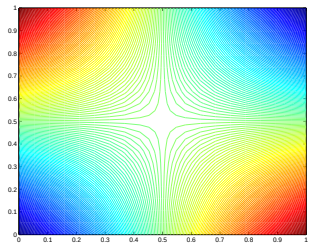
Fonction	$f(x,y) = x^2 + y^2 - 2xy$
Domaine	$D = [0,1] \times [0,1]$
Structure du réseau	3 / 4 / 1
Lots	6 / 2 / 2
Option	3

Apprentissage

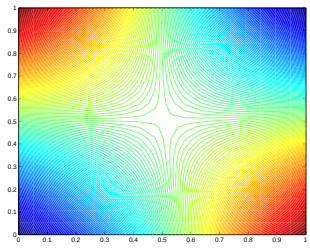
	Réseau avec $\beta = 0$
Nombre d'itérations de GN	36
Nombre d'itérations du GC	257
Erreur d'apprentissage	$2.314\ 10^{-5}$
Erreur de généralisation	0.0079
Erreur de vérification	0.01129



Réseau avec $\beta = 0$



Contours de la fonction,



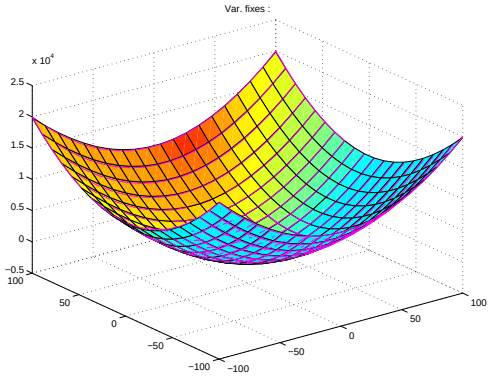
du réseau avec $\beta = 0$

FIG. 3.11 – Apprentissage du "ou exclusif".

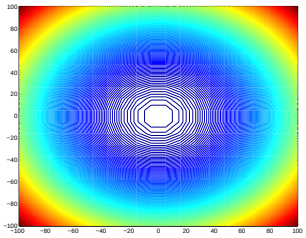
La parabole

Données	
Fonction	$f(x, y) = x^2 + y^2$
Domaine	$D = [-100, 100] \times [-100, 100]$
Structure du réseau	3 / 3 / 1
Lots	20 / 3 / 2
Option	4

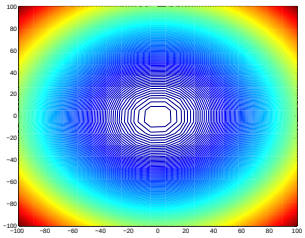
Apprentissage	
	Réseau avec $\beta = 0$
Nombre d'itérations de GN	54
Nombre d'itérations du GC	504
Erreur d'apprentissage	$2.44\ 10^{-4}$
Erreur de généralisation	$4.1\ 10^{-3}$
Erreur de vérification	$3.4\ 10^{-3}$



Réseau avec $\beta = 0$



Contours de la fonction,



du réseau avec $\beta = 0$

FIG. 3.12 – Apprentissage de la parabole

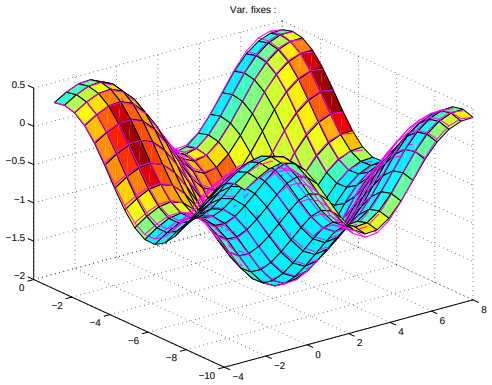
Fonction sinus

Données

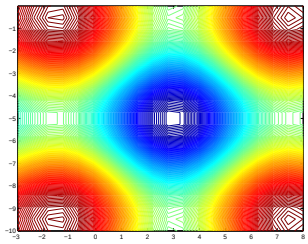
Fonction	$f(x,y) = -\dfrac{\sin(x-3)}{(x-3)} - \dfrac{\sin(y+5)}{(y+5)}$
Domaine	$D = [-3,8] \times [-10,0]$
Structure du réseau	3 / 10 / 1
Lots	22 / 3 / 2
Option	3

Apprentissage

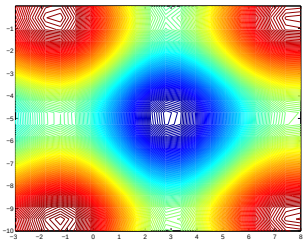
	Réseau avec $\beta = 0$
Nombre d'itérations de GN	93
Nombre d'itérations du GC	2042
Erreur d'apprentissage	$5.21\ 10^{-5}$
Erreur de généralisation	0.010
Erreur de vérification	0.007



Réseau avec $\beta = 0$



Contours de la fonction,



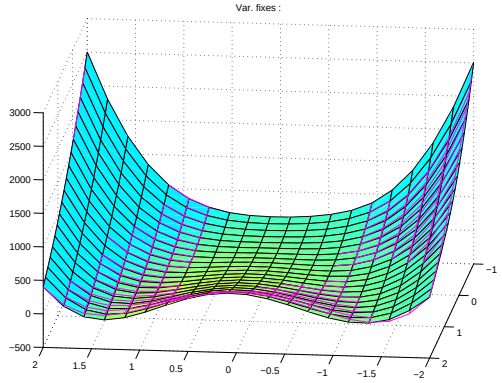
du réseau avec $\beta = 0$

FIG. 3.13 – Apprentissage du sinus

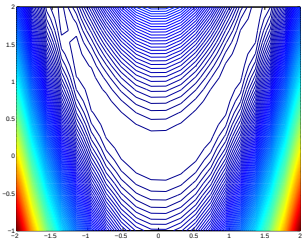
Fonction Rosenbrock

Données	
Fonction	$f(x,y) = 100(x^2+y^2)^2 + (1-x)^2$
Domaine	$D = [-2,2] \times [-2,2]$
Structure du réseau	3 / 10 / 1
Lots	25 / 3 / 2
Option	3

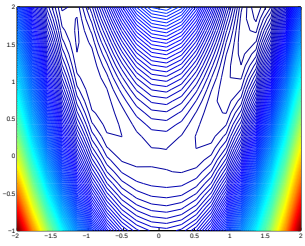
Apprentissage	**Réseau avec $\beta = 0$**
Nombre d'itérations de GN	96
Nombre d'itérations du GC	2366
Erreur d'apprentissage	$1.87\ 10^{-5}$
Erreur de généralisation	0.0079
Erreur de vérification	0.0038



Réseau avec $\beta = 0$



Contours de la fonction,



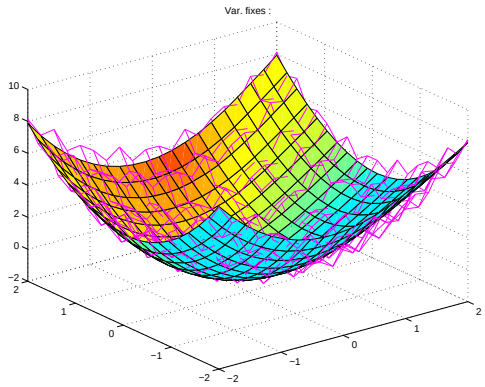
du réseau avec $\beta = 0$

FIG. 3.14 – Apprentissage de la fonction de Rosenbrock

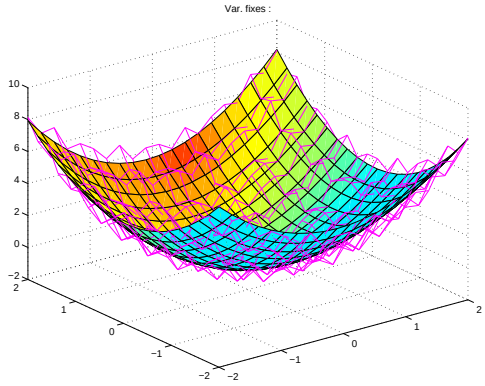
Fonction Rastrigin

	Données
Fonction	$f(x,y) = x^2 + y^2 - cos(18x) - cos(18y)$
Domaine	$D = [-2,2] \times [-2,2]$
Structure du réseau	3 / 3 / 1
Lots	22 / 3 / 2
Option	1

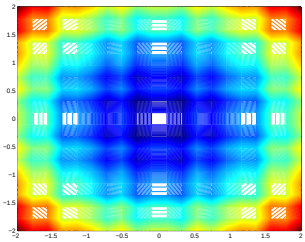
	Apprentissage	
	Réseau avec $\beta = 0$	**Réseau final**
Nombre d'itérations de GN	100	100
Nombre d'itérations du GC	1040	1200
Erreur d'apprentissage	0.031	0.045
Erreur de généralisation	0.075	
Erreur de vérification	0.0278	0.0198



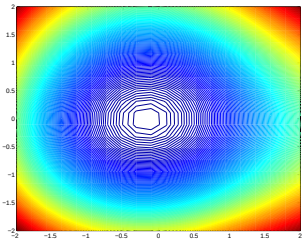
Réseau avec $\beta = 0$



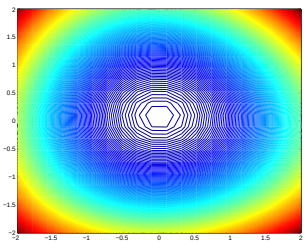
Réseau final



Contours de la fonction,



du réseau avec $\beta = 0$



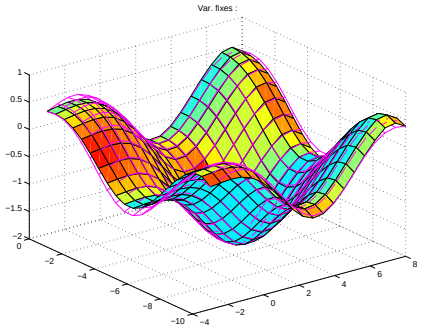
et du réseau final

FIG. 3.15 – Apprentissage de la fonction de Rastrigin

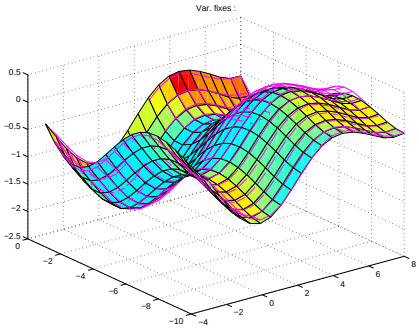
Fonction sinus - sinus décalée

Données	
Nb. de sorties	2
Fonction	$f_1(x,y) = -\frac{\sin(x-3)}{(x-3)} - \frac{\sin(y+1)}{(y+1)}$
	$f_2(x,y) = -\frac{\sin(x+5)}{(x+5)} - \frac{\sin(y+2)}{(y+2)}$
Domaine	$D = [-3,8] \times [-10,0]$
Structure du réseau	3 / 10 / 2
Lots	22 / 3 / 2
Option	3

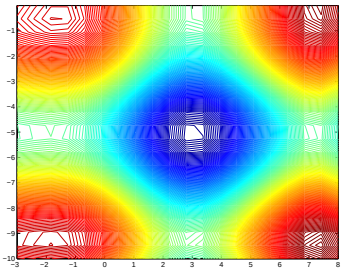
Apprentissage	Réseau avec $\beta = 0$
Nombre d'itérations de GN	100
Nombre d'itérations du GC	2297
Erreur d'apprentissage	0.0013
Erreur de généralisation	0.0260
Erreur de vérification	0.0304



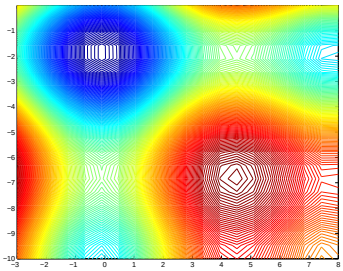
Réseau avec $\beta = 0$ pour f_1



Réseau avec $\beta = 0$ pour f_2



Contours du réseau f_1,



du réseau f_2

FIG. 3.16 – Apprentissage de la fonction sinus - sinus décalé

Fonction somme de sinus et sinus décalé

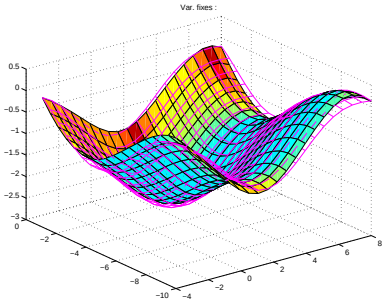
La sotie de cette fonction est la somme des deux fonctions de la section précedente : sinus et sinus décalé.

