كلية العلوم و التقنيات فاس
ⵜⴰⵖⵉⵡⴰⵏⵜ ⵏ ⵜⵎⴰⵙⵙⴰⵏⵉⵏ ⴷ ⵜⵙⵏⵉⵅⵉⵜⵉⵏ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⵎⴰⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah

# Master Mathématique et Application au Calcul Scientifique

# (MACS)

## MEMOIRE DE FIN D'ETUDES

**Pour l'obtention du Diplôme de Master Sciences et Techniques (MST)**

# Optimisation non linéaire à variables mixtes

# Application au problème de classification par RNA

**Réalisé par : Soufyane MAJDOUB**

**Encadré par : Mohamed ETTAOUIL**

**Soutenu le 12/06/2018**

**Devant le jury composé de :**

| | |
|---|---|
| **- Ahmed ELHILALI ALAOUI :** | **Professeur à la FST de Fès** |
| **- Fatima EZZAKI :** | **Professeur à la FST de Fès** |
| **- Mohamed ETTAOUIL :** | **Professeur à la FST de Fès** |
| **- Rachid EL KHAOULANI :** | **Professeur à la FST de Fès** |

**Année Universitaire 2017 / 2018**

FACULTE DES SCIENCES ET TECHNIQUES FES – SAISS
B.P. 2202 – Route d'Imouzzer – FES
212 (0)5 35 61 16 86 – Fax : 212 (0)5 35 60 82 14
Site web : http://www.fst-usmba.ac.ma

# Table des matières


**1 Optimisation non linéaire continu** **9**
1.1 Introduction . . . 9
1.1.1 Conditions nécessaires et suffisantes d'optimalité (de base) . . . 9
1.1.2 Existence et unicité d'un minimum . . . 11
1.2 Optimisation non linéaire sans contraintes . . . 12
1.2.1 Conditions nécessaires d'optimalité . . . 12
1.2.2 Conditions nécessaires et suffisantes d'optimalité . . . 12
1.3 Résolution par méthodes de descente . . . 13
1.3.1 Méthodes de descente . . . 13
1.3.2 Méthodes de descente de gradient . . . 14
1.3.3 Méthode du gradient à pas optimal . . . 14
1.3.4 Méthode du gradient conjugué . . . 15
1.3.5 Méthode du gradient conjugué dans le cas général(méthode de Fletcher et Reeves) 18
1.4 Conditions nécessaires et suffisantes d'optimalité en présence de contraintes . . . 22
1.4.1 Conditions nécessaires de Kuhn et Tucker . . . 22
1.5 Méthodes numériques de résolutions . . . 23
1.5.1 Cas des contraintes d'égalité linéaires . . . 23
1.5.2 Cas des contraintes d'inégalité linéaires . . . 25
1.5.3 Méthodes de pénalisation . . . 26
1.5.4 Méthodes utilisant la notion de dualité . . . 29
1.6 Cas d'une fonction non partout différentiable ou non convexe . . . 31
1.6.1 Cas d'une fonction non partout différentiable . . . 32
1.6.2 Cas d'une fonction non convexe . . . 33
1.7 Conclusion . . . 34

**2 Optimisation non linéaire à variables mixtes** **35**
2.1 Généralités . . . 35
2.2 Relaxation lagrangienne . . . 35
2.2.1 Définitions et résultats généraux . . . 35
2.2.2 Dualité lagrangienne . . . 37
2.2.3 Saut de dualité . . . 38
2.2.4 La relaxation lagrangienne . . . 38
2.3 Conditions d'optimalité en variables mixtes . . . 40
2.4 Méthodes de résolution . . . 42
2.4.1 Méthode des décompositions de Benders . . . 42
2.4.2 Méthode utilisant un principe de pénalisation . . . 48
2.4.3 Déplacement dans l'espace des variables discrètes . . . 49
2.4.4 Méthodes d'énumération . . . 49
2.5 Conclusion . . . 54

**3 Réseaux neuronaux et apprentissage** **55**
3.1 Apprentissage automatique . . . . . 55
3.1.1 les secteurs d'application de L'apprentissage automatique . . . . . 56
3.2 Les grandes classes d'apprentissage . . . . . 56
3.2.1 Apprentissage supervisé . . . . . 56
3.2.2 Apprentissage non supervisé . . . . . 59
3.2.3 Apprentissage profond . . . . . 63
3.3 Réseaux de Neurones Artificiels (RNA) . . . . . 66
3.3.1 Différentes architectures neuronales . . . . . 68
3.4 Apprentissage du PMC . . . . . 71
3.4.1 Algorithme de Rétro-propagation du gradient . . . . . 71
3.4.2 Problème de surajustement et Méthodes pour le limiter . . . . . 73
3.5 Conclusion . . . . . 74

**4 Application au problème de classification** **76**
4.1 Le problème de classification . . . . . 76
4.1.1 Introduction . . . . . 76
4.1.2 Méthodes de classification . . . . . 76
4.1.3 Indicateurs de performance en classification . . . . . 76
4.1.4 Classification des données par le PMC . . . . . 78
4.1.5 Phase d'apprentissage . . . . . 82
4.1.6 Conclusion générale . . . . . 89

# Liste des tableaux

2.1 :Nombre de voisins discrets pour quelques valeurs de $d$. . . . . 42

4.1 Description des bases utilisées . . . . . 78
4.2 Description des ensembles d'apprentissage et de test utilisés . . . . . 82
4.3 L'effet de la variation du nombre de neurones de la couche cachée sur le temps d'apprentissage et le nombre d'itérations nécessaire pour pour converger avec une erreur $\epsilon$ dans le cas $\alpha = 1$ . . . . . 83
4.4 L'effet de la variation du nombre de neurones de la couche cachée sur le temps d'apprentissage et le nombre d'itérations nécessaire pour atteindre l'erreur $\epsilon$ dans le cas $\alpha = 0.5$ . . . . . 84
4.5 L'effet du pas d'apprentissage sur le temps d'apprentissage et la vitesse de convergence. 86
4.6 Taux de classification des trois base déjà décrites. . . . . 88

# Table des figures


2.1 Voisinage discret pour d=2 . . . . . 41
2.2 Allure de la fonction contrainte associée aux variables entières . . . . . 48

3.1 Exemple d'un dendrogramme illustrant la méthode CAH . . . . . 61
3.2 Extraction des caractéristiques et sélection des variables . . . . . 62
3.3 ACP sur des données linéaires . . . . . 62
3.4 ACP sur des données non linéaires . . . . . 63
3.5 Modèle d'un neurone biologique . . . . . 67
3.6 Différents types de fonctions d'activation pour le neurone artificiel . . . . . 67
3.7 Exemple de perceptron simple . . . . . 69
3.8 Exemple de perceptron multi-couche . . . . . 70

4.1 Matrice de confusion dans le cas binaire . . . . . 77
4.2 Courbe ROC . . . . . 78
4.3 Perceptron utilisé pour classifier la base d'Iris . . . . . 79
4.4 Perceptron utilisé pour classifier la base Post-Operative Patient . . . . . 81
4.5 Perceptron utilisé pour classifier la base Wine . . . . . 81
4.6 Les étapes nécessaires avant l'apprentissage . . . . . 82
4.7 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ k=20. . 83
4.8 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ et k=40. 84
4.9 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ et k=60. 84
4.10 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 20$. . . . . 85
4.11 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 40$. . . . . 85
4.12 L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 60$. . . . . 86
4.13 L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.01$ et pour $k = 20$. . . . . 87
4.14 L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.1$ et pour $k = 20$. . . . . 87
4.15 L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ et pour $k = 20$. . . . . 88
4.16 L'évolution de l'erreur moyenne en fonction du nombre de neurones de la couche cachée. 88

# Remerciement

"J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce mémoire."

En premier lieu, je remercie monsieur M.ETTAOUIL, professeur à la faculté des sciences et techniques Fes. En tant que directeur de mémoire, je le remercie de m'avoir proposer le sujet, m'a guider tout au long de ce travail et également de m'avoir aider à trouver des solutions pour avancer. Leurs précieux conseils et leurs commentaires m'ont permis d'améliorer grandement la qualité de mes travaux et de ce mémoire".

En deuxième lieu, je remercie vivement les membres du jury : Madame Fatima EZZAKI, professeur directrice de laboratoire LMCS à la Faculté des Sciences et Techniques de FES, Monsieur Rachid EL KHAOULANI, professeur à la Faculté des sciences et techniques de FES, Monsieur Ahmed ELHILALI ALAOUI, professeur à la Faculté des sciences et techniques de FES, qui m'ont honoré en acceptant d'évaluer et de juger mon travail.

Mes sincères remerciements sont également adressés aux chers collègues, les doctorants : Mr. Noureddine JOUDAR, Mr. Zakariae EN-NAIMANI, Mr. Hassan RAMCHOUN et Mlle KAOUTAR SENHAJI, qui m'ont apporté leurs soutiens pour m'aider á réaliser ce travail.

Je n'oublie pas de remercier ma famille, mes amis et surtout mes parents, qui m'ont toujours soutenu durant mes études.

# Introduction générale

L'optimisation est un sujet très ancien qui connait un nouvel essor depuis l'apparition des ordinateurs et dont les méthodes s'appliquent dans de très nombreux domaines : économie, gestion, intelligence artificielle( apprentissage automatique), planification, logistique, robotique, conception optimale, science de l'ingénieur, traitement de signal, etc.

L'optimisation est aussi un sujet très vaste qui touche aussi bien au calcul des variations, qu'à la recherche opérationnelle (domaine de l'optimisation des processus de gestion ou de décision, en passant par le contrôle optimale.

D'une certaine manière, l'optimisation peut être vue comme une discipline indépendante des autres disciplines telles que l'apprentissage automatique qu'on va traiter dans le troisième chapitre. Cependant, les interactions entre ces deux disciplines sont extrêmement nombreuses et fécondes et il est beaucoup plus naturel de les relier dans un même article. En effet, après l'étape de modélisation d'un problème de logistique ou un problème issu d'un phénomène physique ou d'un système industriel ou économique, après l'étape de simulation numérique sur ce modèle ; la démarche du mathématicien appliqué( qu'il soit ingénieur ou chercheur) ne s'arrête pas là : il lui faut souvent agir sur le phénomène ou sur le système afin d'en améliorer certaines performance. Cette troisième étape est celle de l'optimisation, c'est-à-dire celle de la minimisation (ou de la maximisation) d'une fonction qui dépend de la solution du modèle. par la suite nous mélangerons donc les exemples de problèmes d'optimisation où les modèles sont de nature très différence.

Dans le cas le plus simple, le modèle sera une simple équation algébrique et il s'agira simplement d'optimiser une fonction définie sur un espace de dimension finie (disons $\mathbb{R}^n$). Typiquement, c'est la situation la plus fréquente en recherche opérationnelle et en apprentissage automatique.

Une deuxième catégorie de problèmes correspond au cas où la fonction à optimiser dépend de la solution d'une équation différentielle ordinaire( autrement dit, cette fonction est définie sur un espace de dimension infinie, disons $\mathcal{C}[0,T]$). On parle alors de commande optimale, et les applications sont très nombreuses en automatique et en robotique.
La troisième et la dernière catégorie correspond à l'optimisation de fonctions de la solution d'une équation aux dérivées partielles. il s'agit alors de la théorie du contrôle optimal des systèmes distribués qui a de nombreuses applications, par exemple en conception optimale ou pour la stabilisation de structures mécanique.

On peut aussi séparer l'optimisation en deux grandes branches aux méthodes fortes différentes selon que les variables sont continue ou discrètes. Typiquement, si l'on minimise une fonction $F(x)$ avec $x \in \mathbb{R}$, il s'agit d'optimisation en variables continues, tandis que si $x \in \mathbb{Z}^n$ on a affaire à de l'optimisation combinatoire ou en variables discrètes.

Malgré les apparences, l'optimisation en variables continues est souvent plus "facile" que l'optimisation en variables discrètes car on peut utiliser la notion de dérivée qui est fort utile tant du point de vue théorique qu'algorithmique, l'optimisation combinatoire est naturelle et essentielle dans le nombreux problèmes de la recherche opérationnelle. C'est un domaine où, à coté de résultats théoriques rigoureux, fleurissent de nombreuses "heuristique" essentielles pour obtenir de bonnes performances algorithmiques.

Dans ce mémoire, en va présenter, dans un premier chapitre, les conditions nécessaires et suffisantes d'optimalité d'un problème d'optimisation continu en absence et en présence des contraintes suivis par des méthodes numériques de résolution. Dans le deuxième chapitre, on va découvrir qu'est ce que c'est un problème d'optimisation non linéaire à variables mixtes, puis, on va détailler quelques méthodes

de résolution de ce problème et en va citer des autres. Le troisième chapitre est consacré à la théorie de l'apprentissage automatique, les secteurs d'applications de cette dernière et les principaux types d'apprentissage, puis en va parler en gros à propos les réseaux de neurones artificiels et plus précisément le PMC (perceptron multicouches) avec son algorithme d'apprentissage (rétropropagation du gradient), ce modèle neural sera appliqué au problème de classification des données dans le quatrième chapitre, et au cours de cette application, on va tester la performance du PMC au niveau de la vitesse de convergence et l'influence de l'architecture du PMC sur la vitesse de convergence et sur la performance de ce dernier). Et on va terminer ce mémoire avec une conclusion générale et des perspectives.

# Chapitre 1

# Optimisation non linéaire continu

## 1.1 Introduction

Les problèmes d'optimisation non linéaire à variables continues s'écrivent en générale sous la forme :

$$\left\{\min_{x\in X} f(x)\right. \tag{1.1}$$

selon la nature de la fonction continue $f$ et l'ensemble $X$, on peut caractériser plusieurs types de problèmes, à savoir les problèmes d'optimisation convexe et non convexe, les problèmes d'optimisation différentiable et non différentiable et les d'optimisation non contraint et contraint. mais la forme la plus standard d'un problème d'optimisation non linéaire continue est celle donnée par () où $f$ est une fonction continue, convexe et différentiable, $X$ est convexe.
Dans ce chapitre, on va traiter juste cette forme standard des problèmes d'optimisation.