Données

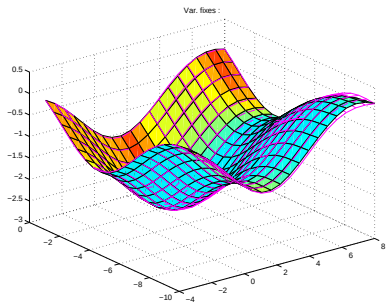
Nb. de sorties	1
Fonction	$f(x,y) = -\frac{\sin(x-3)}{(x-3)} - \frac{\sin(y+1)}{(y+1)} - \frac{\sin(x+5)}{(x+5)} - \frac{\sin(y+2)}{(y+2)}$
Domaine	$D = [-3,8] \times [-10,0]$
Structure du réseau	3 / 10 / 1
Lots	22 / 3 / 2
Option	3

Apprentissage

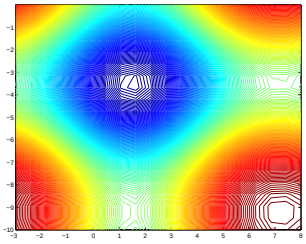
	Réseau avec $\beta = 0$	**Réseau final**
Nombre d'itérations de GN	100	60
Nombre d'itérations du GC	533	1800
Erreur d'apprentissage	0.049	$4.9916e-05$
Erreur de généralisation	0.04555	
Erreur de vérification	0.08079	0.01326



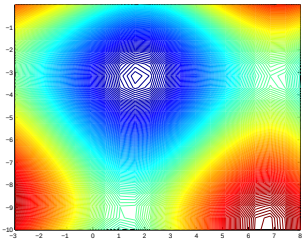
Réseau avec $\beta = 0$



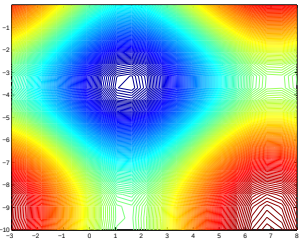
Réseau final



Contours de la fonction,



du réseau avec $\beta = 0$



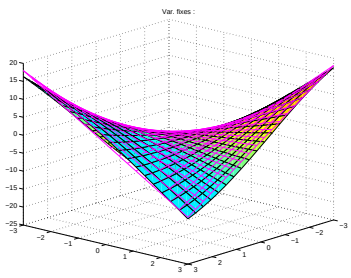
et du réseau final

FIG. 3.17 – Apprentissage de la fonction somme de sinus et sinus décalé

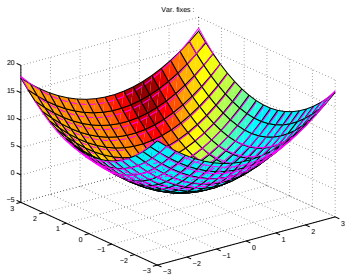
Fonction Ou exclusif - parabole - parabole décalée

Données	
Nb. de sorties	3
Fonction	$f_1(x,y) = x^2 + y^2 - 2xy$ $f_2(x,y) = x^2 + y^2$ $f_2(x,y) = (x-2)^2 + (y-2)^2$
Domaine	$D = [-3,3] \times [-3,3]$
Structure du réseau	3 / 5 / 1
Lots	22 / 3 / 2
Option	3

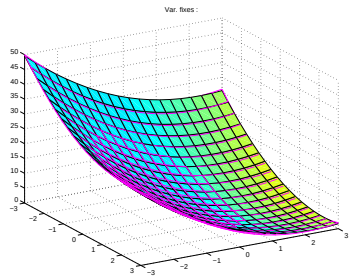
Apprentissage		
	Réseau avec $\beta = 0$	**Réseau final**
Nombre d'itérations de GN	40	40
Nombre d'itérations du GC	817	1175
Erreur d'apprentissage	0.09298	0.00196
Erreur de généralisation	0.088	
Erreur de vérification	0.0658	0.0139



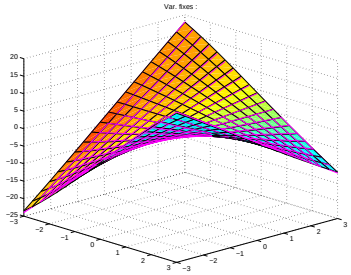
Réseau initial pour f_1



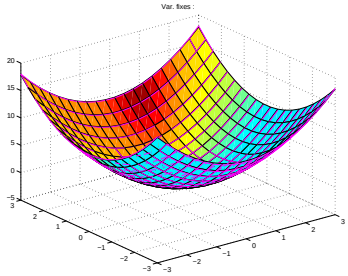
Réseau initial pour f_2



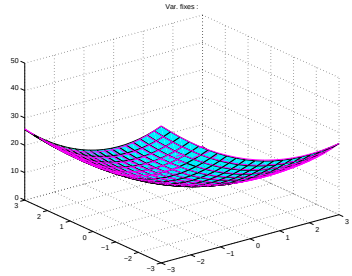
Réseau initial pour f_3



Réseau final pour f_1

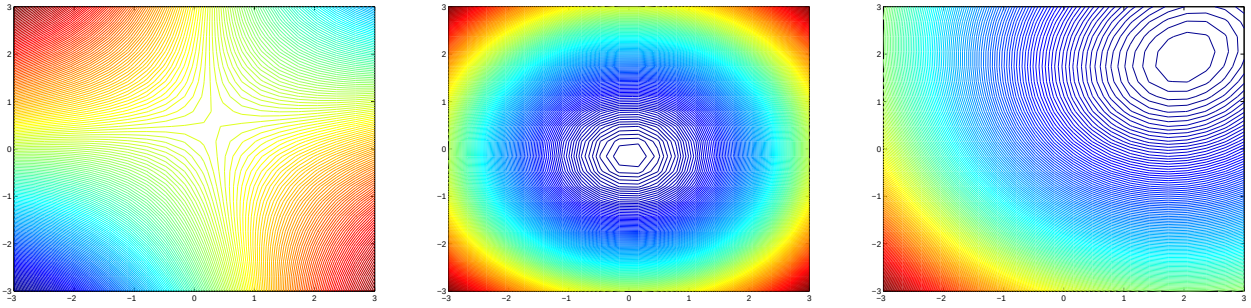


Réseau final pour f_2

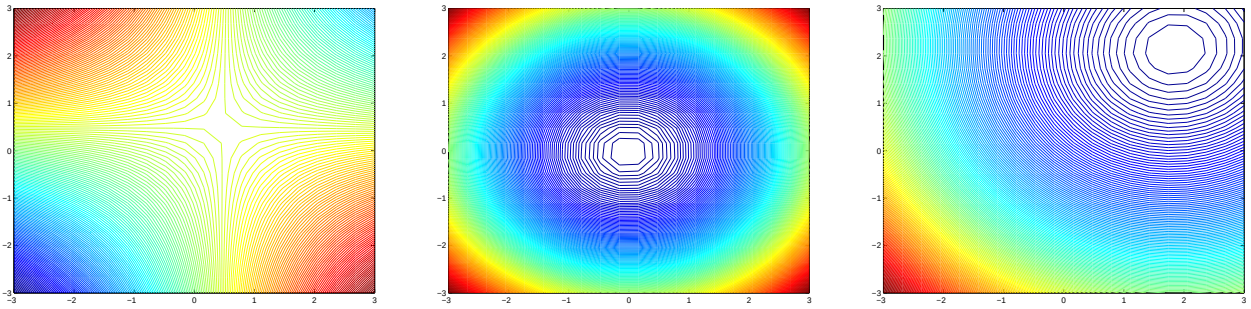


Réseau final pour f_3

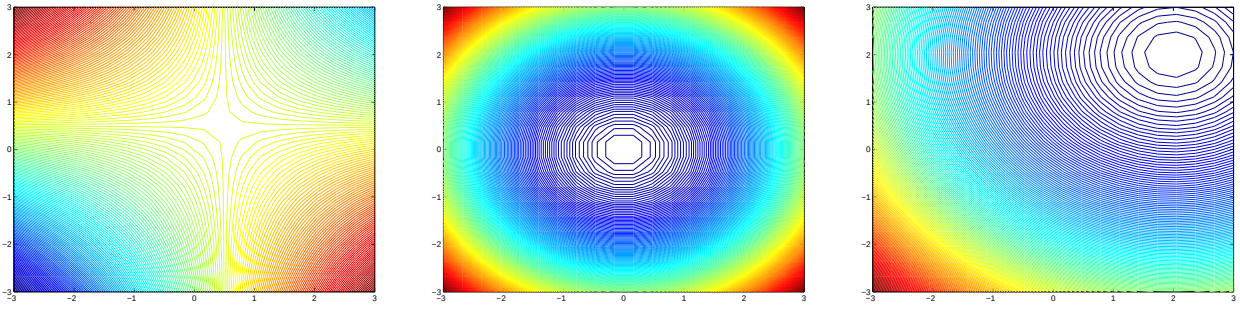
FIG. 3.18 – Fonctions xor - parabole - parabole décalée : réseaux



Contour du réseau initial pour f_1 et pour f_2 et pour f_3



Contour du réseau final pour f_1 et pour f_2 et pour f_3



Contour de la fonction pour f_1 et pour f_2 et pour f_3

FIG. 3.19 – Fonctions xor - parabole - parabole décalée : contours

Fonction somme du Ou exclusif, de la parabole et de la parabole décalée

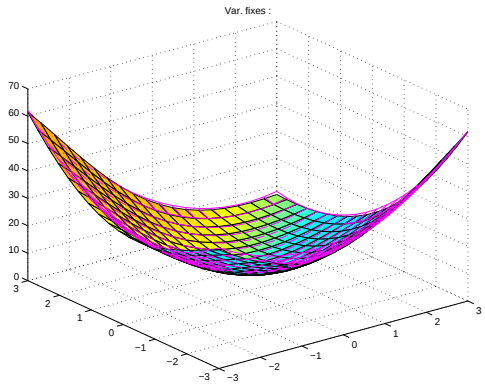
La sortie de cette fonction est la somme des des trois fonctions de la section précedente : ou exclusif, parabole et parabole décalée.