### 1.1.1 Conditions nécessaires et suffisantes d'optimalité (de base)

**Définitions et propriétés**

**Définitions**

1. Soit $X \in \mathbb{R}^n, x^* \in X$ et $f : X \to \mathbb{R}^n$
   On dit que $x^*$ est un point de minimum absolu (ou global) de $f$ sur $X$ si :
   $$f(x) \geq f(x^*), \forall x \in X$$
2. On dit que $x^*$ est un point de minimum relatif (ou local) de $f$ sur $X$, s'il existe $V$ voisinage de $x^*$ dans $\mathbb{R}^n$, tel que :
   $$f(x) \geq f(x^*), \forall x \in X \cap V$$
3. On dit que $x^*$ est un point de maximum absolu (respectivement relatif) de $f$ sur $X$, si $x^*$ est un point de minimum absolu (respectivement relatif) de $-f$ sur $X$.
4. On dit que $x^*$ est un point d'extremum absolu (respectivement relatif) de $f$ sur $X$, si $x^*$ est : soit un point de minimum absolu (respectivement relatif) de $f$ sur $X$, soit un point de maximum absolu (respectivement relatif) de $f$ sur $X$.

**Lemme** Soit $X \in \mathbb{R}^n$, $a \in \mathbb{R}^n$ et $x^*$ un élément appartenant à l'intérieur de $x$. Alors on a l'équivalence suivante :

$$< a, x - x^* > \geq 0, \forall x \in X \Leftrightarrow a = 0 \tag{1.2}$$

**Démonstration** L'implication inverse est évidente. Pour montrer l'implication directe, soit $w \in \mathbb{R}^n$ arbitraire, avec $w \neq 0$. Alors il existe $\theta_0 > 0$ telle que :

$$x^* + \theta w \in X, \forall \theta \in [-\theta_0, \theta_0]$$

(il suffit de prendre $\theta_0 = \frac{r}{2\|w\|}$, où $r > 0$ est tel que $B(x^*; r) \subset X$ ). Alors en prenant $x^* + \theta w$ à la place de $x$ dans $< a, x - x^* >$ on déduit

$$< a, \theta w > \geq 0, \forall \theta \in [-\theta_0, \theta_0]$$

En prenant d'abord $\theta = -\theta_0$ et ensuite $\theta = \theta_0$ dans l'inégalité précédente, on déduit

$$< a, w >= 0 \text{ et ceci quelque soit } w \in \mathbb{R}^n, w \neq 0 \tag{1.3}$$

Comme l'égalité précédente est évidemment valable aussi pour $w = 0$ alors elle est valable pour tout $w \in \mathbb{R}^n$. Ceci donne immédiatement le résultat en prenant $w = a$ dans la dernière égalité (1.2).

**Définition** Soit $X \subset \mathbb{R}^n$ un ensemble et $x^* \in X$. On dit que $w \in \mathbb{R}^n$ est une direction admissible pour $x^*$ en $X$ s'il existe $t_0 > 0$ tel que $x^* + tw \in X, \forall t \in [0; t_0]$.

**Exemples**

1. Si $x^* \in X$ alors tout vecteur $w \in \mathbb{R}^n$ est une direction admissible pour $x^*$ en $X$.
2. Si $X$ est convexe alors pour tout $v \in X$ le vecteur $v - x^*$ est une direction admissible pour $x^*$ en $X$. (Preuve : prendre $t_0 = 1$ et $w = v - x^*$ dans la définition précédente).

**Lemme** Soit $\Omega \subset \mathbb{R}^n$ un ouvert, $X \subset \Omega$ et $f : \Omega \to \mathbb{R}^n$ une fonction de classe $\mathcal{C}^1$ et $x^* \in X$ un point de minimum relatif de $f$ sur $X$. Soit $w \in \mathbb{R}^n$ une direction admissible pour $x^*$ en $X$. Alors

$$< \nabla f(x^*), w > \geq 0$$

**Démonstration** Soit $V$ un voisinage de $x^*$ en $\mathbb{R}^n$ tel que

$$f(x) \geq f(x^*), \forall x \in V \cap X$$

Soit $t_1 > 0$ tel que $x^* + tw \in V, \forall t \in [-t_1, t_1]$( pour $w \neq 0$ il suffit de prendre $t_1 = \frac{r}{2\|w\|}$ où $r > 0$ est tel que $B(x^*; r) \subset V$ ; pour $w = 0$ tout $t_1$ convient). D'autre part on a $x^* + tw \in X, \forall t \in [0, t_0]$ où $t_0$ est comme dans la définition d'une direction admissible, On déduit alors :

$$f(x^* + tw) \geq f(x^*), \forall t \in [0, \min\{t_0, t_1\}]$$

En divisant par $t$ et en passant à la limite pour $t \to 0$ ; $t > 0$, on obtient :

$$\frac{\partial f}{\partial w} \geq 0 \tag{1.4}$$

ce qui est exactement l'inégalité demandée.

**Conditions nécessaires d'un minimum relatif**

**Théorème** Soit $\Omega \subset \mathbb{R}^n$ un ouvert, $X \subset \Omega$ un ensemble convexe et $f : X \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$. Soit $x^* \in X$ un point de minimum relatif de $f$ sur $X$. Alors

$$< \nabla f(x^*), x - x^* > \geq 0, \forall x \in X \text{ (c'est l'inéquation d'Euler)}$$

Si en plus $x^*$ est dans l'intérieur de $X$ alors la condition est équivalente au

$$< \nabla f(x^*), x - x^* >= 0 \text{ (c'est l'équation d'Euler)[?]}$$

**Démonstration** C'est une conséquence immédiate des deux Lemmes précédent et du fait que $x - x^*$ est une direction admissible pour $x^*$ en $X$ ( car $X$ est convexe).

**conditions suffisantes d'un minimum relatif**

**Théorème** Soit $\Omega \subset \mathbb{R}^n$ un ouvert, $X \subset \Omega$ un ensemble convexe et $f : X \to \mathbb{R}$ une fonction de classe $\mathcal{C}^1$ et convexe. Soit $x^* \in X$. Alors les trois assertions suivantes sont équivalentes :

1. $x^*$ est un point de minimum absolu de $f$ sur $X$,
2. $x^*$ est un point de minimum relatif de $f$ sur $X$,
3. $< \nabla f(x^*), x - x^* > \geq 0, \forall x \in X$.

**Démonstration** très facile [**?**].

### 1.1.2 Existence et unicité d'un minimum

**Théorème( d'existence)** Soit $X \subset \Omega$ un ensemble non-vide et fermé et $f : \mathbb{R}^n \to \mathbb{R}$ continue. On suppose que : soit $X$ est borné, soit $X$ est non bornée et $f$ est une fonction coercive sur $X$. Alors le problème de minimisation sans contrainte admet au moins une solution sur $X$.

**Démonstration** On distingue deux cas :
**Cas 1)** L'ensemble $X$ est borné.
Alors comme $X$ est aussi fermé, $X$ est compact. Comme $f$ est continue, le Théorème de Weierstrass nous assure que $f$ est bornée sur $u$ et elle atteint ses bornes. Donc il existe au moins un point de minimum absolu de $f$ sur $X$.
**Cas 2)** L'ensemble $X$ est non borné. Soit $a \in X$ et considérons l'ensemble :

$$E = \{x \in X / f(x) \leq f(a)\} \text{ Remarquons que } a \in E$$

Alors on a :
$E$ est fermé ( car $E = f^{-1}]-\infty, f(a)]$, donc $E$ est l'image inverse d'un intervalle fermé par une fonction continue)
$E$ est borné. En effet, supposons le contraire : alors il existe une suite $x_k$ dans $E$ avec $\|x_k\| \to +\infty$ pour $k \to +\infty$. Comme $f$ est coercive sur $X$, ceci entraine $f(x_k) \to +\infty$ ce qui est absurde, car $f(x_k) \leq f(a); \forall k \in \mathbb{N}$ ) On déduit alors que $E$ est un ensemble compact dans $\mathbb{R}^n$.
Du Théorème de Weierstrass, $\exists x^* \in E$ tel que

$$f(x^*) \leq f(x), \forall x \in E$$

**Théorème( d'unicité)** Soit $X \subset \Omega$ un ensemble fermé et convexe et $f : \mathbb{R}^n \to \mathbb{R}$ strictement convexe. Alors il existe au plus un point de minimum de $f$ sur $X$.

**Démonstration** Supposons qu'ils existent $x_1 \neq x_2 \in X$ réalisant le minimum de $f$ sur $dom(f)$, par convexité de $X$, $\frac{x_1+x_2}{2} \in X$, et comme $f$ est strictement convexe, il suit :

$$\begin{aligned} f\left(\frac{x_1+x_2}{2}\right) &< \frac{1}{2}f(x_1) + \frac{1}{2}f(x_2) \\ &= \frac{1}{2}\min_{x\in dom(f)} f(x) + \frac{1}{2}\min_{x\in dom(f)} f(x) \\ &= \min_{x\in dom(f)} f(x) \end{aligned}$$

ce qui est impossible. D'où l'unicité.

**Corollaire (Existence et unicité)** Soit $X \in \mathbb{R}^n$ un ensemble fermé et convexe et $f : \mathbb{R}^n \to \mathbb{R}$ une fonction elliptique. Alors il existe un unique point de minimum de $f$ sur $X$.

**Démonstration** évidente [**?**].

## 1.2 Optimisation non linéaire sans contraintes

### 1.2.1 Conditions nécessaires d'optimalité

On va maintenant regarder de plus près le cas où $K = \mathbb{R}^n$, c'est à dire le problème sans contraintes $(P)$.

$$\min_{x \in \mathbb{R}^n} f(x) \qquad (P)$$

Dans le cas où $f$ est différentiable, on a le résultat suivant :

**Théorème** Si $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ différentiable et $\widehat{x}$ vérifiant

$$f(\widehat{x}) \leq f(x), \forall x \in \mathbb{R}^n$$

alors on a nécessairement

$$\nabla f(\widehat{x}) = 0$$

**Démonstration** Pour tout $t \in \mathbb{R}^*$ et pour tout $h \in \mathbb{R}^n$ on a :

$$f(\widehat{x}) \leq f(\widehat{x} + th)$$

on a donc ,

$$\lim_{t \to 0^+} \frac{f(\widehat{x}) - f(\widehat{x} + th)}{t} = \nabla^T f(\widehat{x}) h \leq 0$$

et

$$\lim_{t \to 0^-} \frac{f(\widehat{x}) - f(\widehat{x} + th)}{t} = \nabla^T f(\widehat{x}) h \geq 0$$

donc $\nabla^T f(\widehat{x}) h = 0, \forall h \in \mathbb{R}^n$ donc $\nabla f(\widehat{x}) = 0$ ( prendre par exemple $h = \nabla f(\widehat{x})$)

### 1.2.2 Conditions nécessaires et suffisantes d'optimalité

La condition de gradient nul devient suffisante dans le cas où $f$ est convexe[**?**] :

**Théorème** Soit $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ convexe et différentiable. Si $\widehat{x}$ vérifie

$$\nabla f(\widehat{x}) = 0$$

alors on a $f(\widehat{x}) \leq f(x), \forall x \in \mathbb{R}^n$

**Démonstration** Soient $x \in \mathbb{R}^n$ et $\lambda \in [0, 1]$. Puisque $f$ est convexe on a :

$$f(\lambda x + (1 - \lambda)\widehat{x}) \leq \lambda f(x) + (1 - \lambda) f(\widehat{x})$$

On retranche $f(\widehat{x})$ de chaque côté de l'inégalité, on note que

$$\lambda x + (1 - \lambda)\widehat{x} = \widehat{x} + \lambda(x - \widehat{x})$$

puis in divise par $\lambda$, ce qui donne l'inégalité

$$\frac{f(\widehat{x} + \lambda(x - \widehat{x})) - f(\widehat{x})}{\lambda} \leq f(x) - f(\widehat{x})$$

Et si on fait tendre $\lambda$ vers 0 on obtient

$$\nabla^T f(\widehat{x})(x - \widehat{x}) \leq f(x) - f(\widehat{x})$$

donc $0 \leq f(x) - f(\widehat{x})$ , $\forall x \in \mathbb{R}^n$.
Lorsque la fonction n'est pas convexe, on ne peut donner qu'une condition nécessaire et suffisante d'optimalité locale. On désignera par minimum local (que l'on oppose au minimum global) un vecteur vérifiant les conditions suivantes :

**Définition** On rappelle que $x^*$ minimum local de $f$, s'il existe $\delta > 0$ tel que :

$$f(x^*) \leq f(x), \forall x, \|x - x^*| \leq \delta$$

Dans le cas où $f$ est deux fois différentiable on peut alors donner le résultat suivant :

**théorème** Soit $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ deux fois différentiable. Si :

$$\begin{cases} \nabla f(x^*) = 0 \\ \nabla^2 f(x^*) > 0 \end{cases}$$

alors $x^*$ est un minimum local de $f$

**Démonstration** On a :

$$\begin{array}{rcl} f(x^* + th) & = & f(x^*) + t\nabla^T f(x^*)h + \frac{t^2}{2} h^T \nabla^2 f(x^*)h + t^2\|h\|^2\varepsilon(th) \\ & = & f(x^*) + \frac{t^2}{2} h^T \nabla^2 f(x^*)h + t^2\|h\|^2\varepsilon(h) \end{array}$$

On a donc pour $t > 0$ :

$$\frac{f(x^*) + th) - f(x^*)}{t^2} = \frac{1}{2} h^T \nabla^2 f(x^*)h + |h|^2\varepsilon(th) \tag{1.5}$$

Donc si $t$ est suffisamment petit, on aura bien $f(x^* + th) - f(x^*) > 0$ puisque $\nabla^2 f(x^*) > 0$

## 1.3 Résolution par méthodes de descente

### 1.3.1 Méthodes de descente

#### Principe des méthodes de descente

**Définition** Soit $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. On dira qu'un vecteur $d$ est une direction de descente en $x$, s'il existe $\bar{t} > 0$ tel que :

$$f(x + td) < f(x), t \in ]0, \bar{t}] \tag{1.6}$$

Le principe d'une méthode de descente consiste à faire les itérations suivantes :

$$x_{k+1} = x_k + t_k d_k, t_k > 0 \tag{1.7}$$

tout en assurant la propriété :

$$f(x_{k+1}) < f(x_k)$$

Le vecteur $d_k$ est la direction de descente en $x_k$. Le scalaire $t_k$ est appelé le pas de la méthode à l'itération $k$. On peut caractériser les directions de descente en $x_k$ à l'aide de gradient :

**Proposition** Soit $d \in \mathbb{R}^n$ vérifiant

$$\nabla^T f(x)d < 0 \tag{1.8}$$

alors $d$ est une direction de descente en $x$.

**Démonstration** On a pour $t > 0$

$$f(x + td) = f(x) + t\nabla^T f(x)d + t\varepsilon(t) \tag{1.9}$$

donc si on écrit :

$$\frac{f(x + td) - f(x)}{t} = \nabla^T f(x)d + \varepsilon(t)$$

on voit bien que pour $t$ suffisamment petit on aura $f(x + td) - f(x) < 0$
Dans la méthode le choix de $t_k$ est lié à la fonction

$$\varphi(t) = f(x_k + td_k)$$

en particulier, une façon de choisir $t_k$ peut être de résoudre le problème d'optimisation (à une seule variable)

$$\min_{t>0} \varphi(t)$$

Le pas $\widehat{t}_k$ obtenu ainsi s'appelle le pas optimal. La fonction $\varphi(t) = f(x_k + td_k)$ étant différentiable, on a alors nécessairement

$$\varphi'(\widehat{t}_k) = \nabla^T f(x_k + \widehat{t}_k d_k) d_k = 0$$

### 1.3.2 Méthodes de descente de gradient

**Principe des méthodes de descente gradient**

On cherche à déterminer la direction de descente qui fait décroitre $\varphi(t) = f(x + td)$ le plus vite possible( au moins localement). Pour cela on va essayer de minimiser la dérivée de $\varphi(t)$ en 0. On a :

$$\varphi'(0) = \nabla^T f(x) d$$

et en cherche $d$ solution du problème :

$$\min_{d \in \mathbb{R}^n, \|d\|=1} \varphi'(0)$$

La solution est bien sûr :

$$d = -\frac{\nabla f(x)}{\|\nabla f(x)\|} \tag{1.10}$$

en vertu de l'inégalité de Schwartz.
Il y a ensuite de nombreuses façon d'utiliser cette direction de descente. On peut par exemple utiliser un pas fixé a priori $t_k = \rho > 0, \forall k$. On obtient alors la méthode du gradient simple( dite à pas fixe) :

$$\begin{cases} d_k = -\nabla f(x_k) \\ x_{k+1} = x_k + \rho d_k \end{cases} \tag{1.11}$$

Sous certaines hypothèses de régularité ($f$ deux fois différentiable) cette méthode converge si $\rho$ est choisi assez petit.

### 1.3.3 Méthode du gradient à pas optimal

La méthode du gradient à pas optimal consiste à faire les itérations suivantes :

$$\begin{cases} d_k = -\nabla f(x_k) \\ x_{k+1} = x_k + t_k d_k \end{cases} \tag{1.12}$$

où $t_k$ est choisi de manière à ce que

$$f(x_k + t_k d_k) \leq f(x_k + td_k), \forall t > 0 \tag{1.13}$$

Cette méthode possède une propriété intéressante :

**Proposition** Soit $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction différentiable. Les directions de descente $d_k$ générées par la méthode vérifient :

$$d_{k+1}^T d_k = 0 \tag{1.14}$$

**Démonstration** Si on introduit la fonction $\varphi(t) = f(x_k + td_k)$, on a :

$$\varphi'(t) = \nabla^T f(x_k + td_k) d_k$$

et puisque $\varphi$ est dérivable on a nécessairement $\varphi'(t_k) = 0$ donc :

$$\nabla^T f(x_k + t_k d_k) d_k = \nabla^T f(x_{k+1}) d_k = -d_{k+1}^T d_k = 0$$

**Calcul du pas optimal dans le cas quadratique**

On a $f(x) = \frac{1}{2}x^T Ax - b^T x$, avec $A > 0$ et on note $\varphi(t) = f(x_k + td_k)$. Le pas optimal $t_k$ est caractérisé par :

$$\varphi'(t_k) = 0$$

On a donc :

$$\nabla^T f(x_k + t_k d_k)d_k = (A(x_k + t_k d_k) - b)^T d_k = 0$$

Soit :

$$(\nabla^T f(x_k) + t_k A d_k)d_k = 0$$

On obtient donc :

$$t_k = -\frac{\nabla^T f(x_k)d_k}{d_k^T A d_k}$$

qui est bien positif car $d_k$ est une direction de descente et $d_k^T A d_k > 0$ (car $A > 0$) La méthode du gradient à pas optimal peut donc s'écrire (dans le cas quadratique)

$$\begin{cases} d_k = b - Ax_k \\ t_k = \dfrac{d_k^T d_k}{d_k^T A d_k} \\ x_{k+1} = x_k + t_k d_k \end{cases} \tag{1.15}$$

### 1.3.4 Méthode du gradient conjugué

**Introduction**

**Directions conjuguées**

**Définition** Soit $A$ une matrice symétrique $n \times n$, définie positive. On dit que deux vecteurs $x$ et $y$ de $\mathbb{R}^n$ sont $A-$conjugués (ou conjugués par rapport à $A$) s'il vérifient

$$x^T Ay = 0$$

La matrice $A$ étant définie positive, la forme bilinéaire $a(x, y) = x^T Ay$ définit un produit scalaire et la relation traduit l'orthogonalité des vecteurs $x$ et $y$ pour ce produit scalaire.

**théorème** Si $\{d_0, d_1, ..., d_k\}$ sont des directions $A$-conjuguées deux à deux, soit

$$d_i^T A d_k = 0, \forall i, j, \text{avec } i < j \leq k \tag{1.16}$$

alors elles sont linéairement indépendantes.
Considérons maintenant dans $\mathbb{R}^2$ une méthode de descente appliquée à la minimisation d'une forme quadratique définie positive $f(x) = \frac{1}{2}x^T Ax - b^T x$

$$\begin{cases} x_1 = x_0 + \rho_0 d_0 \\ x_2 = x_1 + \rho_1 d_1 \end{cases} \tag{1.17}$$

avec $d_0$ et $d_1$ deux directions $A-$conjuguées et $\rho_0$ et $\rho_1$ déterminés de façon optimale. On a donc les relations suivantes :

$$\begin{cases} \nabla^T f(x_1)d_0 = (Ax_1 - b)^T d_0 = 0 \\ \nabla^T f(x_2)d_1 = (Ax_2 - b)^T d_1 = 0 \end{cases} \tag{1.18}$$

car $\rho_0$ et $\rho_1$ sont optimaux. Montrons que l'on a de plus :

$$\nabla^T f(x_2)d_0 = 0$$

On a :

$$\begin{aligned}\nabla^T f(x_2)d_0 &= (Ax_2 - b)^T d_0 \\ &= (A(x_1 + \rho_1 d_1) - b)^T d_0 \\ &= (Ax_1 - b)^T d_0 + \rho_1 d_1^T A d_0 \\ &= 0\end{aligned}$$

puisque $\nabla^T f(x_2)d_0 = \nabla^T f(x_2)d_1 = 0$ et $d_0, d_1$ linéairement indépendants, on a $\nabla f(x_2) = 0, x_2$ réalise donc le minimum de $f$ sur $R^2$ La relation de conjugaison permet donc à la méthode de descente de converger en deux itérations (dans le cas où $n = 2$).

**Définition** Soit $\{d_0, d_1, ..., d_n\}$ une famille de vecteur $A-$conjugués. On appelle alors méthode de directions conjuguées la méthode :

$$\begin{cases} x_0 \text{ donné} \\ x_{k+1} = x_k + \rho_k d_k, \rho_k \text{ optimal} \end{cases} \tag{1.19}$$

On va maintenant montrer la propriété vérifiée pour $n = 2$, à savoir $x_n = \widehat{x}$ réalise le minimum de $f(x) = \frac{1}{2}x^T Ax - b^T x$, est valable pour tout $n$.

**Lemme fondamental** On se donne a priori une famille $\{d_0, d_1, ..., d_n\}$ de directions conjuguées et on note :

$$E_k = Vect(d_0, d_1, ..., d_k - 1)$$

le sous-espace vectoriel engendré par les vecteurs $d_0, d_1, ..., d_k - 1$. Par construction, l'algorithme de directions conjugué

$$\begin{cases} x_0 \text{ donné} \\ x_{k+1} = x_k + \rho_k d_k \text{ optimal} \end{cases} \tag{1.20}$$

construit itérativement un vecteur $x_k$ vérifiant :

$$x_k \in x_0 + E_k \tag{1.21}$$

Voici l'énoncé du lemme fondamental[**?**] :

**Lemme** Le vecteur $x_k$ défini par l'algorithme réalise le minimum de $f(x) = \frac{1}{2}x^T Ax - b^T x$ sur le sous espace $x_0 + E_k$, c'est à dire $x_k \in x_0 + E_k$ et

$$f(x_k) \leq f(x), \forall x \in x_0 + E_k \tag{1.22}$$

Pour la démonstration de ce lemme nous aurons besoin du théorème suivant :

**théorème** Une condition nécessaire et suffisante pour que $x_k \in E_k + x_0$ réalise le minimum de $f(x) = \frac{1}{2}x^T Ax - b^T x$ sur le sous espace $x_0 + E_k$ est :

$$\nabla^T f(x_k)d_i = 0, \forall i = 0, ..., k - 1 \tag{1.23}$$

**Démonstration** Condition nécessaire : supposons que $f(x_k) \leq f(x), \forall x \in x_0 + E_k$. On a donc pour tout $t \in \mathbb{R}$ :

$$f(x_k) \leq f(x_k + td), \forall d \in E_k$$

On a donc soit :

$$\frac{f(x_k + td) - f(x_k)}{t} \geq 0, \text{ si } t > 0$$

soit

$$\frac{f(x_k + td) - f(x_k)}{t} \leq 0, \text{ si } t < 0$$

Si l'on fait tendre $t$ vers zéro, on en conclut que :

$$\nabla^T f(x_k)d = 0, \forall d \in E_k \tag{1.24}$$

donc en particulier $\nabla^t f(x_k)d_i = 0, \forall i = 0, ..., k - 1$. On admettra que la condition est suffisante.

**Démonstration du lemme fondamental** Pour $k = 1$ on a :

$$x_1 = x_0 + \rho_0 d_0, \tag{1.25}$$

avec $\rho_0$ optimal, c'est à dire $\nabla^T f(x_1)d_0 = 0$. Puisque $d_0 \in E_1$ la propriété est donc vérifiée pour $k = 1$. Supposons maintenant que la propriété est vérifiée à l'ordre $k$ :

$$\nabla^t f(x_k)d_i = 0, \forall i = 0, ..., k-1 \tag{1.26}$$

D'une part $\rho_k$ est optimal donc $\nabla^T f(x_{k+1}d_k = 0$. est optimal donc pour $0 \leq i < k$ :

$$\begin{array}{rcl} \nabla^T f(x_{k+1}d_i) & = & (A(x_k + \rho_k d_k) - b)^T d_i \\ & = & (Ax_k - b)^T d_i + \rho_k d_k^T A d_i \\ & = & 0 \end{array}$$

car $\rho_k$ est optimal et $d_k^T A d_i = 0$ (conjugaison). On a donc :

$$\nabla^T f(x_{k+1})d_i, \forall i = 0, ..., k \tag{1.27}$$

ce qui démontre le lemme fondamental.
Un corollaire direct est donc que la méthode de directions conjuguées converge en $n$ itérations au plus, puisque $E_{n-1} = \mathbb{R}^n$.

### Algorithme de la méthode du gradient conjugué

L'idée de la méthode est de construire itérativement des directions $d_0, ..., d_k$ mutuellement conjuguées. A chaque étape $k$, la direction $d_k$ est obtenue comme combinaison linéaire du gradient en $x_k$ et de la direction précédente $d_{k-1}$, les coefficients étant choisis de telle manière que $d_k$ soit conjuguée avec toutes les directions précédentes. Si l'on note $g_k = \nabla f(x_k)$, l'algorithme prend la forme suivante :
On se donne $x_0$ et on pose $d_0 = -g_0$

$$\begin{cases} x_{k+1} = x_k + \rho_k d_k, \text{ avec :} \\ \rho_k = -\frac{g_k^T d_k}{d_k^T A d_k} \\ d_{k+1} = g_{k+1} + \beta_k d_k, \text{ avec :} \\ \beta_k = \frac{g_{k+1}^T A d_k}{d_k^T A d_k} \end{cases} \tag{1.28}$$

Notons d'une part que la formule de $\rho_k$ définit bien le pas optimal : en effet, on a bien :

$$\nabla^T f(x_{k+1})d_k = g_k^T d_k + \rho_k d_k^T A d_k = 0$$

On va maintenant montrer que l'algorithme ci-dessus définit bien une méthode de directions conjuguées.

**Théorème** A une itération $k$ quelconque de l'algorithme où l'optimum n'est pas encore atteint, c'est à dire $g_k \neq 0$, on a :

$$\begin{array}{rcll} \rho_k & = & \frac{g_k^T g_k}{d_k T A d_k} & (1) \\ \beta_k & = & \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k} & (2) \\ & = & \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} & (3) \end{array}$$

et les directions $d_0, ..., d_{k+1}$ sont mutuellement conjuguées.