Données

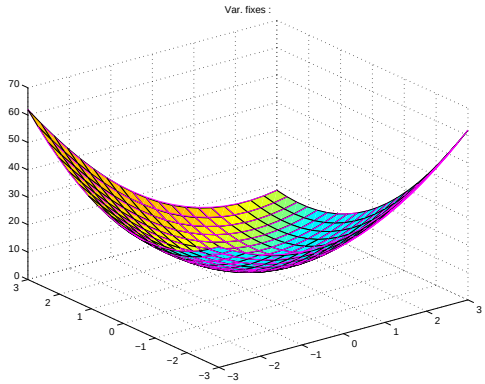
Nb. de sorties	1
Fonction	$f(x,y) = 2x^2 + 2y^2 - 2xy + (x-2)^2 + (y-2)^2$
Domaine	$D = [-3,3] \times [-3,3]$
Structure du réseau	3 / 5 / 1
Lots	22 / 3 / 2
Option	3

Apprentissage

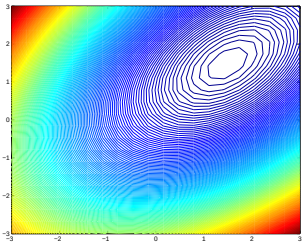
	Réseau avec $\beta = 0$	**Réseau final**
Nombre d'itérations de GN	30	60
Nombre d'itérations du GC	765	1800
Erreur d'apprentissage	0.00322	$8.64772e-05$
Erreur de généralisation	0.006580	
Erreur de vérification	0.014723	0.0029756



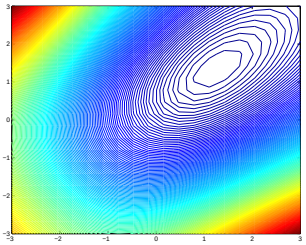
Réseau avec $\beta = 0$



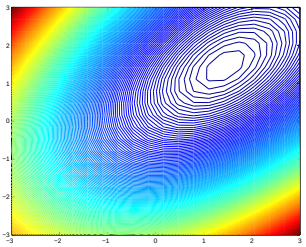
Réseau final



Contours de la fonction,



du réseau avec $\beta = 0$



et du réseau final

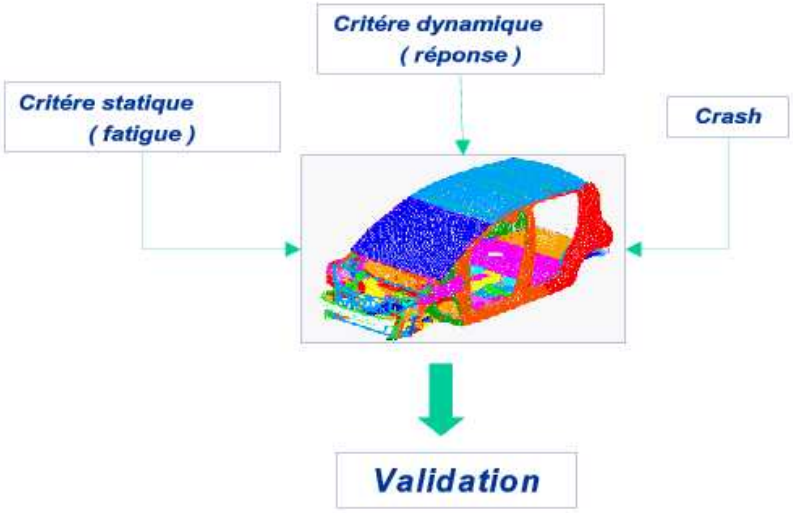
FIG. 3.20 – Apprentissage de la fonction somme de xor, parabole et parabole décalée

3.7.2 Exemples réels

CRASH

On étudie un modèle de choc frontal d'un véhicule sur une barrière déformable. Il s'agit de déterminer l'influence de certains paramètres sur le comportement du véhicule :

x_1 : raideur du "bumper" de la barrière,
x_2 : raideur du corps de la barrière,
x_3 : position horizontale de la barrière,
x_4 : position verticale de la barrière,
x_5 : angle d'impact du véhicle sur la barrière,
x_6 : vitesse d'impact du véhicule,
x_7 : assiette du véhicule.



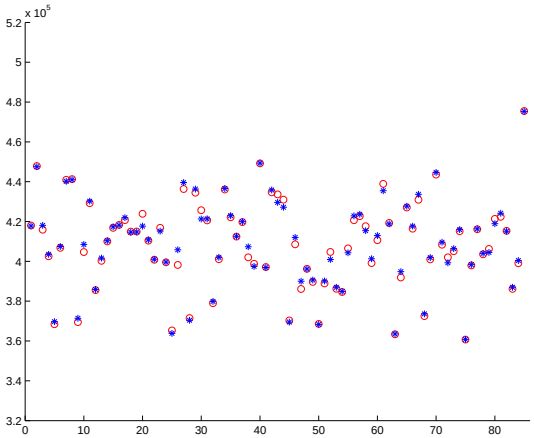
La sortie observée est y, l'effort sur le mur en fond de barrière.

Comme l'évaluation de y en fonction des x_i est très coûteuse, on cherche à approcher ce problème à l'aide d'un réseau de neurones. Pour ce faire, on dispose de 100 couples d'apprentissage. On considère

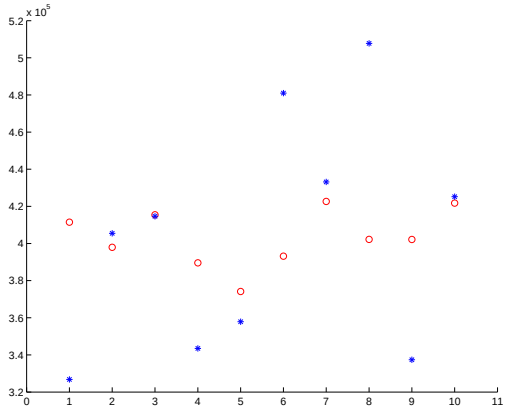
- un lot L1 de 80 couples pour l'apprentissage,
- un lot L2 de 15 couples pour le choix du paramètre de régularisation β,
- et un lot L3 de 5 couples pour la vérification de la généralisation.

La stratégie d'apprentissage du réseau de neurones à trois couches que l'on considère est décrite ci-dessous. Les résultats qui suivent ont été obtenus avec 12 neurones dans la couche cachée.

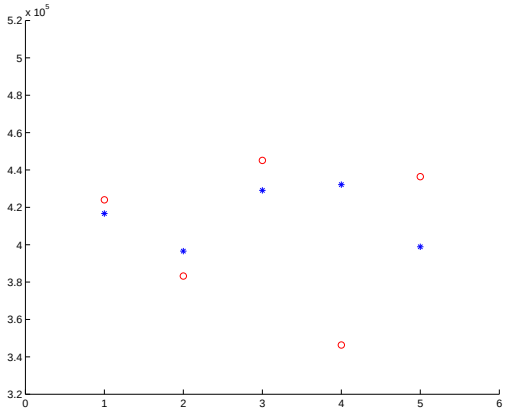
On fait l'apprentissage "par coeur", c'est-à-dire que l'on cherche à bien apprendre les couples du lot L1, mais on ne fait aucun traitement pour généraliser les lots L2 et L3. Dans la figure 3.21, on peut constater que l'apprentissage du lot L1 est correct, et comme prévu (puisque rien n'a été fait), il n'y a pas de généralisation sur les lots L2 et L3.



Apprentissage du lot L1 avant régularisation



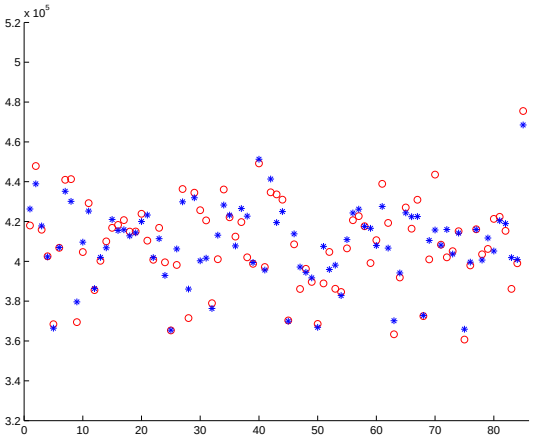
Comportement du lot L2
avant régularisation



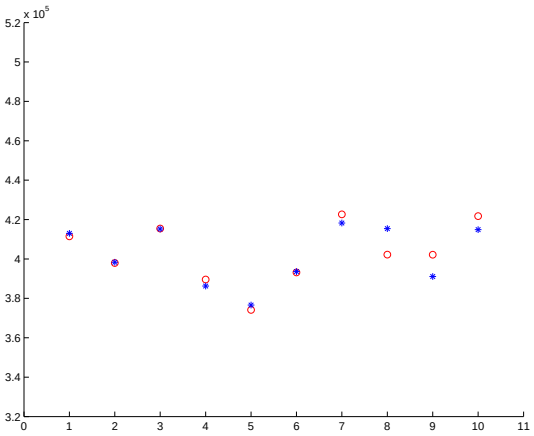
Comportement du lot L3
avant régularisation

FIG. 3.21 – Crash : apprentissage avant régularisation

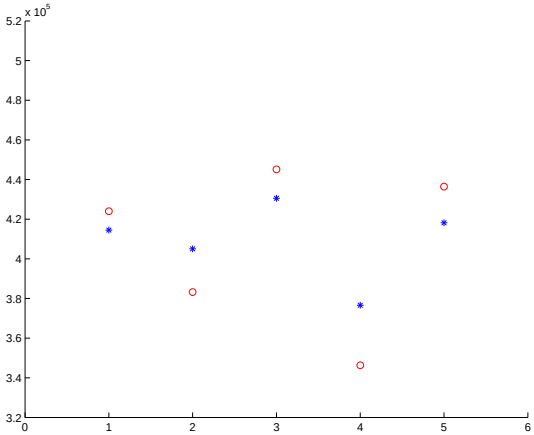
Pour parvenir à généraliser les lots L2 et L3, la seconde étape est nécessaire. Elle sert à trouver le paramètre de régularisation β qui permet que le résidu sur le lot L2 soit le plus petit possible, sans trop dégrader l'apprentissage du lot L1. Une fois trouvé le paramètre de régularisation, on apprend les lots L1 et L2 ensemble, puis on vérifie la généralisation sur le lot L3. On peut apprécier le résultat sur la figure 3.22.



Apprentissage du lot L1
après régularisation



Comportement duApprentissage du lot L2
après régularisation



Comportement du lot L3 après régularisation

FIG. 3.22 – Crash : apprentissage après régularisation

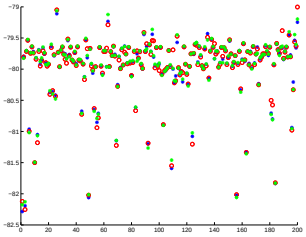
Un autre problème du CRASH

On étudie un modèle de choc latéral. Le modèle du pied milieu est découpé en 5 zones. Les variables d'entrée sont les épaisseurs de coque (de 0.6 a 1.5 avec un pas de 0.1) pour chaque zone. Les variables de sortie, au nombre de 6, sont des critères de déplacement. On dispose de 250 données que l'on réparti en 3 lots :

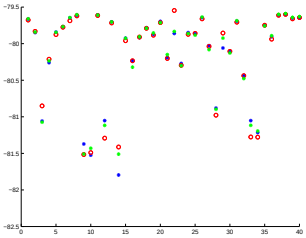
- un lot L1 de 200 couples pour l'apprentissage,
- un lot L2 de 40 couples pour le choix du paramètre de régularisation β,
- et un lot L3 de 10 couples pour la vérification de la généralisation.