**Démonstration** On raisonne par récurrence sur $k$ en supposant que $d_0, ..., d_k$ sont mutuellement conjuguées.
-Montrons d'abord l'équivalence entre $\rho_k$ dans la formule (1.27) et (1).
Comme $d_0, ..., d_k$ sont mutuellement conjuguées $x_k$ réalise le minimum de $f$ sur $x_k + E_k$, on a : $g_k^T d_{k-1} = 0$, d'où :

$$g_k^T d_k = g_k^T(-g_k + \beta_k d_{k-1}) = -g_k^T g_k$$

Pour montrer (2), on note que :

$$g_{k-1} - g_k = A(x_{k+1} - x_k) = \rho_k A d_k \tag{1.29}$$

on a alors :

$$g_{k+1}^T A d_k = \frac{1}{\rho_k} g_{k+1}^T (g_{k+1} - g_k)$$

et en utilisant l'expression de $\rho_k$ il vient bien :

$$\beta_k = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k}$$

ce qui démontre (2).
On a de plus $g_{k+1}^T g_k = 0$ car $g_k = d_k - \beta_{k-1} d_{k-1}$ appartient à $E_{k+1}$ et que $g_{k+1}$ est orthogonal à ce sous-espace (les directions $d_0, ..., d_k$ sont conjuguées, par hypothèse de récurrence), ceci démontre (3).
Montrons maintenant que $d_{k+1}^T A d_i = 0$ pour $i = 0, ..., k$. On a d'une part :

$$d_{k+1}^T A d_k = (-g_{k+1} + \beta_k d_k)^T A d_k = 0$$

par définition de $\beta_k$. D'autre part, on a pour $i < k$ :

$$d_{k+1}^T A d_i = -g_{k+1}^T A d_i + \beta_k d_k^T A d_i$$

avec $d_k^T A d_i = 0$ en vertu de l'hypothèse de récurrence. On a ensuite, en utilisant la formule(1.28) :

$$g_{k+1}^T A d_i = \frac{1}{\rho_i} g_{k+1}^T (g_{i+1} - g_i)$$

et si l'on note que :

$$g_{i+1} - g_i = -d_{i+1} + (\beta_i + 1) d_i - \beta_{i-1} d_{i-1}$$

on a bien :

$$g_{k+1}^T (g_{i+1} - g_i) = 0$$

car $g_{k+1}^T d_{i+1} = g_{k+1}^T d_i = g_{k+1}^T d_{i-1} = 0$, en vertu du fait que $g_{k+1}$ est orthogonal à $E_{k+1}$ et que $i < k$. On a donc bien $d_{k+1}^T A d_i = 0$, ce qui achève la démonstration.

### 1.3.5 Méthode du gradient conjugué dans le cas général(méthode de Fletcher et Reeves)

La méthode de Fletcher et Reeves est une extension directe de la méthode du Gradient conjugué pour les fonction quelconques. Appliquée à une fonction quadratique, elle se comporte comme cette dernière : On se donne $x_0$ et on pose $d_0 = -\nabla f(x_0)$.

$$\begin{cases} x_{k+1} = x_k + \rho_k d_k, \text{ avec } \rho_k \text{ optimal} \\ d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k, \text{ avec} \\ \beta_k = \frac{\|\nabla f(x_{k+1})\|^2}{\|\nabla f(x_k)\|^2} \end{cases} \tag{1.30}$$

Cette méthode est intéressante car elle ne nécessite pas de stocker une matrice. Sa vitesse de convergence est très supérieure à celle de la méthode du gradient (ce point sera clarifié pour le cas quadratique dans le grain suivant).
La variante dite de Polak-Ribière consiste à définir $\beta_k$ par la formule(2). La convergence de la méthode de Fletcher-Reeves se démontre pour une classe assez large de fonctions $f$, ce qui est infaisable pour la variante de Polak-Ribière. Par contre on peut montrer que cette dernière converge plus rapidement (quand elle converge effectivement !), c'est donc la méthode qui est utilisée en général.
L'efficacité de la méthode du gradient conjugué repose essentiellement sur deux points :

— La recherche linéaire (détermination du pas optimal) doit être exacte,
— Les relations de conjugaison doivent être précises.

La recherche du pas optimal doit être réalisée à l'aide d'un algorithme spécifique puisque $f$ est quelconque. Par contre la notion de conjugaison n'a pas de sens dans le cas non-quadratique (sauf près de l'optimum, mais on ne le connaît pas. Il faut donc tester au cours des itérations si l'hypothèse d'approximation quadratique est vérifiée. On peut surveiller les indicateurs suivants :

— $|\nabla^T f(x_{k+1} \nabla f(x_k)|$ doit être petit,

— On doit avoir :

$$\frac{\nabla^T f(x_{k+1}) d_{k+1}}{\|\nabla f(x_{k+1})\| \|d_{k+1}\|} \leq -\alpha \tag{1.31}$$

avec $0 < \alpha \leq 0$ pas trop petit, c'est à dire que $d_{k+1}$ doit être une direction de descente « raisonnable ».
Dans le cas où ces conditions ne sont pas vérifiées, on rompt la conjugaison et on redémarre l'algorithme avec $d_{k+1} = -\nabla f(x_{k+1})$. On peut aussi décider de faire ce redémarrage arbitrairement toutes les $p$ itérations ($p$ fixé de l'ordre de $n$ par exemple).

**Interprétation de la méthode du gradient conjugué**

**Définition** On appelle kième sous-espace de Krylov associé à la matrice $A$ et au vecteur $g_0$ le sous espace

$$K_k = Vect(g_0, Ag_0, ..., A^{k-1}g_0) \tag{1.32}$$

Par construction, dans la méthode du gradient conjugué appliquée au cas quadratique, on a $E_k = K_k$ comme le montre le résultat suivant :

**Proposition** Dans la méthode du gradient conjugué on a :

$$E_k = Vect(d_0, d_1, ..., d_{k-1}) = Vect(g_0, Ag_0, ..., A^{k-1}g_0)$$

**Démonstration** Cette propriété est vérifiée à l'ordre $k = 1$ puisque $d_0 = -g_0$. Supposons qu'elle soit vérifiée à l'ordre $k$. On a alors la formule qui nous permet d'écrire :

$$\begin{array}{rcl} d_{k+1} & = & A(x_k + \rho_k d_k) - b + \beta_k d_k \\ & = & g_k + \rho_k A d_k + \beta_k d_k \\ & = & d_k - \beta_{k-1} d_{k-1} + \rho_k A d_k + \beta_k d_k \end{array}$$

ce qui permet de conclure que $d_{k+1} \in K_{k+1}$. La propriété est donc vérifiée pour tout $k > 0$.
Comme dans le cas de l'algorithme du gradient à pas optimal, nous choisissons maintenant de mesurer la distance séparant $x_k$ du vecteur $\widehat{x} = A^{-1}b$ à l'aide de la fonction définie par :

$$E(x) = \|x - \widehat{x}\|_A^2 = (x - \widehat{x})^T A (x - \widehat{x}) \tag{1.33}$$

Minimiser $E(x)$ est équivalent à minimiser $f(x) = \frac{1}{2}x^T Ax - b^T x$ comme le montre la proposition suivante :

**proposition** Soit $f(x) = \frac{1}{2}x^T Ax - b^T x$ une forme quadratique définie positive et $\widehat{x} = A^{-1}b$. On a :

$$E(x) = (x - \widehat{x})^T A (x - \widehat{x}) = f(x) + c$$

où $c$ est une constante.
On va maintenant illustrer d'un autre point de vue la convergence particulière de l'algorithme du gradient conjugué. Tout vecteur $x \in x_0 + E_k$ s'écrit :

$$x = x_0 + \sum_{j=0}^{k-1} \gamma_j A^j g_0$$

et comme $g_0 = Ax_0 - b = A(x_0 - \widehat{x})$ on a donc :

$$x - \widehat{x} = x_0 - \widehat{x} + \sum_{j=0}^{k-1} \gamma_j A^{j+1}(x_0 - \widehat{x}) = p(A)(x_0 - \widehat{x})$$

où le polynôme

$$p(z) = 1 + \sum_{j=0}^{k-1} \gamma_j z^{j+1}$$

est de degré $k$ et satisfait $p(0) = 1$. Puisque le vecteur $x_k$ obtenu à l'étape $k$ de l'algorithme du gradient conjugué vérifie

$$f(x_k) \leq f(x), \forall x \in E_k + x_0 \tag{1.34}$$

on a, en vertu du résultat démontré dans la proposition précédente,

$$E(x_k) = \|x_k - \widehat{x}\|_A^2 \leq \|p(A)(x_0 - \widehat{x})\|_A^2 \tag{1.35}$$

pour tout polynôme $p \in P_k$ vérifiant $p(0) = 1$.

**Convergence de la méthode du gradient conjugué**

Le résultat suivant va nous permettre de retrouver d'une autre manière la propriété de convergence finie de l'algorithme du G.C :

**proposition** Soit $A$ une matrice définie positive et $x_k$ le vecteur obtenu à l'étape $k$ de l'algorithme du G.C. Alors on a :

$$E(x_k) \leq E(x_0) \min_{p \in P_k, p(0)=1} \max_{z \in \sigma(A)} p(z)^2 \tag{1.36}$$

**Démonstration** Puisque la matrice $A$ est définie positive il existe une matrice orthogonale $U$ telle que : $A = UDU^T$ avec $D = diag(\lambda_1, ..., \lambda_n)$, où $\sigma(A) = \{\lambda_i\}_{i=1,...,n}$ sont les valeurs propres de $A$. Si on définit $A^{1/2} = UD^{1/2}U^T$ on a :

$$\|x\|_A^2 = \|A^{1/2}x\|^2$$

donc :

$$\|p(A)(x_0 - \widehat{x})\|_A^2 = \|A^{1/2}p(A)(x_0 - \widehat{x})\|^2 \leq \|p(A)\|^2 \|x_0 - \widehat{x}\|_A^2,$$

où on a utilisé la propriété que $p(A)$ et $A^{1/2}$ commutent (ces deux matrices ont les mêmes vecteurs propres). Puisque l'on a aussi $A_j = UD_jU^T$ les valeurs propres de $p(A)$ sont données par les nombres $p(\lambda_i)$ pour $i = 1, ..., n$, et donc :

$$\|p(A)\|^2 = \max_{i=1,...,n} p(\lambda_i)^2$$

On a donc bien :

$$E(x_k) \leq E(x_0) \min_{p \in P_k, p(0)=1} \max_{z \in \sigma(A)} p(z)^2$$

On a le corollaire suivant, qui permet d'exhiber le polynôme optimal $p(z)$pour $k = n$ :

**théorème** Soit $A$ une matrice définie positive. L'algorithme du G.C converge en $n$ itérations au plus. Plus précisément, si la matrice $A$ possède $k \leq n$ valeurs propres distinctes, alors l'algorithme du G.C converge en $k$ itérations au plus.

**Démonstration** Dans les deux cas possibles, notons :

$$\overline{p}(z) = \prod_{i=1}^{k} \frac{\lambda_i - z}{\lambda_i}$$

On a bien $\overline{p}(z)$ de degré $k, \overline{p}(0) = 1$ et par construction $\overline{p}(\lambda_i) = 0$ pour $i = 1, ..., k$. En vertu du résultat montré dans la proposition, on a donc :

$$E(x_k) = 0.$$

Soit $x_k = \widehat{x}$, la méthode du gradient conjugué étant en général utilisée comme une méthode itérative, il est intéressant de la comparer à la méthode du gradient à pas optimal. Le résultat suivant sera admis (la démonstration repose sur la détermination d'un polynôme particulier $p(z)$ solution d'un problème de moindre carrés [**?**]).

**théorème** Soit $A$ une matrice définie positive et $x_k$vecteur obtenu à l'étape k de l'algorithme du G.C. Alors on a :

$$E(x_k) \leq 4E(x_0)\left(\frac{\sqrt{\chi(A)}-1}{\sqrt{\chi(A)}+1}\right)^{2k}$$

où on a noté $\chi(A) = \lambda_n/\lambda_1$ le conditionnement de $A$ pour la norme euclidienne.
Pour l'algorithme du gradient à pas optimal on avait :

$$E(x_k) \leq E(x_0)\left(\frac{\chi(A)-1}{\chi(A)+1}\right)^{2k}$$

on voit donc que pour une même matrice $A$, la méthode du gradient conjugué convergera plus rapidement. Cependant cette estimation peut être très pessimiste car dans le cas où les valeurs propres sont groupées autour de valeurs distinctes, on peut être très proche du cas ou certaines valeurs propres sont multiples (et ou le nombre théorique d'itérations est inférieur à $n$) tout en ayant un mauvais conditionnement.

## 1.4 Conditions nécessaires et suffisantes d'optimalité en présence de contraintes

### 1.4.1 Conditions nécessaires de Kuhn et Tucker

On s'intéresse maintenant au problème suivant :

$$\begin{cases} \min_{x\in\mathbb{R}^n} f(x) \\ s.t \\ g(x) \leq 0 \end{cases} \qquad (Pc)$$

où $g : \mathbb{R}^n \to \mathbb{R}^m$, est différentiable (il n'y a ici aucune condition sur $m$). On notera $K$ l'ensemble des points admissibles, c'est à dire :

$$K = \{x \in \mathbb{R}^n : g(x) \leq 0\}$$

Au point solution de (Pc), il va de soi que les contraintes effectivement actives vérifieront $g_i(\widehat{x}) = 0$. Cependant, puisque l'on ne sait pas a priori quelles sont ces contraintes, le passage de (Pc) a un problème du type (Pce)( problème avec contraintes d'égalités) n'est pas direct.