Nb. de sorties	6
Structure du réseau	6 / 10 / 6
Lots	200 / 40 / 10
Option	1

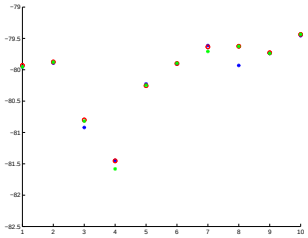
On fait l'apprentissage « par coeur », pour bien apprendre les couples du lot L1. Dans les figures ci-dessous (3.23, 3.24, 3.25, 3.26, 3.27, 3.28), on peut constater, en comparant l'écart entre les étoiles bleues et les cibles rouges, que l'apprentissage du lot L1 est correct. Les étoiles vertes représentent les résultats de l'apprentissage après régularisation, c'est-à-dire après la phase 2 décrite dans la section 3.6.3 page 41.



Apprentissage du lot L1

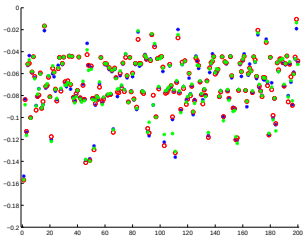


Apprentissage et comportement du lot L2

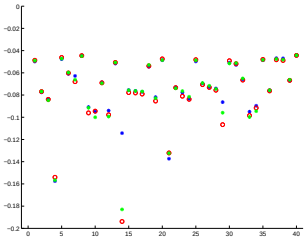


Comportement du lot L3

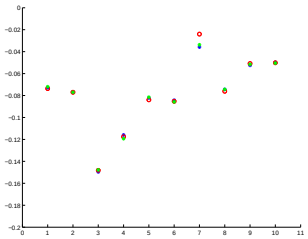
FIG. 3.23 – Crash : apprentissage avant et après régularisation du déplacement 1



Apprentissage du lot L1

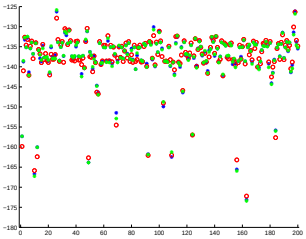


Apprentissage et comportement du lot L2

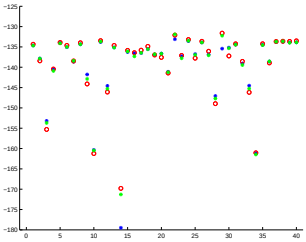


Comportement du lot L3

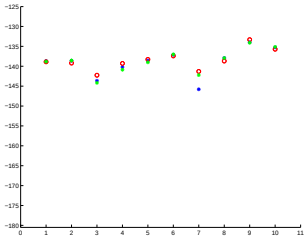
FIG. 3.24 – Crash : apprentissage avant et après régularisation du déplacement 2



Apprentissage du lot L1

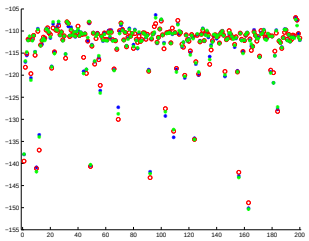


Apprentissage et comportement du lot L2

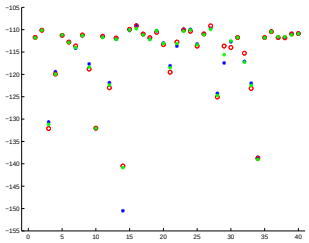


Comportement du lot L3

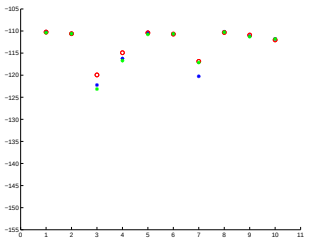
FIG. 3.25 – Crash : apprentissage avant et après régularisation du déplacement 3



Apprentissage
du lot L1

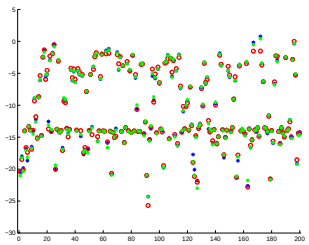


Apprentissage et comportement
du lot L2

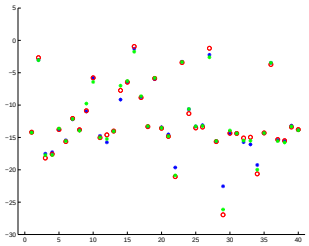


Comportement
du lot L3

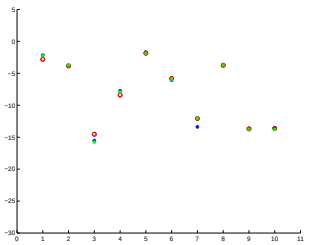
FIG. 3.26 – Crash : apprentissage avant et après régularisation du déplacement 4



Apprentissage
du lot L1

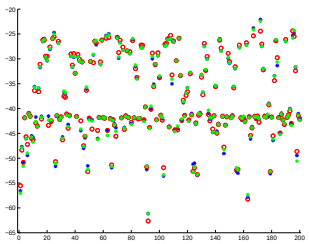


Apprentissage et comportement
du lot L2

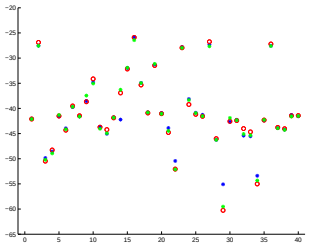


Comportement
du lot L3

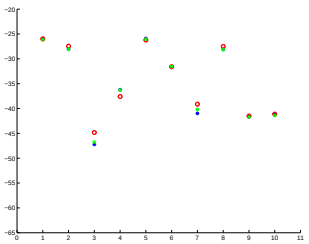
FIG. 3.27 – Crash : apprentissage avant et après régularisation du déplacement 5



Apprentissage
du lot L1



Apprentissage et comportement
du lot L2



Comportement
du lot L3

FIG. 3.28 – Crash : apprentissage avant et après régularisation du déplacement 6

Dans les figures précédentes, on peut remarquer que, pour tous les déplacements, l'apprentissage avant régularisation du lot L1 est correct. En ce qui concerne la généralisation,

les lots L2 et L3 se comportent déjà bien avant régularisation (étoiles bleues). Après la seconde phase (étoiles vertes), les résultats sont encore meilleurs : l'apprentissage des lots L1 et L2 est correct et la généralisation du lot L3 n'est pas dégradée.

Un exemple en industrie pétrolière

Le cas PUNQ est un modèle de réservoir synthétique inspiré de données réelles. Il a été créé, dans le projet européen PUNQ, afin de comparer les différentes méthodes d'inversion permettant de contraindre les modèles stochastiques de réservoir aux données dynamiques. Le modèle géologique est composé de 5 couches indépendantes. Les couches 1, 3, 4 et 5 sont supposées de bonne qualité tandis que la couche 2 est de moins bonne qualité. Les porosités et perméabilités ont été générées en utilisant des méthodes géostatistiques. Sur la carte du toit du réservoir (figure 3.29), on peut voir que celui-ci est entouré par un aquifère au Nord et à l'Ouest et est délimité par une faille au Sud et à l'Est.

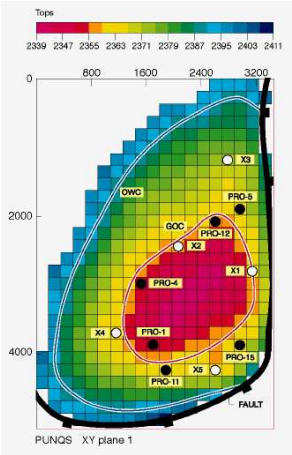


FIG. 3.29 – Cas pétrole : toit d'un modèle de réservoir synthétique

La fonction objectif (représentée figure 3.30) a été calculée en prenant en compte le « Gas Oil Ratio », la « Bottom Hole Flowing Pressure »et le « Water Cut »pour les 6 puits producteurs. Les données observées utilisées correspondent à un cas de référence connu. Les deux paramètres qu'on a fait varier pour tracer la fonction coût correspondent, pour x_1, à un multiplicateur de la perméabilité horizontale pour les 5 couches et, pour x_2, à la même chose pour la perméabilité verticale. La quantité x_1 prend 29 valeurs entre 0.1 et 10, tandis que x_2 en prend 28 entre 10^{-6} et 3.

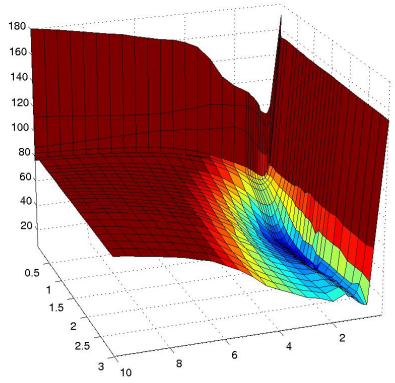


FIG. 3.30 – Cas pétrole : représentation de la fonction objectif à minimiser

On dispose de 812 données qui sont réparties de la manière suivante
– un lot L1 de 80% des données pour l'apprentissage,
– un lot L2 de 20% des couples pour le choix du paramètre de régularisation β.
Les résultats qui suivent ont été obtenus avec 20 neurones dans la couche cachée.

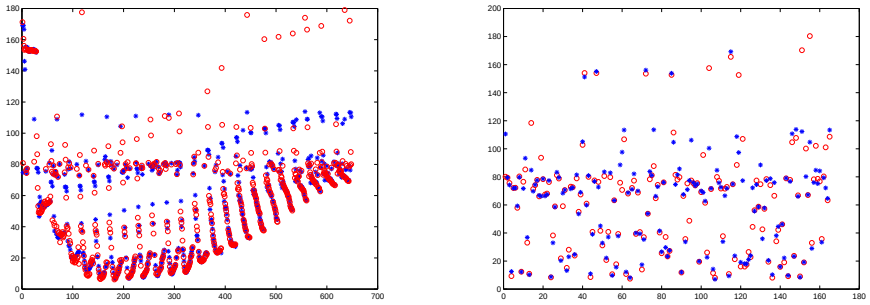


FIG. 3.31 – Apprentissage et généralisation avec $\beta = 0$ (cibles en rouge, réseau en bleu)

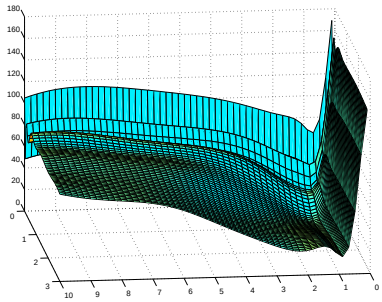


FIG. 3.32 – Cas pétrole : réseau approchant la fonction objectif ($\beta = 0$)

Dans la figure 3.31, on peut constater que l'apprentissage est correct, sauf pour certaines valeurs qui se trouvent au-dessus de 120. On passe donc à la phase suivante : trouver la

valeur du paramètre de régularisation β. Les résultats obtenus pendant cette phase peuvent être observés sur les figures 3.33 et 3.34.