**Définition** On appelle contraintes saturées en $\widehat{x}$ l'ensemble des indices $i$ tel que $g_i(\widehat{x}) = 0$, et on note :

$$I(\widehat{x}) = \{i : g_i(\widehat{x}) = 0\} \tag{1.37}$$

On note alors $S(\widehat{x})$ l'ensemble :

$$S(\widehat{x}) = \{x \in \mathbb{R}^n : g_i(\widehat{x}) = 0, \forall i \in I(\widehat{x}\}$$

Le concept de direction admissible se définit comme suit :

**Définition** On dit que $y \in \mathbb{R}^n$ est une direction admissible en $\widehat{x} \in K$ s'il existe $\alpha > 0$ et une courbe $x(t)$ vérifiant :

$$\begin{cases} x(t) \in K, \forall t \in [-\alpha, \alpha] \\ x(0) = \widehat{x} \\ \dfrac{dx}{dt}(0) = y \end{cases}$$

On notera alors $y \in C(\widehat{x})$

**Lemme** Soit $y \in \mathbb{R}^n$ une direction admissible en $\widehat{x} \in K$, alors on a nécessairement :

$$\nabla^T g_i(\widehat{x}) y \leq 0, \forall i \in I(\widehat{x})$$

**Démonstration** Considérons une courbe $x(t)$ définie pour $t \in [\alpha, \alpha]$ vérifiant :

$$\begin{cases} x(t) \in K, \forall t \in [-\alpha, \alpha], \alpha > 0 \\ x(0) = \widehat{x} \\ \dfrac{dx}{dt}(0) = y \end{cases}$$

Comme $g_i(\widehat{x}) < 0$ pour $i \notin I(\widehat{x})$, on aura toujours $g_i(x(t)) < 0$ pour $t$ suffisamment petit. Par contre, pour $i \in I(\widehat{x})$ on doit avoir $g_i(x(t)) \leq 0$ pour $t$ suffisamment petit. Si on utilise le développement de Taylor de $g_i(x(t))$ en $t = 0$ on doit donc avoir :

$$g_i(\widehat{x}) + t\nabla g_i^T(\widehat{x}) y + t\epsilon(t) \leq 0$$

Puisque $g_i(\widehat{x}) = 0$ il faut donc nécessairement que l'on ait :

$$\nabla g_i(\widehat{x})^T y \leq 0$$

**Définition** On dit que $\widehat{x}$ est un point régulier pour la contrainte $g(x) \leq 0$ si

— $g(\widehat{x}) \leq 0$

— Les vecteurs $\{\nabla g_i(\widehat{x})\}_{i \in I(\widehat{x})}$ sont linéairement indépendants.

Sous l'hypothèse de régularité de $\widehat{x}$ on aura :

$$\nabla g_i(\widehat{x})^T y \leq 0, i \in I(\widehat{x}) \Rightarrow y \in C(\widehat{x})$$

**Proposition** Soit $\widehat{x}$ la solution du problème (Pc). Il existe $\eta > 0$ tel que :

$$\forall x \in B(\widehat{x}, \eta), g_i(x) < 0, i \notin I(\widehat{x})$$

où on a noté $B(\widehat{x}, \eta)$ la boule de centre $\widehat{x}$ et de rayon $\eta$. Alors $\widehat{x}$ est la solution du problème :

$$\begin{cases} \min_{x \in B(\widehat{x},\eta)} f(x) \\ g_i(x) = 0, i \in I(\widehat{x}) \end{cases}$$

Ce résultat est uniquement dû à la continuité de $g$, et montre que l'on est localement ramené à un problème avec contraintes d'égalité [**?**]. On peut donc maintenant énoncer le résultat principal :

**Théorème** Soit $\widehat{x} \in K$ un point régulier solution du problème (Pc). Alors il existe un unique vecteur $\lambda \in \mathbb{R}^m$ tel que :

$$\begin{cases} \nabla f(\widehat{x}) + \sum_{i=1}^m \lambda_i \nabla g_i(\widehat{x}) = 0 \\ \lambda_i \geq 0, i = 1, ..., m \\ \lambda_i g_i(\widehat{x}) = 0, i = 1, ..., m \end{cases}$$

**Démonstration** [**?**]

## 1.5 Méthodes numériques de résolutions

### 1.5.1 Cas des contraintes d'égalité linéaires

#### Méthode du gradient projeté

On s'intéresse à un problème avec contraintes d'égalité linéaires :

$$\begin{cases} \min_{x \in \mathbb{R}^n} f(x) \\ s.t \\ Ax - b = 0 \end{cases} \qquad (Pce)$$

et nous ferons l'hypothèse que $A \in \mathcal{M}_{pn}(\mathbb{R})$ est de rang maximal. Une idée assez naturelle consiste à appliquer une méthode de descente qui prenne en compte la contrainte $Ax - b = 0$. Supposons que nous disposons d'un point $x_0 \in K = \{x \in \mathbb{R}^n, Ax - b = 0\}$. On peut chercher une direction de descente $d$ qui permette, au moins pour un petit déplacement, de rester dans $K$. Si on note $A_0 \in M_{pn}(\mathbb{R})$ la matrice composée des lignes $i \in I_0$ on doit donc avoir $A_0 d = 0($ en effet, on a $A_0(x + td)b = tA_0 d = 0, \forall t \in [-\alpha, \alpha]$, puisque $x \in K$, et donc les directions admissibles $d$ sont caractérisées par$A_0 d = 0)$. Or dans notre cas toutes les contraintes sont d'égalité. Donc, une direction admissible $d$ doit vérifier :

$$Ad = 0.$$

On peut chercher la meilleure direction de descente en résolvant le problème

$$\begin{cases} \min_{x \in \mathbb{R}^n} \nabla^T f(x) d \\ s.t \\ Ad = 0, \\ \|d\| = 1 \end{cases} \qquad (gp)$$

**Proposition** Le vecteur $d$ solution du problème (gp) est donné par $d = \frac{y}{\|y\|}$ où $y$ est la projection orthogonale de $\nabla f(x)$ sur $KerA$.

**Démonstration** On peut écrire que : $-\nabla f(x) = y + z$ où $y \in \ker A$ et $z \in \ker A^{\perp}$, ces deux sous-espaces étant complémentaires dans $\mathbb{R}^n$. On a donc :

$$\nabla^T f(x) d = -y^T d$$

Comme $d$ est un vecteur unitaire quelconque $y^T d$ sera maximal pour $d = \dfrac{y}{\|y\|}$, d'où le résultat. On remarquera que si $y \neq 0$, le vecteur d est bien une direction de descente.

**Algorithme du gradient projeté**

Pour former la matrice de projection sur $\ker A$ on utilise en général la factorisation $QR$ de la matrice $A^T$, qui s'exprime sous la forme :

$$A^T = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

où $R \in M_{pp}(\mathbb{R})$ est triangulaire supérieure et $Q \in \mathcal{M}_{nn}(\mathbb{R})$ est orthogonale, et se décompose en $Q = [UV]$, où les colonnes de $U \in \mathcal{M}_{n,p}(\mathbb{R})$ forment une base orthogonale de $ImA^T$ et les colonnes de $V \in \mathcal{M}_{n,n-p}(\mathbb{R})$ une base orthogonale de $(ImA^T)^{\perp} = \ker A$. Dans ce cas la matrice de la projection orthogonale sur $\ker A$ s'écrit :

$$P = I - UU^T = VV^T$$

**Remarque** Dans l'algorithme que nous allons étudier, la matrice de projection peut être calculée une fois pour toutes puisque $A$ est donnée. Cependant, pour les problèmes avec contraintes d'inégalité linéaires, on sera amené à considérer une succession de problèmes avec contraintes d'égalité, et la matrice $A$ pourra évoluer à chaque itération, par ajout ou suppression d'une ligne. Le choix de la factorisation $QR$ est tout indiqué car il existe des techniques de mise à jour particulièrement économiques, ce qui n'est pas le cas quand on exprime la matrice P sous la forme classique

$$P = I - A^T \left[AA^T\right] A$$

La méthode du gradient projeté consiste tout simplement à mettre en œuvre une méthode de descente utilisant à chaque pas la direction $d_k = VV^T \nabla f(x_k)$. Les itérations sont poursuivies jusqu'à ce que $d_k = 0$.
Cela signifie alors que $\nabla f(x) \in ImA^T$ et donc qu'il existe $\lambda$ tel que :

$$\nabla f(x_k) = -A^T \lambda$$

On peut utiliser la factorisation de $A^T$ pour obtenir $\lambda$ par résolution du système linéaire

$$R\lambda = -U^T \nabla f(x)$$

**Algorithme :** ( gradient projeté)

**1 : Poser $k = 0$ et choisir $x_0$ admissible**
2 : Calculer la projection $d_k = VV^T \nabla f(x_k)$
3 : Si $d_k = 0$

- Calculer $\lambda = R^1 U^T \nabla f(x_k)$
- Arrêter les itérations.

4 : Déterminer $\rho_k > 0$ réalisant le minimum de $f(x_k + \rho d_k)$
5 : Poser $x_{k+1} = x_k + \rho_k d_k$, faire $k \longleftarrow k + 1$ et retourner en 2.

### 1.5.2 Cas des contraintes d'inégalité linéaires

**Méthode de directions réalisables**

On s'intéresse maintenant à un problème avec contraintes d'inégalités linéaires

$$\begin{cases} \min_{x\in\mathbb{R}^n} f(x) \\ s.t \\ Ax - b \leq 0 \end{cases}$$

On peut essayer de voir comment adapter la stratégie de l'algorithme du gradient projeté. Supposons que nous disposons d'un point initial admissible $x_0 \in K = \{x \in \mathbb{R}^n, Axb \leq 0\}$. Notons $I_0$ l'ensemble des indices des contraintes saturées, soit

$$I_0 = i : A_i x_0 b_i = 0$$

On peut chercher une direction de descente d qui permette, au moins pour un petit déplacement, de rester dans $K$. Si on note $A_0 \in \mathcal{M}_{pn}(\mathbb{R}$ la matrice composée des lignes $i \in I_0$ on doit donc avoir

$$A_{I_0} d = 0$$

Après calcul de la factorisation $\begin{pmatrix} U & V \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$ de $A_{I_0}^T$, une direction admissible $d$ peut être obtenue par $d_k = VV^T \nabla f(x_k)$
Il y a ensuite deux cas à envisager :

1. Si $d \neq 0$, il faut déterminer le déplacement maximal autorisé par les contraintes non saturées, c'est à dire $\rho_{max}$ tel que :

   $$\rho_{max} = \{\rho : \rho \geq 0, A_i(x_0 + \rho d)b_i \leq 0, i \notin I_0\}$$

   Ensuite, on cherche le pas optimal $\rho_{opt}$ dans direction $d$. Ce pas pouvant faire sortir du domaine admissible, on prendra donc toujours :

   $$= \min(\rho_{opt}, \rho_{max})$$

   en notant bien que lorsque $\rho = \rho_{max}$, cela signifie qu'une nouvelle contrainte sera saturée.
2. Si $d = 0$ cela signifie que $\nabla f(x) \in ImA_{I_0}^T$ et donc qu'il existe $\lambda$ tel que :

   $$\nabla f(x) = ImA_{I_0}^T \lambda$$

   et qui s'obtient par résolution du système linéaire

   $$R\lambda = U^T \nabla f(x)$$

   et il faut ensuite considérer deux cas
   — Si $\lambda \geq 0$, alors x satisfait les condition de Kuhn et Tucker. Le point $x$ est donc un optimum local du problème.
   — Sinon, on supprime dans $I_0$ une des contraintes pour lesquelles $\lambda < 0$ (par exemple la plus négative). On obtient alors une nouvelle matrice $A_1$ qui permet de déterminer une nouvelle direction de descente en $x_0$. On peut ensuite poursuivre les itérations. On peut donc résumer l'algorithme de la façon suivante :

**Algorithme :** ( gradient projeté)

**1 : Poser** $k = 0$ **et choisir** $x_0$
2 :déterminer $I_k = \{i : A_i x_k - b_i = 0\}$
3 : Former la matrice Si $A_{I_k} = \{A_i\}_{i \in I_k}$
4 : Calculer ou mettre à jour la factorisation

$$A_{I_k}^T = \begin{pmatrix} U_k & V_k \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{1.38}$$

5 :Calcule la projection $d_k = -V_k V_k^T \nabla f(x_k)$
6 :Si $d_k = 0$
- Calculer $\lambda = -(R_k)^{-}1 U_k^T \nabla f(x_k)$
- Si$\lambda \geq 0$ alors on s'arrête
- Sinon, choisir $j$ tel que $\lambda_j \leq \lambda_i, \forall_i$, faire $I_k = I_k - \{j\}$ et retourner en 3

7 :Calculer $\rho_{max} = \{\rho : \rho \geq 0, A_i\,(x_k + \rho d_k)\,a - b_i \leq 0, i \notin I_k\}$
8 :Déterminer $\rho_k$ réalisant le minimum de $f(x_k + \rho d_k)$ sur $[0, \rho_{max}]$
9 :Poser $x_{k+1} = x_k + \rho_k d_k$, faire $k \longleftarrow k + 1$ et retourner en 2 .

### 1.5.3 Méthodes de pénalisation

La pénalisation est un concept simple qui permet de transformer un problème d'optimisation avec contraintes en un problème ou en une suite de problèmes d'optimisation sans contrainte. C'est un concept qui a une utilité à la fois théorique et numérique.