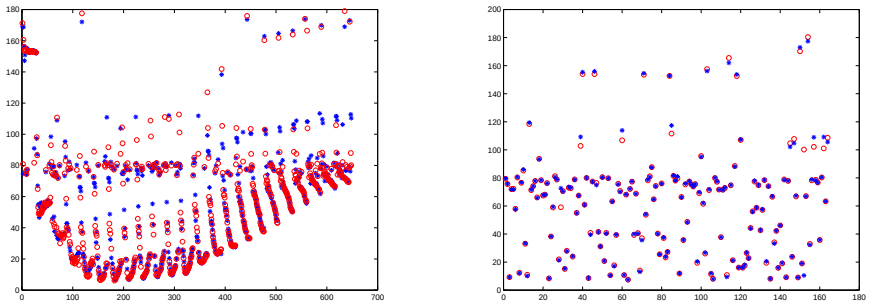


FIG. 3.33 – Apprentissage et généralisation avec β optimal (cibles en rouge, réseau en bleu)

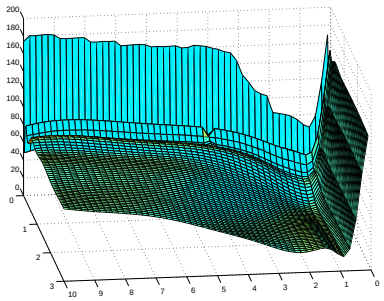


FIG. 3.34 – Cas pétrole : réseau approchant la fonction objectif (β optimal)

Après la phase 2, on peut observer que les résultats sont bien meilleurs. L'apprentissage n'a pas été dégradé et on parvient à obtenir une très bonne généralisation.

Chapitre 4

Apprentissage pour l'optimisation

Certains problèmes des sciences physiques sont difficiles à résoudre dans le sens où les seules données sont des mesures parfois coûteuses ou longues à réaliser. Une optimisation est, dans ces cas-là, forcément limitée par le nombre réduit de données. On choisit donc souvent de remplacer la fonction à optimiser par un modèle peu coûteux auquel on peut appliquer les algorithmes classiques de l'optimisation [23, 19, 56, 9]. Ainsi, on construit une surface de réponse à partir des mesures dont on dispose. Nous avons vu au chapitre 3 comment construire une surface de réponse à l'aide des réseaux de neurones. Ces derniers permettent de disposer d'un gradient, pour un coût ajouté négligeable, grâce à la différentiation automatique [22, 8, 7, 49, 50, 28, 21]. Nous allons expliquer comment nous avons exploité le modèle ainsi obtenu pour résoudre le problème (d'optimisation) original.

4.1 Boucle d'optimisation

On note $G : \mathbb{R}^e \to \mathbb{R}$ la fonction à minimiser et R un réseau de neurones qui doit l'approcher. Soit Ω l'ensemble des couples d'apprentissage. Il est choisi en utilisant les techniques de plan d'expérience proposées dans le chapitre 2. On construit alors un premier modèle R par réseau de neurones, avec l'ensemble Ω, en utilisant la technique proposée dans la section 3.6 page 39. Ce modèle est ensuite optimisé par rapport aux signaux d'entrée. Notons X^* la solution de cette optimisation. On compare alors $R(X^*)$ à $G(X^*)$. Si les deux valeurs coïncident, alors on considère que la quantité X^* est le minimum de la fonction G et l'algorithme s'arrête. Si le réseau R et la fonction G ne donnent pas le même résultat ($|R(X^*) - G(x^*)| > \varepsilon$), on considère que le réseau n'a pas suffisamment bien appris la fonction. On ajoute alors le nouveau couple $(X^*, G(X^*))$ à l'ensemble d'apprentissage Ω et on refait la boucle d'optimisation (fig. 4.1). Des travaux similaires sont consultables dans [20, 41]. Une bonne approximation de la fonction dépend de plusieurs facteurs (nombre d'itérations dans la méthode de Gauss-Newton, nombre de neurones cachés, ...), mais le nombre des couples initiaux est fondamental. Dans ce chapitre, nous commençons par faire une première approximation avec un nombre limité de couples d'apprentissage. Puis nous enrichissons le modèle R en ajoutant les couples de points proposés par l'optimiseur.

Nous allons illustrer cette démarche en l'appliquant à des fonctions de dimension 2. Puis nous résumerons dans un tableau les résultats que nous avons obtenus pour quelques

fonctions de dimension supérieure. Enfin, nous conclurons avec un résultat dans le domaine pétrolier et une perspective dans le domaine automobile.

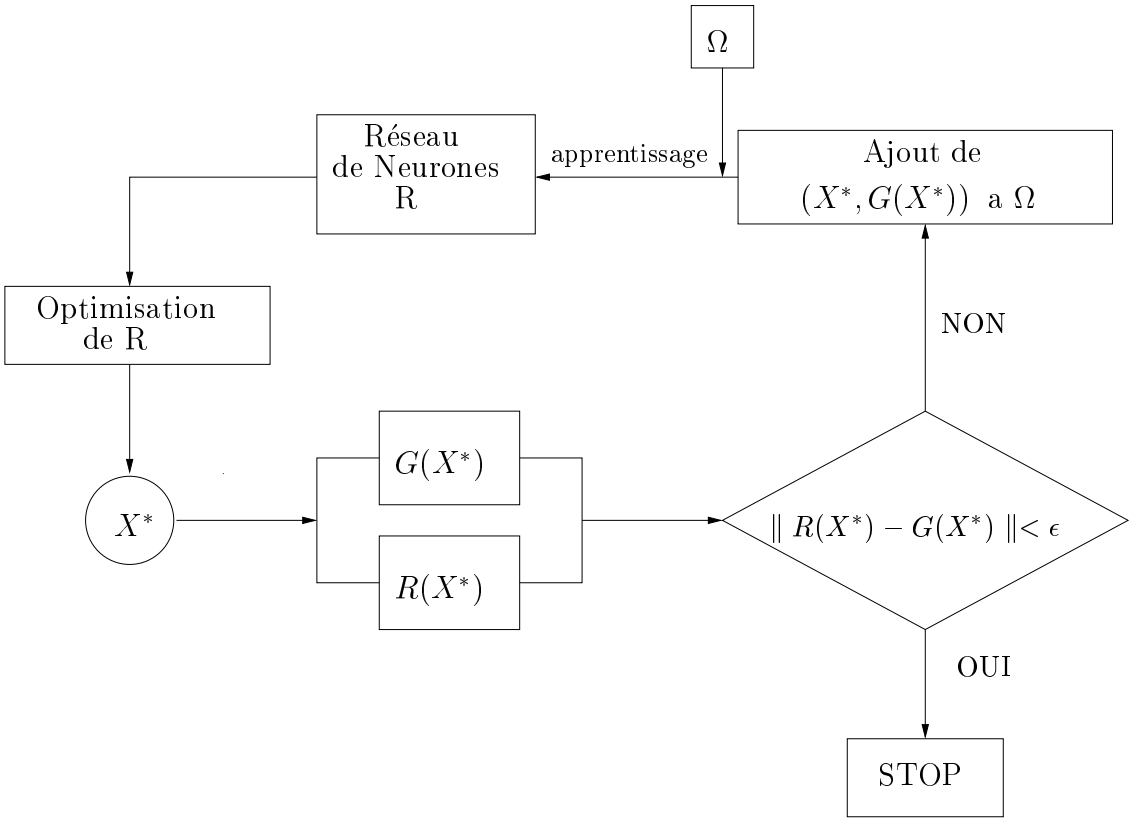


FIG. 4.1 – Schéma de la boucle d'optimisation

4.2 Résultats numériques

4.2.1 Exemples en dimension deux

Nous allons tester la méthode décrite ci-dessus en l'appliquant à des fonctions de deux variables.

Comme cela a déjà été dit dans la section précédente, la première étape consiste à faire une première approximation de la fonction considérée. Pour ce faire, nous utilisons la méthode décrite dans le chapitre 3. La différence que l'on peut observer entre les premiers réseaux dans ce chapitre 4 et les résultats d'approximation présentés dans le chapitre 3 est due au nombre de couples utilisés pour faire le premier apprentissage. En effet, si le premier réseau approche trop fidèlement la fonction à minimiser, il existe de gros risques que l'optimisation aboutisse à un minimum local. Nous préférons donc démarrer avec un réseau qui approche très grossièrement la fonction à minimiser et laisser la liberté à l'optimiseur de proposer les points pertinents qui permettront d'améliorer le réseau au fur et à mesure.

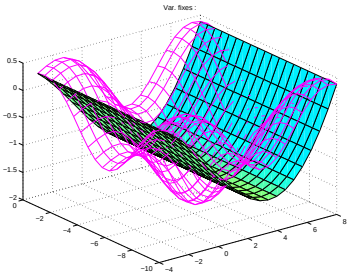
Sinus

Données

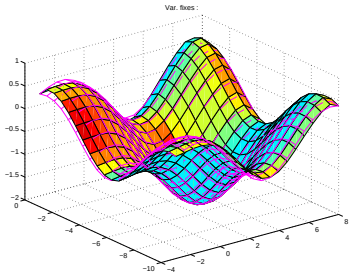
Fonction	$f(x,y) = -\frac{\sin(x-3)}{(x-3)} - \frac{\sin(y+5)}{(y+5)}$
Domaine	$D = [-3,8] \times [-10,0]$
Structure du réseau	3 / 5 / 1
Nombre de couples initiaux	9
Option	3

Apprentissage et optimisation

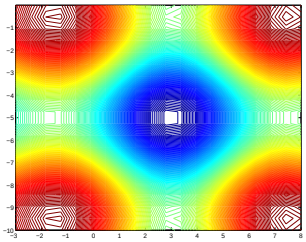
Solution exacte/approchée	[3.0 , -5.0] / [3.012, -5.000]
Nb. total de couples	30



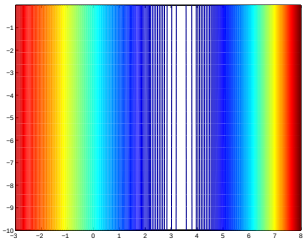
Premier réseau



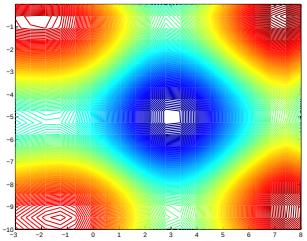
Réseau final



Contours de la fonction,



du premier réseau



et du réseau final

FIG. 4.2 – Apprentissage et optimisation de la fonction sinus

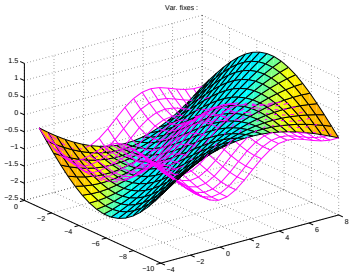
« Sinus décalé »

Données

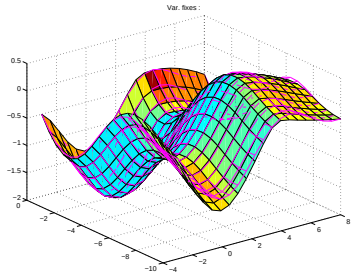
Fonction	$f(x,y) = -\frac{\sin(x)}{(x)} - \frac{\sin(y+2)}{(y+2)}$
Domaine	$D = [-3, 8] \times [-10, 0]$
Structure du réseau	3 / 5 / 1
Nb. de couples d'apprentissage initial	9
Option	3

Apprentissage et optimisation

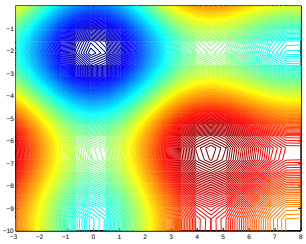
Solution exacte/approchée	[0.0 , -2.0] / [0.003, -1.997]
Nb. total de couples	23



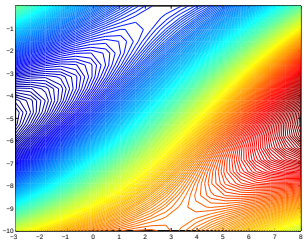
Premier Réseau



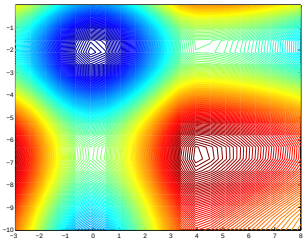
Réseau final



Contours de la fonction,



du premier réseau



et du réseau final

FIG. 4.3 – Apprentissage et optimisation de la fonction « sinus décalé »

« Chameau »

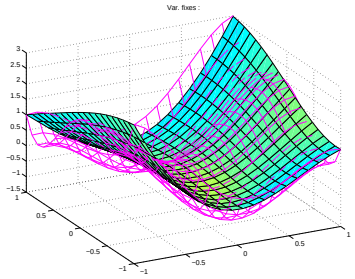
Cette fonction est symétrique par rapport à l'origine. Elle admet 3 minima locaux. Le minimum global se situe au point $\pm(0.0898, -0.7126)$

Données

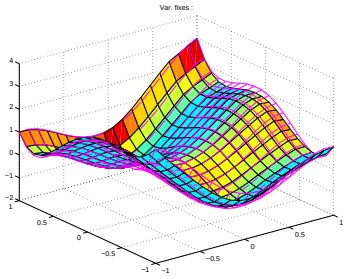
Fonction	$f(x,y) = 4x^4 - \frac{21}{10}x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$
Domaine	$D = [-1,1] \times [-1,1]$
Structure du réseau	3 / 14 / 1
Nombre de couples initiaux	15
Option	3

Apprentissage et optimisation

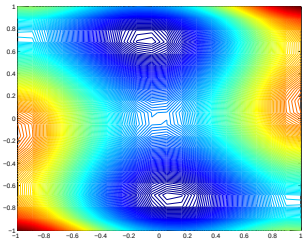
Solution exacte/approchée	[0.0898 , 0.7126] / [-0.0917, 0.7128]
Nb. total de couples	30



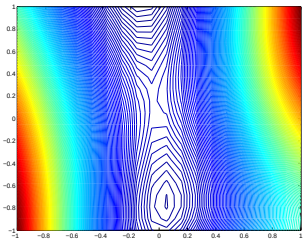
Premier réseau



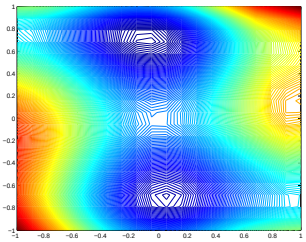
Réseau final



Contours de la fonction,



du premier réseau



et du réseau final

FIG. 4.4 – Apprentissage et optimisation de la fonction « chameau »

Branin

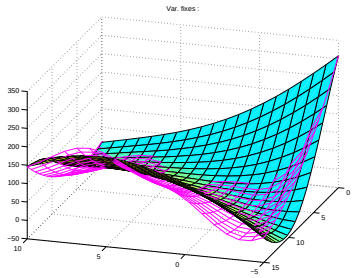
Cette fonction admet sa valeur minimale (environ 0.398) en 3 points différents :
$(-3.142, 12.275)$, $(3.142, 2.275)$, $(9.425, 2.425)$.

Données

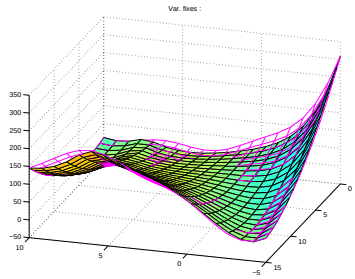
Fonction	$f(x,y) = \left(y - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6\right)^2 + \left(10\left(1 - \frac{1}{8\pi}\right)\cos(x) + 10\right)$
Domaine	$D = [-5, 10] \times [0, 15]$
Structure du réseau	3 / 12 / 1
Nb. de couples d'apprentissage initial	15
Option	3

Apprentissage et optimisation

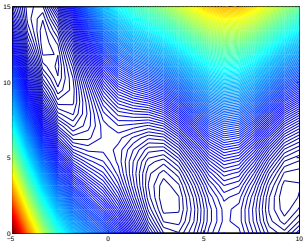
Solution exacte/approchée	[9.425 , 2.425] / [9.4399 , 2.4814]
Nb. total de couples	24



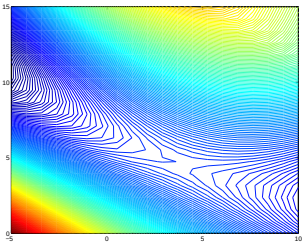
Premier réseau



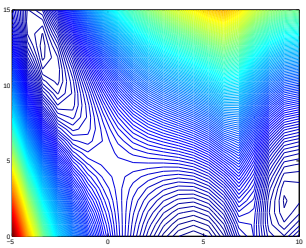
Réseau final



Contours de la fonction,



du premier réseau



et du réseau final

FIG. 4.5 – Apprentissage et optimisation de la fonction de Branin

Griewank

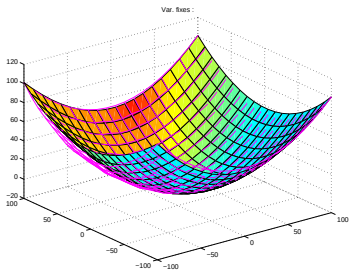
Cette fonction admet plus de 500 minima locaux. Son minimum global est atteint en $x = y = 0$.

Données

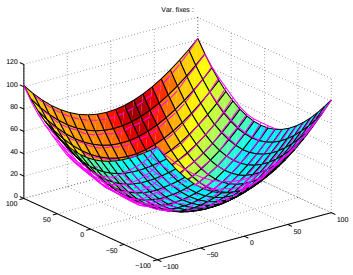
Fonction	$f(x,y) = \frac{x^2}{200} + \frac{y^2}{200} - \cos(\frac{x}{\sqrt{1}})\cos(\frac{y}{\sqrt{2}}) + 1$
Domaine	$D = [-100, 100] \times [-100, 100]$
Structure du réseau	3 / 10 / 1
Nb. de couples d'apprentissage initial	10
Option	3

Apprentissage et optimisation

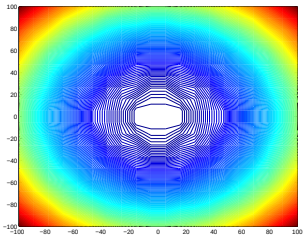
Solution exacte/approchée	[0.0,0.0] / [-0.073, 0.079]
Nb. total de couples	13

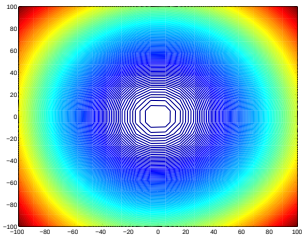


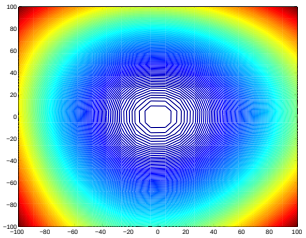
Premier réseau



Réseau final







Contours de la fonction, du premier réseau et du réseau final

FIG. 4.6 – Apprentissage et optimisation de la fonction de Griewank

Rastrigin

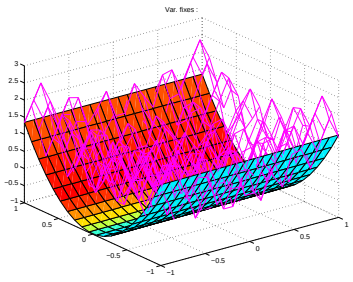
Cette fonction admet plus de 50 minima locaux. Son minimum global est atteint au point $x = y = 0$.

Données

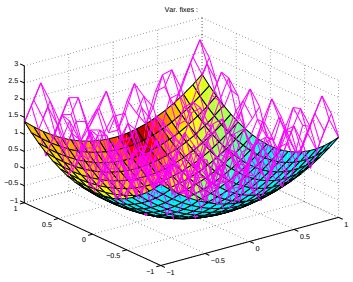
Fonction	$f(x,y) = x^2 + y^2 - cos(18x) - cos(18y)$
Domaine	$D = [-1,1] \times [-1,1]$
Structure du réseau	3 / 10 / 1
Nb. de couples d'apprentissage initial	10
Option	3

Apprentissage et optimisation

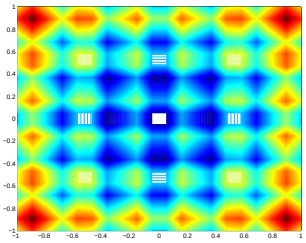
Solution exacte/approchée	[0.0,0.0] / [-0.0152 , -0.0326]
Nb. total de couples	20



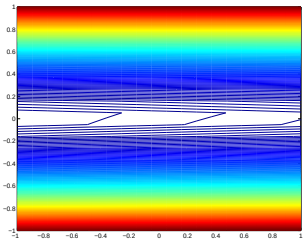
Premier réseau



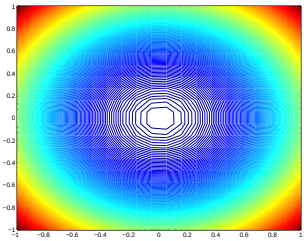
Réseau final



Contours de la fonction,



du premier réseau



et du réseau final

FIG. 4.7 – Apprentissage et optimisation de la fonction de Rastrigin

Rosenbrock

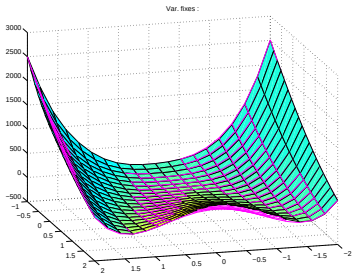
Cette fonction est connue pour être la « fonction banane » et son minimum global est atteint au point $x = y = 1.0$.

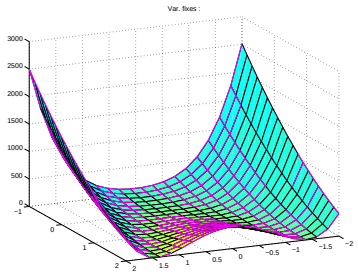
Données

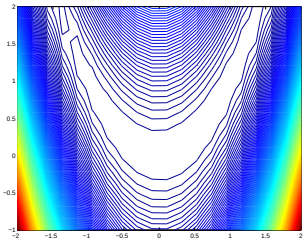
Fonction	$f(x, y) = 100(x^2 + y^2)^2 + (1 - x)^2$
Domaine	$D = [-2, 2] \times [-2, 2]$
Structure du réseau	3 / 12 / 1
Nb. de couples d'apprentissage initial	25
Option	3

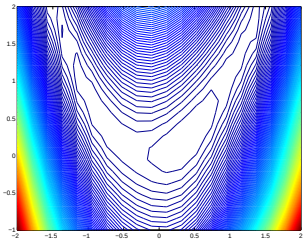
Apprentissage et optimisation

Solution exacte/approchée	[1.0 , 1.0] / [1.062, 1.127]
Nb. total de couples	50


Premier réseau


Réseau final


Contours de la fonction,


du premier réseau

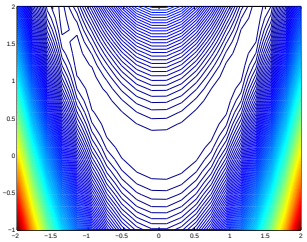

et du réseau final

FIG. 4.8 – Apprentissage et optimisation de la fonction de Rosenbrock

Récapitulatif

Nous indiquons, dans les tableaux ci-dessous (4.1, 4.2), l'ensemble des résultats numériques d'optimisation que nous avons obtenus pour l'optimisation de fonction de deux variables.