#### Méthode de pénalisation externe

On considère un problème avec contraintes d'inégalité non-linéaires :

$$\begin{cases} \min_{x \in \mathbb{R}^n} f(x) \\ s.c \\ g(x) \leq 0 \end{cases} \qquad (Pc)$$

Le but des méthodes de pénalisation est de résoudre (Pc) de façon approchée, de la façon suivante. On définit la fonction (dite fonction de pénalisation) $\varphi(x)$ par :

$$\varphi(x) = \sum_{i=1}^{m} (g_i^+(x))^2$$

où $[.]^+$ est la fonction partie positive définie par :

$$y^+ = max(0, y).$$

Cette fonction modifie la fonction objectif à l'extérieur du domaine réalisable, si on note $K = \{x \in \mathbb{R}^n, g(x) \leq 0\}$, la fonction $\varphi$ vérifie par construction :

$$\begin{cases} \varphi(x) = 0, \text{ pour } x \in K \\ \varphi(x) > 0, \text{ pour } x \notin K \end{cases}$$

On introduit alors le problème $(P_\epsilon)$

$$\begin{cases} \min_{x \in \mathbb{R}^n} f_\epsilon(x) \\ f_\epsilon(x) = f(x) + \frac{1}{\epsilon}\varphi(x) \end{cases} \qquad (P_\epsilon)$$

dont on notera $x_\epsilon$, la solution, vérifiant

$$f_\epsilon(x_\epsilon) \leq f_\epsilon(x) \forall x \in \mathbb{R}^N \tag{1.39}$$

**Algorithme :** (Algorithme de la méthode de pénalisation )

**1 :choisir** $x_0, \epsilon_1 = 0$ **et poser** $k = 1$
2 :Trouver $x_k$ solution de problème $\min_{x\in\mathbb{R}^n} f_{\epsilon_k}(x)$ en partant de $x_{k-1}$
3 :poser $\epsilon_{k+1} = \frac{\epsilon_k}{100}$
4 :faire $k \longleftarrow k+1$ et retourner en 2

Le but de ce terme additionnel est de pénaliser la violation des contraintes (dans le cas pénalisation extérieur). Le nom de pénalité extérieure provient du fait que $x_\epsilon$ est toujours à l'extérieur ( au sens large) de $K$ comme le montre le résultat suivant :

**Proposition** S'il existe au moins une contrainte saturée à l'optimum $\widehat{x}$ du problème (Pc) alors le vecteur solution du problème pénalisé $P_\epsilon$ vérifie nécessairement

$$\exists i_0, g_{i0}(x_\epsilon) \geq 0 \tag{1.40}$$

**Démonstration** Montrons la contraposée : si $g_i(x_\epsilon) < 0, \forall i$ on a par définition $x_\epsilon \in K$ Puisque

$$f_\epsilon(x_\epsilon) \leq f_\epsilon(x), \forall x \in \mathbb{R}^n \tag{1.41}$$

donc en particulier pour $x = \widehat{x}$, on a :

$$f_\epsilon(x_\epsilon) \leq f_\epsilon(\widehat{x})$$

mais comme $x_\epsilon \in K$ et $\widehat{x} \in K$, on a :

$$\varphi(x_\epsilon) = \varphi(\widehat{x}) = 0$$

et donc

$$f(x_\epsilon) \leq f(\widehat{x}) \tag{1.42}$$

D'où $x_\epsilon = \widehat{x}$. On a donc $g_i(\widehat{x}) < 0, \forall i$ et aucune contrainte n'est saturée en $\widehat{x}$.
En générale, on a toujours $x_\epsilon \notin K$ , mais sous des hypothèses assez peu restrictives, $x_\epsilon$ tend vers une solution du problème (Pc) quand $\epsilon$ tend vers 0.

**Théorème** Soit $\varphi : \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction de pénalisation extérieure vérifiant :
- $\varphi(x) \geq 0$
- $\varphi(x) = 0 \Leftrightarrow x \in K$
- $\varphi$ continue

On suppose d'autre part que $f$ est continue, que $K$ est fermé et que l'une des deux conditions suivantes est vérifiée :
- $f(x) \longrightarrow +\infty$ quand $\|x\| \longrightarrow \infty$
- $K$ est borné et $\varphi(x) \longrightarrow +\infty$ quand $\|x\| \longrightarrow +\infty$
- $\varphi$ continue

Alors, quand $\epsilon_k$ tend vers 0, la suite $x_{\epsilon_k}$ admet au moins un point d'accumulation qui est alors une solution optimale du problème (Pc). Lorsqu'on met en œuvre cette méthode de façon pratique, on ne peut pas prendre tout de suite $\epsilon_k$ très petit, à cause des problèmes de conditionnement que cela peut causer. On commence donc avec une valeur du type $\epsilon 0 = 1$, et chaque solution $x_{\epsilon_k}$ est prise comme vecteur initial pour résoudre le problème avec $\epsilon_{k+1} = \frac{\epsilon_k}{100}$ ( par exemple). On peut bien sûr utiliser n'importe quelle méthode pour résoudre le problème $min_x f_{\epsilon_k}(x)$ (gradient conjugué, ...).

**Méthode de pénalisation interne**

Dans certains problèmes, le fait que les itérés $x_\epsilon$ générés par pénalisation extérieure ne soient pas admissibles peut être un inconvénient, par exemple, parce que f n'est pas définie à l'extérieur de K. On peut introduire des méthodes de pénalisation dans lesquelles les itérés $_\epsilon$ restent dans K. On parle alors

de pénalisation intérieure. L'idée est d'utiliser un terme de pénalisation qui tend vers l'infini lorsque x s'approche de la frontière $\partial K$ de $K$.
Dans le cas des méthodes internes, en général, $x_\epsilon$ n'est jamais dans $K$ (sauf cas particulier) : cela peut poser des problèmes si par exemple la fonction $f$ n'est pas définie hors de $K$. Les méthodes internes permettent d'éviter cet inconvénient. Leur principe est le même que pour les méthodes externes : on considère une fonction :

$$f_\epsilon(x) = f(x) + \epsilon\psi(x) \tag{1.43}$$

mais ici la fonction $\psi(x)$ est définie pour $x \in K$ et est de type

$$\psi(x) = \sum_{i=1}^{m} \frac{1}{g_i(x)^2} \tag{1.44}$$

puisque l'on a $\psi(x) \longrightarrow \infty$ quand on s'approche de la frontière de $K$ , on qualifie souvent $\psi$ de fonction barrière. Les propriété de convergence sont les même que pour les méthodes externes, mais il faut ici disposer d'un $x_0 \in K$, ce qui peut être difficile dans certains cas.

**Estimation des multiplicateurs**

Les méthodes de pénalisation ne sont en général jamais utilisées pour obtenir la solution du problème avec contraintes, car cela nécessiterait d'utiliser des paramètres de pénalisation beaucoup trop petits. En revanche, elles permettent de calculer des estimations correctes des multiplicateurs. Pour les méthodes externes, le point $x_k$ est solution du problème min $f_{\epsilon_k}(x)$ où

$$f_\epsilon(x) = f(x) + \frac{1}{\epsilon}\sum_{i=1}^{m}[g_i^+(x)]^2 \tag{1.45}$$

et vérifie donc les conditions d'optimalité :

$$\nabla f(x_k) + \frac{2}{\epsilon}\sum_{i=1}^{m} g_i^+(x_k)\nabla g_i(x_k) = 0$$

Sous les hypothèses du théorème précédent $x_k \to x$ et donc pour les contraintes non saturées, puisque $g_i(\widehat{x}) < 0$, il existe $k_0$ tel que

$$k > k_0 \Rightarrow g_i(x_k) < 0, i \notin I(\widehat{x})$$

Si on suppose que $\widehat{x}$ est régulier, les conditions de Kuhn et Tucker sont vérifiées et on a

$$\nabla f(\widehat{x}) + \sum_{i\in I} \lambda_i \nabla g_i(\widehat{x}) = 0$$

Si on note maintenant que pour $k > k_0$ :

$$\nabla f(x_k) + \frac{2}{\epsilon}\sum_{i\in I} g_i^+(x_k)\nabla g_i(x_k) = 0 \tag{1.46}$$

alors par continuité de $\nabla f$ et $\nabla g$ on en déduit que pour $i \in I$

$$\lim_{k\to+\infty} \frac{2}{\epsilon} g_i^+(x_k) = \lambda_i$$

On peut bien sûr faire le même type de raisonnement pour la méthode de pénalité interne.

### 1.5.4 Méthodes utilisant la notion de dualité

**Éléments sur la dualité( dualité Lagrangienne)**

On s'intéresse ici aux problèmes avec contrainte d'inégalité du type

$$\begin{cases} \min_{x\in\mathbb{R}^n} f(x) \\ g(x) \leq 0 \qquad (P) \end{cases}$$

et on note comme d'habitude $K = \{x \in \mathbb{R}^n, g(x) \leq 0\}$. Le problème (P) est appelé problème primal par opposition au problème dual que l'on va maintenant définir. Soit $\varphi(x)$ une fonction indicatrice de $K$ :

$$\begin{cases} \varphi(x) = 0, \text{ Si } x \in K \\ \varphi(x) + \infty, \text{ sinon} \end{cases}$$

Alors le problème primal est équivalent à :

$$\min_{x\in\mathbb{R}^n} f(x) + \varphi(x)$$

On peut construire la fonction $\varphi$ de la façon suivante :

$$\varphi(x) = \max_{\lambda\geq 0} \lambda^T g(x) = \max_{\lambda\geq 0} \sum_{i=1}^{m} \lambda_i g_i(x)$$

On peut vérifier que la fonction ainsi définie a bien les caractéristiques d'une fonction indicatrice de $K$ : si $x \in K$ on a $g_i(x) \leq 0$ et donc $\lambda^T g(x) \leq 0$, la max est donc atteint pour $\lambda = 0$, si $x \notin K$ il existe $j$ tel que $g_j(x) > 0$, et donc $\lambda^T g(x)$ peut être rendu arbitrairement grand en faisant tendre $\lambda_j$ vers $+\infty$ Le problème primal est donc équivalent au problème :

$$\min_{x\in\mathbb{R}^n} \left( f(x) + \max_{\lambda\geq 0} \lambda^T g(x) \right)$$

et si on utilise le lagrangien $L(x, \lambda) = f(x) + \lambda^T g(x)$, on peut alors noter que le problème primal s'écrit :

$$\min_{x\in\mathbb{R}^n} \max_{\lambda\geq 0} L(x, \lambda)$$

**Définition** On appelle problème dual du problème. le problème

$$\max_{\lambda\geq 0} \min_{x\in\mathbb{R}^n} L(x, \lambda)$$

et appelle $w(\lambda) = \min_{x\in\mathbb{R}^n} L(x, \lambda)$ la fonction duale.

**Proposition** La fonction duale est concave.

**Démonstration** Soient $\lambda_1 \geq 0, \lambda_2 \geq 0, \theta \in [0, 1]$ et $\lambda = \theta\lambda_1 + (1 - \theta)\lambda_2$, il existe $x_1, x_2$ et $x$ tel que :

$$\begin{cases} w(\lambda_1) = L(x_1, \lambda_1) \\ w(\lambda_2) = L(x_2, \lambda_2) \\ w(\lambda) = L(x, \lambda) \end{cases}$$

On a donc par définition de la fonction duale :

$$\begin{cases} w(\lambda_1) \leq L(x, \lambda_1) \\ w(\lambda_2) \leq L(x, \lambda_2) \end{cases}$$

Si on multiplie la première inéquation par $\theta$ et la deuxième par $(1 - \theta)$ il vient :

$$\theta w(\lambda_1) + (1 - \theta)w(\lambda_2) \leq f(x) + [\theta\lambda_1 + (1 - \theta)\lambda_2]^T = g(x) = w(\lambda) \tag{1.47}$$

Ce qui est remarquable dans cette propriété est que le résultat ne suppose absolument rien sur la convexité des fonctions $f$ et $g_i$.

**Point-col du lagrangien**

On montre facilement la proposition suivante :

**proposition** on a :

$$\max_{\lambda\geq 0}\{\min_{x\in\mathbb{R}^n} L(x,\lambda)\} \leq \min_{x\in\mathbb{R}^n}\{\max_{\lambda\geq 0} L(x,\lambda)\}$$

**Démonstration** On a $L(x,\lambda) \leq \max_{\lambda\geq 0} L(x,\lambda)$ et donc par définition de $w(\lambda)$ :

$$w(\lambda) \leq \min_{x\in\mathbb{R}^n}\max_{\lambda\geq 0} L(x,\lambda)$$

On a donc :

$$\max_{\lambda\geq 0} w(\lambda) \leq \min_{x\in\mathbb{R}^n}\max_{\lambda\geq 0} L(x,\lambda)$$

Ce qui montre le résultat :

$$\min_{x\in\mathbb{R}^n}\max_{\lambda\geq 0} L(x,\lambda) = f(\widehat{x})$$

où $\widehat{x}$ est la solution du problème primal, on a donc :

$$\max_{\lambda\geq 0} w(\lambda) \leq f(\widehat{x})$$

Alors s'il existe bien un maximum de la fonction duale atteint pour $\lambda = \overline{\lambda}$, la valeur $w(\overline{\lambda})$ est un minorant de $f(\widehat{x})$ et il existe un point $x(\overline{\lambda})$ tel que :

$$w(\overline{\lambda}) = L(x(\overline{\lambda}),\overline{\lambda}) \leq f(\widehat{x})$$

Le théorème suivant précise dans quelles conditions on a $x(\overline{\lambda}) = \widehat{x}$

**théorème** S'il existe un couple $(\widehat{x},\widehat{\lambda})$ tel que :

$$L(\widehat{x},\lambda) \leq L(\widehat{x},\widehat{\lambda}) \leq L(x,\widehat{\lambda}), \forall x\in\mathbb{R}^n, \forall\lambda\in\mathbb{R}^m$$

alors $\widehat{x}$ est une solution du problème primal et $\widehat{\lambda}$ est le multiplicateur de Kuhn et Tuker associé. Un point vérifiant cette propriété est appelé un point-col du lagrangien. On a dans ce cas :

$$L(\widehat{x},\widehat{\lambda}) = \max_{\lambda\geq 0} w(\lambda) = \min_{x\in K} f(x)$$

Lorsque ce point existe, on peut donc résoudre le problème dual à la place du problème primal : l'intérêt principal est la concavité de la fonction duale ainsi que la simplicité des contraintes. On voit aussi que même lorsqu'il n'existe pas de point col, le maximum de la fonction duale fournit un minorant de $f(\widehat{x})$, ce qui peut être utile dans certaines circonstances. On appelle alors la différence $f(\widehat{x}) - w(\widehat{\lambda})$ le saut de dualité.

**théorème** Si $f$ est strictement convexe, si les $g_i$ sont convexes et si $K$ est d'intérieur non-vide, l'existence de $\widehat{x}$ est équivalente de $\widehat{\lambda}$ et on a

$$w(\widehat{\lambda}) = L(\widehat{x},\widehat{\lambda}) = f(\widehat{x})$$

Il existe cependant des cas où il existe un point-col et les conditions précédentes ne sont pas vérifiées. Quand il n'y a pas de point-col, on peut faire alors appel à des techniques où on utilise un lagrangien augmenté du type

$$L(x,\lambda,r) = f(x) + \lambda^T g(x) + r\sum_{i=1}^{m}(g_i^+(x))^2$$

pour définir la fonction duale. Ce type d'approche permet de généraliser les méthodes duales pour les cas typiquement non-convexes.