Ces résultats nous semblent satisfaisants. En particulier, on peut remarquer que le nombre d'évaluations de la fonction ne dépasse jamais 50.

Naturellement, pour les fonctions de Rastrigin ou de Griewank, le réseau n'approche pas la fonction, mais nous rappelons que l'objectif n'est pas de construire une bonne approximation de ces fonctions, mais bien de trouver le minimum global sans être piégé par les minima locaux.

4.2.2 Exemples en dimension supérieure

Nous avons également réalisé des tests en dimension supérieure, dont les résultats figurent dans le tableau 4.3. Ils nous paraissent assez satisfaisants en ce sens que le minimum approché n'est pas trop éloigné du minimum exact.

4.2.3 Comparaison de nos résultats avec ceux de C. Massat

Le tableau 4.4 permet de comparer nos résultats (colonne MV), en terme de nombre d'évaluations de la fonction, à ceux obtenus par C. Massat [50] (colonne CM). Pour mémoire, cette dernière utilisait une méthode combinant des idées issues des méthodes de descente, des algorithmes génétiques et du recuit simulé.

4.2.4 Exemples industriels

Dans l'industrie pétrolière

Il s'agit du problème décrit dans le chapitre 3.

Comme dans la majorité des cas industriels, nous ne disposions pas de la fonction-objectif, mais uniquement de 812 mesures.

Nous avons commencé par construire un réseau (visualisé dans la figure 3.34 page 62) à l'aide de toutes ces données. C'est ce modèle que nous avons utilisé en guise de fonction-objectif dans la boucle d'optimisation 4.1, que nous avons initialisé avec un ensemble de 100 couples d'apprentissage. Comme il est signifié dans le tableau 4.5, nous n'avons eu besoin que de 12 évaluations supplémentaires de la fonction-objectif pour aboutir à un minimum qui a complètement satisfait les ingénieurs pétroliers.

Nb de couples initial / total	Sol exacte	Sol approchée
100 / 112	(1, 1)	(1.01245, 0.98270)

TAB. 4.5 – Résultats de l'optimisation pour le cas pétrole (**Punq**)

Fonction	**Domaine**	**Nb de couples initial/total**	**Solution exacte**	**Solution approchée**
$x^2 + y^2$	$-100 \leq x, y \leq 100$	10 / 19	(0,0)	(0.051421,0.07004)
$(x-2)^2 + (y-2)^2$	$-3 \leq x, y \leq 3$	10 / 13	(2,2)	(1.9929,1.99055)
$-\frac{\sin(x-3)}{(x-3)} - \frac{\sin(y+5)}{(y+5)}$	$-3 \leq x \leq 8$ $-10 \leq y \leq 0$	9 / 30	(3,-5)	(3.01209,-5.0002)
$-\frac{\sin(x)}{(x)} - \frac{\sin(y+2)}{(y+2)}$	$-3 \leq x \leq 8$ $-10 \leq y \leq 0$	9 / 23	(0,2)	(0.00319,-1.99695)
$(30 + x\sin(x))(4 + e^{-y})$	$0 \leq x, y \leq 5$	9 / 12	(5,5)	(5,5)
$e^x(4x^2 + 2y^2 + 4xy + 2y + 1)$	$-1 \leq x, y \leq 1$	9 / 12	(0.5,-1)	(0.51352,-1)
Goldstein $[1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)]*$ $[30 + (2x-3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$	$-2 \leq x, y \leq 2$	9 / 46	(0,-1)	(-0.086046,-1.01584)

TAB. 4.1 – Résultats d'optimisation pour des fonctions en dimension 2

Fonction	Domaine	Nb de couples initial/total	Solution exacte	Solution approchée
Hamma [31] $\frac{1}{3}x^3 + y^3 - x^2 - y^2 + \frac{8}{3}$	$0 \leq x, y \leq 5$	9 / 24	(2,2)	(1.9994,2.00978)
Rosenbrock $100(x^2 + y^2)^2 + (1 - x)^2$	$-2 \leq x, y \leq 2$	25 / 50	(1,1)	(1.06197,1.12756)
Branin $(y - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6)^2$ $+10(1 - \frac{1}{8\pi})\cos(x) + 10$	$-5 \leq x \leq 10 \quad 0 \leq y \leq 15$	15 / 24	(9.425 , 2.425)	(9.4399 , 2.4814)
Freudenstein $(-13 + x + ((15 - y)y - 2)y)^2$ $+(-29 + x + ((y + 1)y - 14)y)^2$	$0 \leq x \leq 15 \quad 2 \leq y \leq 5$	9 /20	(5,4)	(5.049, 3.98)
Griewank $\frac{x^2}{200} + \frac{y^2}{200} - \cos(\frac{x}{\sqrt{1}})\cos(\frac{y}{\sqrt{2}}) + 1$	$-100 \leq x, y \leq 100$	10 / 13	(0,0)	(-0.073,0.079)
Rastrigin $x^2 + y^2 - \cos(18x) - \cos(18y)$	$-1 \leq x, y \leq 1$	10 / 20	(0,0)	(-0.0152 , -0.0326)
Chameau $4x^4 - \frac{21}{10}x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$	$-1 \leq x, y \leq 1$	15 / 30	(0.0898, 0.7126)	(-0.0917, 0.7128)

TAB. 4.2 – Résultats d'optimisation pour des fonctions en dimension 2 (suite)

Tab. 4.3 – Résultats d'optimisation pour des fonctions en dimension supérieure

Fonction	**Dim**	**Domaine**	**Nb de couples initial/total**	**Solution exacte**	**Solution approchée**
$\sum_{i=1}^{4} x_i^2$	4	$-100 \leq x_i \leq 100$	30 / 38	(0, 0, 0, 0)	(0.01606, 0.115235, 0.07596, 0.01757)
$(x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 +$ $(x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$	4	$-3 \leq x_i \leq 3$	20 / 50	(0, 0, 0, 0)	(-0.02593, 0.05995, 0.016419, -0.01721)
$\sum_{i=1}^{6} x_i^2$	6	$-100 \leq x_i \leq 100$	20 / 36	(0, 0, 0 0, 0, 0)	(0.08385, 0.08517, 0.00215, 0.00066, -0.03629, -0.016263)

Fonction	Domaine	MV	CM
Chameau $4x^4 - \frac{21}{10}x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$	$-1 \leq x, y \leq 1$	30	94
Goldstein $[1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)]*$ $[30 + (2x-3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$	$-2 \leq x, y \leq 2$	46	355
$(x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 +$ $(x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$	$-3 \leq x_i \leq 3$	50	700
Branin $(y - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6)^2$ $+10(1 - \frac{1}{8\pi})\cos(x) + 10$	$-5 \leq x \leq 10$ $0 \leq y \leq 15$	24	117
Rastrigin $x^2 + y^2 - \cos(18x) - \cos(18y)$	$-1 \leq x, y \leq 1$	20	98

TAB. 4.4 – Comparaison de nos résultats avec ceux de la thèse de C. Massat

Dans l'industrie automobile

Un autre exemple industriel est en cours d'étude dans le domaine de l'optimisation de forme de pare-brise chez VALEO. L'essuyage d'un pare-brise est fortement influencé par la forme de la vitre et par l'orientation de l'axe de rotation du système d'essuyage. L'optimisation de la forme de la vitre est donc effectuée en deux temps. La première étape consiste à optimiser, pour une vitre donnée, l'orientation de l'axe de rotation du système d'essuyage. Dans ce contexte, on a 3 variables indépendantes qui correspondent à des angles d'orientation du balai et 6 critères globaux (3 pour le côté passager et 3 pour le coté conducteur). Dans un second temps, on procède à l'optimisation de la forme du pare-brise. Cette fois, les paramètres d'entrée sont au nombre de 9 et correspondent à des points de contrôle d'une modification de surface représentée par un produit tensoriel de fonctions splines. On apprend au réseau les différents critères globaux. La fonction coût se présente alors comme une somme pondérée des différents critères. Le lecteur intéressé pourra se référer à la thèse de Fabien Muradore [54].

Conclusion

Nous avons contribué par ce travail à une amélioration des réseaux neuronaux. Cette amélioration comprend plusieurs aspects. Le premier aspect est lié à la mise en oeuvre d'une méthode de Gauss Newton pour l'apprentissage sans calcul de jacobienne. En effet, en combinant les modes direct et inverse de la différentiation automatique, nous évitons le calcul de la jacobienne, de son produit par sa transposée et la factorisation de la matrice ainsi obtenue. Cette méthode permet de réduire d'une manière considérable la mémoire nécessaire, permettant ainsi le traitement de problèmes d'apprentissage qui étaient hors de portée par les méthodes classiques.

Une autre amélioration concerne la maîtrise d'une technique de régularisation qui permet la généralisation (réponse exacte du réseau neuronal sur des données non apprises). La méthode de régularisation communément utilisée consiste à réduire le nombre de cellules cachées du réseau neuronal. Cette approche conduit malheureusement à un mauvais apprentissage. Nous proposons de commencer à apprendre une partie des données (lot L1) sans régularisation pour s'assurer que le réseau comprend suffisamment de neurones pour effectuer un apprentissage (par coeur). Ensuite, nous appliquons un certain nombre de techniques de régularisation :

- Gauss-Newton,
- Tikhonov, restreint au premier étage du réseau,
- arrêt prématuré des itérations de Gauss-Newton,
- réduction de nombre de neurones cachées.

de sorte à minimiser le résidu sur les données non apprises (lot L2).

En ce qui concerne l'optimisation, au lieu de minimiser une fonction coût complexe définie par un code (ou par des expériences coûteuses), nous proposons d'appliquer l'algorithme d'optimisation à un modèle réduit défini par un réseau neuronal. Chaque fois que l'algorithme d'optimisation, appliqué au modèle, converge, nous évaluons la fonction coût complexe au point optimal proposé et nous demandons au réseau neuronal d'apprendre ce nouveau point, ... etc. Le processus s'arrête au bout d'un certain nombre d'itérations. La succès constaté de la méthode (au plus 50 évaluations de la fonction coût complexe) est intimement lié à la maîtrise de la méthode de régularisation. Remarquons que c'est exactement le même algorithme qui est appliqué à des fonctions de nature très différentes : d'un côté les fonctions de Griewank et de Rastrigin qui présentent plusieurs minima locaux, de l'autre côté la fonction de Rosenbrock qui présente des difficultés liées à son bassin d'attraction.

Bibliographie

[1] U. an der Heiden. *Analysis of neural networks*, volume 35 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1980.

[2] A. Aussem. *Théorie et application des réseaux de neurones récurrents et dynamiques à la prédiction, à la modélisation et au contrôle adaptatif des processus dynamiques*. PhD thesis, Université René Descartes - Paris V, 1995.

[3] J. Balicki, A. Stateczny, and B. Żak. Genetic algorithms and Hopfield neural networks for solving combinatorial problems. *Appl. Math. Comput. Sci.*, 7(3) :567–592, 1997.