**Algorithme :** (Algorithme d'uzawa )

**1 : poser** $k = 0$ **et** $\lambda_0 = 0$
2 : Déterminer $x_k$ solution du problème $\min_{x\in\mathbb{R}^n} f(x) + \lambda_k^T g(x)$
3 : Si $\max_i g_i(x_k) < \epsilon$ alors on s'arrête.
4 : Sinon, calculer $\lambda_{k+1} = [\lambda_k + \rho_k g(x_k)]^+$
5 : Faire $k \longleftarrow k+1$ et retourner en 2.

**Méthode d'Uzawa**

Le principe de la méthode d'Uzawa est d'utiliser la méthode du gradient pour maximiser la fonction duale, tout en tenant compte de la contrainte $\lambda \geq 0$, cela donne la méthode :

$$\lambda_{k+1} = [\lambda_k + \rho_k \nabla w(\lambda_k)]^+$$

L'utilisation de cette méthode suppose que la fonction duale est différentiable (au moins a l'optimum). Ce sera le cas si le minimum en $x$ de $L(x, \widehat{\lambda})$ est unique. Dans ce cas si on note $x(\lambda)$ le vecteur tel que :

$$w(\lambda) = L(x(\lambda), \lambda)$$

on peut écrire que :

$$\begin{aligned} \nabla w(\lambda) &= \nabla_x L(x(\lambda), \lambda) \frac{dx(\lambda)}{d\lambda} \\ &= g(x(\lambda)) \end{aligned}$$

puisque $x(\lambda)$ est par définition le minimum en $x$ de $L(x, \lambda)$ L'algorithme de la méthode est donc le suivant :

Au point 4 on peut choisir $\rho_k$ fixe ou bien faire une recherche linéaire. Lorsque la fonction duale est mal conditionnée, on peut aussi utiliser une méthode de quasi-Newton. Dans le test d'arrêt choisi la valeur de $\epsilon > 0$ devra être choisie prudemment : en effet, s'il n'existe pas de point-col on ne peut avoir $x_k \in K$ et donc si $\in$ est trop petit l'algorithme ne s'arrêtera pas.

**Méthode d'Arrow et Hurwicz**

Cette méthode est très voisine de la méthode d'Uzawa. Au lieu de déterminer $x_k$ comme le minimum de $L(x, \lambda_k)$ on se contente d'un pas dans la direction $-\nabla_x L(x, \lambda_k)$ : on définit $x_{k+1}$ par

$$x_{k+1} = x_k - \alpha_k \nabla_x L(x_k, \lambda_k) \tag{1.48}$$

et $\lambda_{k+1}$ par

$$\lambda_{k+1} = [\lambda_k + \rho_k g(x_k)]^+ \tag{1.49}$$

## 1.6 Cas d'une fonction non partout différentiable ou non convexe

On rencontre de nombreuses situations en Recherche Opérationnelle dans lesquelles on doit optimiser une fonction non partout différentiable. En général, il s'agit de minimiser une fonction convexe de type " max " ou de maximiser une fonction concave de type " min ". Les techniques de Programmation Non Linéaire pour les problèmes d'optimisation différentiable sont alors inopérantes et des méthodes adaptées doivent être mises au point pour ces situations. Dans cette partie, nous nous intéressons essentiellement à la classe des fonctions convexes. Ces hypothèses requièrent un outil spécial : le sous-différentiel de l'analyse convexe, noté $\partial J(x_0)$. Les fonctions convexes ont des propriétés remarquables qui permettent d'analyser plus facilement leurs propriétés et aussi de les minimiser efficacement. De nombreux problèmes non convexes irrésolvables peuvent être approchés par des problèmes convexes( qu'on va voir aussi dans cette partie) qui eux sont presque systématiquement minimisés en des temps rapides [**?**].

**Algorithme 1 :** algorithme de sous gradient dans le cas non contraint

**Imput :**
$N$ : un nombre d'itération
$x_0$ : un point de départ

**Output :**
$x_N$ : une solution approchée

**Algorithme :**
début
pour $k$ allant de 0 à $N$ faire
**calculer** $s_k \in \partial f(x_k)$
**recherche linéaire** : calculer un pas $\tau_k$ tel que

$$f(x_k - \tau_k \frac{s_k}{\|s_k\|}) < f(x_k)$$

**mise à jour**

$$x_{k+1} = x_k - \tau_k \frac{s_k}{\|s_k\|}$$

fin pour
fin

### 1.6.1 Cas d'une fonction non partout différentiable

D'une manière générale, les méthodes pour l'optimisation non différentiable s'inspirent largement des méthodes différentiables.

#### Méthodes de sous-gradient

Les méthodes de sous-gradients sont une généralisation naturelle des méthodes de gradient au cas non différentiable en remplaçant le gradient par un sous-gradient arbitraire. Ces méthodes (appelées méthodes de Kiev) ont été introduites dans les années 60 par N.Z.Shor pour la minimisation sans contrainte de fonctions convexes, et l'on trouve les premiers résultats de convergence dans les travaux de Y.M. Ermoliev [**?**].

#### Algorithme de sous-gradient

Sachant que le sous-différentiel n'est souvent pas connu dans son intégralité, supposons que l'on dispose de l'oracle suivant :
Pour tout $x \in \mathbb{R}^n$, on sait calculer la valeur $f(x)$ de la fonction objectif $f$ en $x$ et au moins un sous-gradient $s \in \partial f(x)$.
Cet oracle fournira dans les algorithmes des candidats à la descente (et non pas nécessairement des directions de descente).
Dans cette partie, on s'intéresse aux méthodes de premier ordre appliquées à la classe de problèmes suivante :

$$\min_{x \in X \subset \mathbb{R}^n} f(x)$$

où $X \subset \mathbb{R}^n$ est un ensemble convexe, $f : X \to \mathbb{R}^n$ est convexe et $f$ est Lipschitz de constante $L$ sur $X$, avec $X = epi(f)$.
On note $x^*$ un minimiseur du problème déjà défini. Notons que ce problème n'a pas de raison particulière d'admettre un minimiseur unique.

#### Règles pour choisir les pas

Plusieurs types différents de règles de taille de pas sont utilisés.

— Pas a taille constant $\alpha_k = h$ indépendant de $k$.
— Pas a longueur $\frac{h}{\|s_k\|}$ constante, avec $h = \|x^{k+1} - x^k\|$
— Règles des séries divergentes mais de carré-sommables $i.e$

$$\sum_{k=0}^{\infty} \alpha_k = \infty \text{ et } \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

— Règles des séries divergentes $i.e$

$$\sum_{k=0}^{\infty} \alpha_k = \infty \text{ et } \lim_{k \to +\infty} \alpha_k = 0$$

Si l'une des conditions suivantes est satisfaite, alors la méthode de sous-gradient converge.

**Remarque** La méthode de sous gradient n'est pas une méthode de descente.

### 1.6.2 Cas d'une fonction non convexe

**Transformée de Fenchel ou conjuguée convexe**

La fonction conjuguée, aussi appelée transformée de Fenchel ou transformée de Legendre- Fenchel est utilisée pour :
— convexifier une fonction (en calculant la bi-conjuguée, i.e. la conjuguée de la conjuguée).
— calculer le sous-différentiel d'une fonction convexe.
— calculer des problèmes dits " duaux ", en optimisation. Ces problèmes apportent souvent beaucoup d'information sur les problème " primaux ", i.e. ceux que l'on souhaite résoudre
Elle a été introduite par Mandelbrojt en 1939 puis précisée et améliorée par Fenchel en 1949. Cette transformée généralise la transformation de Legendre (1787) [**?**].

**Définition (Conjuguée convexe)** Soit $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ une fonction.
Sa conjuguée est définie par :

$$f^*(s) = \sup_{x \in \mathbb{R}^n} < s, x > -f(x)$$

L'application $f \to f^*$ est appelée transformation de Lengendre-Fenchel. La fonction $f^*$ est appelée conjuguée convexe, transformée de Fenchel ou transformée de Legendre- Fenchel de $f$. La motivation pour introduire cette transformation est la suivante. On peut définir la convexifiée fermée d'une fonction comme l'enveloppe supérieure de toutes les minorantes affines de $f$. Parmi toutes les minorantes affines $x \to < s, x > +\alpha$, on ne peut garder que celle qui est la plus haute, c'est-à-dire qui a le plus grand $\alpha$. Il faut donc déterminer le plus grand $\alpha$ tel que $\forall x \in \mathbb{R}^n$ :

$$< s, x > +\alpha \leq f(x)$$

La plus petite valeur de $-\alpha$ est donnée par :

$$f^*(s) = \sup_{x \in \mathbb{R}^n} (< s, x > -f(x))$$

Maintenant, on va citer sans démonstration des propriétés intéressantes s'appuient sur la convexification des fonctions

**Propositions**

1. Pour toute fonction $f$ , $f^*$ est convexe, fermée.
2. Pour tout $(x, s) \in \mathbb{R}^n \times \in \mathbb{R}^n$

$$f^*(s) + f(x) \geq < s, x >$$

**Définition** Soit $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ une fonction.
Sa biconjuguée est définie par :

$$f^{**}(s) = \sup_{x \in \mathbb{R}^n} < s, x > -f^*(x)$$

**Théorèmes**

1. La biconjuguée de $f$ est la plus grande fonction convexe fermée inférieure à $f$.
   On peut aussi la voir comme la fonction dont l'épigraphe est l'enveloppe convexe fermée de $epi(f)$.
2. Soit $f$ une fonction convexe fermée. On a Pour tout $(x, s) \in \mathbb{R}^n \times \in \mathbb{R}^n$ :

$$s \in \partial f^*(x) \Leftrightarrow x \in \partial f^* * (s)$$

**Proposition** La biconjuguée de $f$ satisfait $f^{**} = f$ si et seulement si $f$ est convexe fermée.

## 1.7 Conclusion

Dans ce chapitre, dans un premier temps, nous avons cité quelques théorèmes d'existence et d'unicité ainsi que les conditions nécessaires et suffisantes d'optimalité dans le cas continu, en faisant l'appel au théorème de Weistrass qui constitue un outil très puissant pour démontrer la convergence des méthodes. Dans un deuxième temps, nous avons traité les conditions d'optimalités des fonctions différentiables dans les deux cas : non contraint et contraint, puis, nous avons donné quelques méthodes numériques de recherche de cet optimum, ces méthodes sont souvent des méthodes de descentes , à savoir : des méthodes de résolution des problèmes d'optimisation différentiable dans le cas non contraint, comme la méthode du gradient à pas fixe, la méthode du gradient à pas optimal, la méthode des gradients conjugués, la méthode de Fletcher et Reeves qui est une généralisation de la méthode des gradient conjugué pour les fonctions non quadratique et la méthode de Polak-Ribière qui est une variante de la dernière, et des des méthodes de résolution des problèmes d'optimisation différentiable dans le cas contraint telles que la méthode des gradients projetés pour les contraintes linéaires, les méthodes de pénalisation, les méthodes duales( méthode d'Uzawa, méthode d'Arrow et Hurwicz) pour des problèmes à contraintes quelconque.
A la fin du chapitre, nous avons donné des techniques de résolution dans le cas des problèmes d'optimisation non différentiable ou non convexe, à savoir l'utilisation du sous différentiel de l'analyse convexe qui est un outil très puissant dans ce cas, puis la convexification des fonction non convexe à travers la transformée de Fenchel ou la biconjugué convexe d'une fonction.
On conclut que les problèmes d'optimisation continu admettent plusieurs méthodes de résolution, ces méthodes sont puissantes, faciles à appliquer et rendent ces problèmes faciles à résoudre.

# Chapitre 2

# Optimisation non linéaire à variables mixtes

## 2.1 Généralités

Les problèmes d'optimisation en variables mixtes, aussi appelés problèmes d'optimisation mixte, conjuguent les attributs des problèmes d'optimisation combinatoire et continue. Les problèmes d'optimisation continue consistent à minimiser (ou maximiser) une fonction sur un domaine continu. Des algorithmes polynomiaux existent pour de nombreux problèmes de cette famille. Les problèmes d'optimisation combinatoire consistent à minimiser une fonction sur un domaine discret. Bien que décrit de façon concise, ce domaine (aussi appelé ensemble des solutions) est souvent de très grande taille. Les problèmes d'optimisation mixte comportent des variables de décision discrètes et continues. Si les contraintes et l'objectif du problème peuvent être exprimés de manière linéaire en fonction de ses variables de décision, alors on parle de programmation linéaire mixte.
Ces problèmes d'optimisation en variables mixtes apparaissent dans divers domaines industriels. La planification des arrêts de centrales nucléaires [9][10][11] en est un exemple dans le secteur de l'énergie. Ce problème consiste à ordonnancer les arrêts de maintenance des centrales nucléaires et à déterminer un plan de production de toutes les centrales thermiques du parc, sur un horizon donné. Comme autres exemples de problèmes d'optimisation mixte à forts enjeux économiques, nous pouvons citer : planification prévisionnelle de maintenance des raffineries [12], optimisation de réseaux de conduites pétrolières[13] , optimisation du placement des puits en ingénierie de réservoir[13] , optimisation du chargement de porte containers[14], construction des rotations d'avions avec affectation des passagers [15]. La plupart des approches proposées pour résoudre ces problèmes de grande taille se basent sur des décompositions, qui font l'objet d'une section dans ce chapitre.

## 2.2 Relaxation lagrangienne

Un cadre particulièrement intéressant pour la relaxation, est celui donné par la dualité lagrangienne. Son principe consiste à ajouter des termes d'une inégalité dans la fonction objective en ajoutant un multiplicateur visant à pénaliser la fonction objective si la solution trouvée ne vérifie pas cette inégalité. Cette relaxation fournit fréquemment de très bonne valeur de relaxation, bien souvent meilleure que la relaxation continue.