[4] S. D. Balkin and D. K. Lin. A neural network approach to response surface methodology. *Communications in Statistics - Theory and Methods*, 29(9-10) :2215–222, 2000.

[5] P. L. Bartlett. For valid generalization, the size of the weights is more important than the size of the network. *Neural Information Processing Systems*, 9, 1997.

[6] D. Benoist, Y. Tourbier, and S. Germain-Tourbier. *Plans d'expériences : construction et analyse*. Technique et documentation Lavoisier, 1994.

[7] Martin Berz, Christian Bischof, and George Corliss, editors. *Computational differentiation*, Philadelphia, PA, 1996. Society for Industrial and Applied Mathematics (SIAM). Techniques, applications, and tools.

[8] Christian Bischof and Andreas Griewank. Computational differentiation and multidisciplinary design. In *Inverse problems and optimal design in industry (Philadelphia, PA, 1993)*, volume 10 of *European Consort. Math. Indust.*, pages 187–211. Teubner, Stuttgart, 1994.

[9] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. Sagastizábal. *Numerical optimization*. Universitext. Springer-Verlag, Berlin, 2003. Theoretical and practical aspects, Translated and revised from the 1997 French original.

[10] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets : Backpropagation, conjugated gradient, and early stopping. In *Neural information Processing Systems*, 2000.

[11] A. Cichocki and Unbehauen R. *Neural Networks for Optimization and Signal Processing*. John Wimey & Sons, 1993.

[12] W. G. Cochran and G. M. Cox. *Experimental designs*. Wiley Classics Library. John Wiley & Sons Inc., New York, second edition, 1992. A Wiley-Interscience Publication.

[13] D. Collombier. *Plans d'expérience factoriels*, volume 21 of *Mathématiques & Applications (Berlin) [Mathematics & Applications]*. Springer-Verlag, Berlin, 1996. Construction et propriétés des fractions de plans. [Construction and properties of fractions of designs].

[14] I. De Falco, A. Cioppa, P. Natale, and E. Tarantino. Artificial neural networks optimization by means of evolutionary algorithms. http ://citeseer.nj.nec.com/defalco97artificial.html, 1997.

[15] D. de Werra and A. Hertz. Tabu search techniques. A tutorial and an application to neural networks. *OR Spektrum*, 11(3) :131–141, 1989.

[16] G. Dreyfus, J. M. Martinez, M. Samuelides, M.B. Gordon, F. Badran, S. Thiria, and L. Hérault. *Réseaux de neurones. Méthodologie et applications*. Editions Eyrolles, 2000.

[17] D. L. Elliott. A better activation function for artificial neural networks. Technical report, Institute for Systems Research, College Park, MD, 20742 USA, 1993.

[18] D. Euvrard. *Résolution numérique des équations aux dérivées partielles de la physique, de la mécanique et des sciences de l'ingénieur*. Enseignement de la Physique : Mathématiques pour la Physique. [The Teaching of Physics : Mathematics for Physics]. Masson, Paris, third edition, 1994. Différences finies, éléments finis, problèmes en domaine non borné. [Finite differences, finite elements, problems in unbounded domains].

[19] R. Fletcher. *Practical methods of optimization*. A Wiley-Interscience Publication. John Wiley & Sons Ltd., Chichester, second edition, 1987.

[20] A. Gaidon. Multi-objective optimization on ramjet-powered missile performance. In *MACSInet*, 2003.

[21] J. C. Gilbert. Principes de la différentiation automatique. Technical report, INRIA, 2001.

[22] J. C. Gilbert, G. Le Vey, and J. Masse. La différentiation automatique de fonctions représentées par des programmes. Technical report, INRIA, 1997.

[23] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1981.

[24] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, pages 219–269, 1995.

[25] J. Goupy. *La méthode des plans d'expériences : optimisation du choix des essais et de l'interprétation des résultats*. Dunod, 1996.

[26] J. Goupy. *Plans d'expériences pour surfaces de réponse*. Dunod, 1999.

[27] J. Goupy. *Plans d'expériences : les mélanges*. Dunod, 2000.

[28] A. Griewank. *Evaluating derivatives*, volume 19 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. Principles and techniques of algorithmic differentiation.

[29] Ph. Guillaume and M. Masmoudi. Computation of high order derivatives in optimal shape design. *Numer. Math.*, 67(2) :231–250, 1994.

[30] H. Hamda and M. Schoenauer. Topological optimum design with evolutionary algorithms. *J. Convex Anal.*, 9(2) :503–517, 2002. Special issue on optimization (Montpellier, 2000).

[31] S. B. Hamma. *Etude de méthodes numériques d'optimisation globale.* PhD thesis, Université Paul Sabatier, 1992.

[32] Q. Han, L.-Z. Liao, H. Qi, and L. Qi. Stability analysis of gradient-based neural networks for optimization problems. *J. Global Optim.*, 19(4) :363–381, 2001.

[33] S. Haykin. *Neural Networks, a comprehensive foundation.* Society for Industrial and Applied Mathematics (SIAM), New Jersey, 1999.

[34] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation.* Santa Fe Institute Studies in the Sciences of Complexity. Lecture Notes, I. Addison-Wesley Publishing Company Advanced Book Program, Redwood City, CA, 1991. With forewords by Jack Cowan and Christof Koch.

[35] G. E. Hinton. How neural networks learn from experience. *Scientific American*, pages 104–109, 1991.

[36] J. J. Hopfield and D. W. Tank. "Neural" computation of decisons in optimization problems. *Biol. Cybernet.*, 52(3) :141–152, 1985.

[37] K. Hornik, M. Stinchcombe, and White H. Multilayer feedforward networks are universal approximators. *Neural Networks*, pages 359–366, 1989.

[38] K. Hornik, M. Stinchcombe, and White H. Universal approxiamtion of an unknown mapping ans its deivatives using multilayer feedforward networks. *Neural Networks*, pages 551–560, 1990.

[39] K. Hornik, M. Stinchcombe, and White H. Approxiamtion capabilities of multilayer feedforward networks. *Neural Networks*, pages 251–257, 1991.

[40] J. F. Jodouin. *Les Réseaux de Neurones. Principes et définitions.* Hermes, 1994.

[41] M. Karakasis, A. Giotis, C. Kyriakos, and C. Giannakoglou. Multi-objective robust design optimisation of airfoil in transonic operating conditions. In *MACSInet*, 2003.

[42] C. T. Kelley. *Iterative methods for optimization*, volume 18 of *Frontiers in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[43] A. Kirsch. *An introduction to the mathematical theory of inverse problems*, volume 120 of *Applied Mathematical Sciences.* Springer-Verlag, New York, 1996.

[44] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited : a stepwise procedure for building and training a neural network. In *Neurocomputing (Les Arcs, 1989)*, volume 68 of *NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci.*, pages 41–50. Springer, Berlin, 1990.

[45] W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *J. Phys. A*, 20(11) :L745–L752, 1987.

[46] S. Lawrence, C. L. Giles, and A. C. Tsoi. Lessons in neural network training : overfitting may be harder than expected. In *Fourteenth Conferenceon Artificial Intelligence*, 1997.

[47] S. Lawrence, A. C. Tsoi, and A. D. Back. Function approximation with neural networks and local methods : bias, variance and smoothness. In *Australian Conference on Neural Networks*, 1996.

[48] M. Masmoudi. *Outils pour l'optimisation de forme*. PhD thesis, Thèse d'état de l'université de Nice, 1987.

[49] M. Masmoudi and C. Massat. A new global optimization algorithm based on automatic différentiation. Technical report, CERFACS, 1996.

[50] C. Massat. *Une nouvelle méthode d'optimisation globale basée sur la différentiation automatique*. PhD thesis, Thèse de l'université Paul Sabatier, 1997.

[51] W. S. Mc Culloc and W. Pitts. A logical calculuss of the ideas immanent in neurons activity. *Bulletin of Mathematical Biophysics*, pages 115–133, 1943.

[52] M. Mongeau, H. Karsenty, V. Rouzé, and J.-B. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optim. Methods Softw.*, 13(3) :203–226, 2000.

[53] P. Monk. Burger's equation. Technical report, University of Delaware, 1997.

[54] F. Muradore. *Optimisation de forme pour l'amélioration de la qualité d'essuyage des pare-brise*. PhD thesis, Université Paul Sabatier/VALEO, à paraître.

[55] F. Muyl. *Méthodes d'optimisation hybrides appliquées á l'optimisation de formes en aérodynamique automobile*. PhD thesis, Université Pierre Marie Curie, 2003.

[56] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999.

[57] M. Ohlsson, C. Peterson, and Bo Soderbers. Neural networks for optimization problems with inequality constraints the knapsack problem. *Neural Computation*, pages 331–339, 1993.

[58] D. P. O'Leary. Near-optimal parameters for Tikhonov and other regularization methods. *SIAM J. Sci. Comput.*, 23(4) :1161–1171 (electronic), 2001.

[59] J. C. Parikh and R. Pratap. An evolutionary model of a neural network. *J. Theoret. Biol.*, 108(1) :31–38, 1984.

[60] C. Peterson and Bo Soderbers. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, pages 3–22, 1989.

[61] M. Pillet. *Introduction aux plans d'expériences par ma méthode Taguchi*. Les Editions d'Organisation Université, 1992.

[62] C. Poloni and V. Pediroda. Ga coupled with computationally expensive simulations : tools to improve efficiency. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 267–288, 1998.

[63] L. B. Rall and G. F. Corliss. An introduction to automatic differentiation. In *Computational differentiation (Santa Fe, NM, 1996)*, pages 1–18. SIAM, Philadelphia, PA, 1996.

[64] J. J. Rousseau. Physique et simulation numérique. http ://www.univ-lemans.fr/enseignements/physique/02/, 2003.

[65] W. Sarle. Stopped training and other remedies for overfitting. In *27th Symposium on the Interface*, 1995.

[66] K. R. Shah and B. K. Sinha. Optimal designs with component-wise orthogonal row-column structures. *J. Combin. Inform. System Sci.*, 18(1-2) :68–78, 1993.

[67] K. R. Shah and B. K. Sinha. Row-column designs. In *Design and analysis of experiments*, volume 13 of *Handbook of Statist.*, pages 903–937. North-Holland, Amsterdam, 1996.

[68] K. Smith. Neural networks for combinatorial optimization : A review of more than a decade of research. *INFORMS Journal in Computing*, 1999.

[69] S. A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2(6) :625–639, 1988.

[70] E. Soria and A. J. Serrano. Redes neuronales : Una breve introduccion. EMESIS, www.uv.es/ soriae/charla.PD, 2001.

[71] K. T. Sun and H. C. Fu. A hybrid neural network model for solving optimization problems. *IEEE Trans. Comput.*, 42(2) :218–227, 1993.

[72] S. Thiria, Y. Lechevallier, O. Gascuel, and S. Canu. Statistique et Méthodes Neuronales. Dunod, 1997.

[73] M. Vigier. *Pratique des plans d'expériences : méthodologie Taguchi.* Les Editions d'Organisation Université, 1988.

[74] Ph. Wasserman. *Neural computing, theory and practice.* Van Nostrand Reinhold, 1989.

[75] Z.-H. Xin and H.-J. Zhang. Neural network and genetic algorithms for topology optimization of the CCS7 network. *Int. Trans. Oper. Res.*, 9(4) :427–436, 2002.