### 2.2.1 Définitions et résultats généraux

On considère ici un programme mathématique $PM$ (qui peut être discret) écrit de la façon suivante :

$$\begin{cases} \min f(x) \\ g_i(x) \leq 0, i \in I \\ x \in S \subset \mathbb{R}^n \end{cases} \tag{2.1}$$

On associe à chaque contrainte $i \in I$ une valeur réelle $\lambda_i \geq 0$ que l'on appelle multiplicateur de Lagrange. On considère également le vecteur réel $\lambda = (\lambda_i, i \in I)$. On pose alors la fonction de Lagrange

$$L(x, \lambda) = f(x) + \sum_{i \in I} \lambda_i g_i(x) \tag{2.2}$$

qui est une fonction qui a pour paramètres les deux vecteurs $x$ et $\lambda$ et prend ses valeurs dans $R$. On suppose ici que le $PM$ est tel que le problème $\min_{x \in S}\{L(x, \lambda)\}$ a une solution optimale pour tout $\lambda \geq 0$.

**Définition** Soient $\overline{x} \in S$ et $\overline{\lambda} \geq 0$, on dit que le couple $(\overline{x}, \overline{\lambda})$ est un point-selle de la fonction de Lagrange si

$$\begin{cases} L(\overline{x}, \lambda) \leq L(\overline{x}, \overline{\lambda}), \forall \lambda \geq 0 \\ L(\overline{x}, \overline{\lambda}) \leq L(x, \overline{\lambda}), \forall x \in S \end{cases} \tag{2.3}$$

En d'autre terme, un point-selle $(\overline{x}, \overline{\lambda})$ est tel que la fonction $L$ pour $x$ fixé à $\overline{x}$ atteind un maximum en $\overline{\lambda}$ et que la fonction $L$ pour $\lambda$ fixé à $\overline{\lambda}$ atteind un minimum en $\overline{x}$. Afin de caractériser les points-selles , on a le résultat suivant :

**théorème** Étant donnés des vecteurs $\overline{x}$ et $\overline{\lambda} \geq 0$, le couple $(\overline{x}, \overline{\lambda})$ est un point-selle de la fonction de Lagrange si et seulement si

i) $L(\overline{x}, \overline{\lambda}) = \min_{x \in S}\{L(x, \overline{\lambda})\}$
ii) $g_i(\overline{x}) \leq 0, \forall i \in I$
iii) $\overline{\lambda}_i g_i(\overline{x}) = 0, \forall i \in I$

**Démonstration** ($\Rightarrow$)la condition $i)$ se déduit par construction d'un point-selle. Par définition, on sait que $\forall \lambda \geq 0, L(\overline{x}, \lambda) \leq L(\overline{x}\overline{\lambda})$, ce qui peut s'écrire également

$$\sum_{i \in I} \lambda_i g_i(\overline{x}) \leq \sum_{i \in I} \overline{\lambda}_i g_i(\overline{x}), \forall \lambda \geq 0 \qquad (*) \tag{2.4}$$

Supposons que $ii)$ soit faux et qu'il existe $i_0 \in I$ tel que $g_{i_0}(\overline{x}) > 0$, dans ce cas, en prenant $\lambda_{i_0}$très grand, on ne vérifierait pas $(*)$, une contradiction. Donc $ii)$ est vérifié. En appliquant $(*)$ avec $\lambda = 0$, on obtient $\sum_{i \in I} \overline{\lambda}_i g_i(\overline{x}) \geq 0$, or comme par définition $\overline{\lambda}_i \geq 0$ et $g_i(\overline{x}) \leq 0$ pour tout $i \in I$, on obtient donc $iii)$. ($\Leftarrow$)Par la propriété $i)$, on sait que pour tout $x \in S, L(\overline{x}, \overline{\lambda}) \leq L(x, \overline{\lambda})$, c'est-à-dire une partie de la définition d'un point-selle. Par $ii)$, pour tout $\lambda \geq 0, L(\overline{x}, \lambda) = f(\overline{x}) + \sum_{i \in I} \lambda_i g_i(\overline{x}) \leq f(\overline{x})$. De plus, par $iii)$, comme $\overline{\lambda}_i g_i(\overline{x}) = 0, \forall i \in I$, on obtient que $L(\overline{x}, \overline{\lambda}) = f(\overline{x})$. Par conséquent, $L(\overline{x}, \lambda) \leq f(\overline{x}) = L(\overline{x}, \overline{\lambda})$. Donc $(\overline{x}, \overline{\lambda})$ est un point-selle pour $L$. Le théorème suivant permet de faire le lien entre le point-selle et l'optimum du problème mathématique discret $PMD$.

**Théorème** Étant donné un point-selle $(\overline{x}, \overline{\lambda})$ pour $L$ défini par les vecteurs $\overline{x}$ et $\overline{\lambda} \geq 0$, alors $\overline{x}$ est un optimum global pour le $PMD$.

**Démonstration** En appliquant le théorème précédent, on sait par $i)$ que pour tout $x \in S$

$$f(\overline{x}) + \sum_{i \in I} \overline{\lambda}_i g_i(\overline{x}) \leq f(x) + \sum_{i \in I} \overline{\lambda}_i g_i(x) \tag{2.5}$$

Or par $iii)$, on obtient

$$f(\overline{x}) \leq f(x) + \sum_{i \in I} \overline{\lambda}_i g_i(x) \tag{2.6}$$

et par $ii)$, comme de plus $\lambda \geq 0$, on obtient finalement que $f(\overline{x}) \leq f(x)$ pour tout $x \in S$. Ces théorèmes permettent ainsi d'étudier la fonction de Lagrange pour résoudre le $PM$. Cette relaxation ramène parfois à des problèmes d'optimisation continue en ôtant des contraintes difficiles. Néanmoins, cette étude n'est possible que pour les $PM$ où il existe un point-selle pour $L$ : or il n'existe pas de point-selle pour les $PMD$ ! On va néanmoins pouvoir utiliser la fonction de Lagrange dans le contexte de la dualité lagrangienne.

### 2.2.2 Dualité lagrangienne

On considère la fonction, dite duale lagrangienne, suivante :

$$w(\lambda) = \inf_{x \in S} \{L(x, \lambda)\}, \forall \lambda \geq 0 \tag{2.7}$$

On définit alors le problème dual (lagrangien) du $PM$ :

$$\begin{cases} \max w(\lambda) = \max_{\lambda}\{\min_{x \in S} L(x, \lambda)\}) \\ \lambda \subset \mathbb{R}+^{m} \end{cases} \tag{2.8}$$

En fait, ce problème dual correspond au fait de rechercher un point-selle pour $L$. On peut néanmoins noter que ce problème est défini même s'il n'existe pas de point-selle (comme c'est le cas pour un $PMD$). On a alors les théorèmes suivants de la dualité.

**Théorème (Théorème de la dualité faible)** Si l'on note $f(x^*)$ l'optimum global de $f$ et $w(\lambda^*)$ l'optimum global du problème dual, alors pour tout $\lambda \geq 0, w(\lambda) \leq w(\lambda^*) \leq f(x^*)$.

**Démonstration** Soit $\lambda \geq 0$. Par définition de la fonction duale, on a $w(\lambda) \leq f(x) + \sum_{i \in I} \lambda_i g_i(x)$, pour tout $x \in S$. Comme $\lambda g(x) \leq 0$ pour tout $x \in S, w(\lambda^*) \leq f(x)$. En particulier, pour $x^*$ et $\lambda^*$ solutions optimales, on a donc $w(\lambda) \leq w(\lambda) \leq f(x^*)$.

**théorème(Concavité de la fonction duale)** La fonction duale $\lambda \mapsto w(\lambda)$ est une fonction concave de $\lambda$.

**Démonstration** Soit $\lambda^1$ et $\lambda^2$ deux vecteurs réels positifs et soit $t \in [0, 1]$. Posons alors $\lambda = t\lambda^1 + (1-t)\lambda^2$. Pour ce $\lambda$ donné, on sait par hypothèse qu'il existe un $\overline{x}$ tel que

$$w(\lambda) = f(\overline{x}) + \sum_{i \in I} \lambda_i g_i(\overline{x})$$

D'autre part, par définition de $w$ appliquée à $\lambda^1$ et $\lambda^2$, on a

$$\begin{cases} w(\lambda^1) \leq f(\overline{x}) + \sum_{i \in I} \lambda_i^1 g_i(\overline{x}) \\ w(\lambda^2) \leq f(\overline{x}) + \sum_{i \in I} \lambda_i^2 g_i(\overline{x}) \end{cases}$$

Par combinaison linéaire des deux inégalités, on obtient donc

$$tw(\lambda^1) + (1-t)w(\lambda^2) \leq f(\overline{x}) + \sum_{i \in I} (t\lambda_i^1 + (1-t)\lambda_i^2) g_i(\overline{x})$$

Par conséquent,

$$tw(\lambda^1) + (1-t)w(\lambda^2) \leq f(\overline{x}) + \sum_{i \in I} \lambda_i g_i(x) = w(\lambda)$$

Ainsi la fonction $w$ est concave, ce qui permet d'utiliser des outils efficaces de l'optimisation continue. Il est important de noter qu'aucune hypothèse n'a été faite sur la nature des fonctions $f$ et $g_i$ qui peuvent être quelconque, ni sur la convexité de l'ensemble $S$. Dans le cas des ensembles $S$ discrets, la fonction $f$ n'est pas différentiable en tout point mais la concavité de $w$ permet cependant de montrer que tout optimum local de $w$ est global : ce qui fait que le problème dual est en général plus facile à résoudre que le problème dual. Enfin, ce dernier théorème indique que, si le $PM$ admet un point-selle, optimiser $w$ revient à optimiser $f$.

**théorème (Théorème de la dualité)** Si le problème $PM$ admet un point-selle $(x^*, \lambda^*)$ alors $w(\lambda^*) = f(x^*)$. Inversement, s'il existe $x^*$ et $\lambda^* \geq 0$ tels que $w(\lambda^*) = f(x^*)$, alors le $PM$ admet un point-selle et $(x^*, \lambda^*)$ est un tel point-selle. En fait, cette théorie de la dualité permet de ramener, lorsqu'il y a des points-selles, le $PMD$ a un la minimisation d'une fonction concave. De plus, on sait facilement utiliser des méthodes de sous-gradient pour cette fonction. Remarque : la dualité lagrangienne appliquée à un programme linéaire est en fait équivalente à la dualité "classique" de la $PL$. Malheureusement, comme un $PMD$ ne possède pas en général de point-selle, ce dernier théorème n'est cependant pas applicable pour $S$ discret, on verra dans la section suivante que l'écart $f(x^*) - w(\lambda^*)$ est souvent non nul.

### 2.2.3 Saut de dualité

On appelle saut de dualité le fait que la valeur $f(x^*) - w(\lambda^*)$ soit non nul. C'est le cas lorsqu'un $PM$ ne possède de point-selle pour sa fonction duale associée. C'est le cas par exemple de la plupart des $PMD$.

**Exemple**

Prenons par exemple le PLNE suivant où $S = \{0,1\}^3$

$$\begin{cases} \min f(x) = 10 - 3x_1 - 2x_2 - x_3 \\ 2x_1 + 3x_2 + 4x_3 \leq 4 \\ x = (x_1, x_2, x_3) \in \{0,1\}^3 \end{cases}$$

L'ensemble des points de $S$ est $y^1 = (0,0,0), y^2 = (1,0,0), y^3 = (0,1,0), y^4 = (1,1,0), y^5 = (0,0,1), y^6 = (1,0,1), y^7 = (0,1,1)$ et $y^8 = (1,1,1)$ Parmi elles, on trouve une solution optimale au PLNE pour $y^2 = (1,0,0)$ qui vaut $f(y^2) = 7$. qui vaut $g(x) = 2x_1 + 3x_2 + 4x_3 - 4$, on définit la fonction de Lagrange pour chaque point $y^i$ de $S$ comme étant

$$L(y^i, \lambda) = f(y^i) + \lambda g(y^i) = 10 - 3y_1^i - 2y_2^i - y_3^i + \lambda(2y_1^i + 3y_2^i + 4y_3^i - 4), \forall \lambda \in \mathbb{R}^+,$$

c'est à dire

$$\begin{cases} L(y^1, \lambda) = 10 - 4\lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^2, \lambda) = 7 - 2\lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^3, \lambda) = 8 - \lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^4, \lambda) = 5 + \lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^5, \lambda) = 9, \forall \lambda \in \mathbb{R}^+ \\ L(y^6, \lambda) = 6 + 2\lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^7, \lambda) = 7 + 3\lambda, \forall \lambda \in \mathbb{R}^+ \\ L(y^8, \lambda) = 4 + 5\lambda, \forall \lambda \in \mathbb{R}^+ \end{cases}$$

La fonction duale du $PMD$ est $w(\lambda) = \min_{i \in \{1,\ldots,8\}} \{L(y^i, \lambda)\}, \forall \lambda \geq 0$ qui est facile à calculer pour un $\lambda$ donné. On obtient par exemple $w(0) = 4, w(0;5) = 5,5$ et $w(1) = 5$. On veut en fait résoudre le programme dual qui est alors $\max_{\lambda \geq 0} w(\lambda)$. Or la fonction $w$ est concave ! On sait donc par ces trois valeurs (en $0, 0;5$ et $1$) que son optimum sera atteint sur l'intervalle $[0,1]$. On peut ainsi, par exemple par recherche dichotomique, on obtient alors un maximum pour que l'on peut visualiser comme étant le point le plus haut de l'enveloppe des droites $L(y^i, \lambda)$, c'est-à-dire, après calcul simple, $\lambda^* = \frac{2}{3}$ et $w(\lambda^*) = \frac{17}{3}$. (On peut noter que comme la relaxation linéaire de ce problème est égale à la relaxation lagrangienne, on obtient également un optimum fractionnaire à $x_R = (1, \frac{2}{3}, 0)$ qui donne $f(x_R) = \frac{17}{3}$.) Ainsi, pour cet exemple, on obtient un saut de dualité de $f(y^2) - w(\lambda^*) = \frac{4}{3}$.

### 2.2.4 La relaxation lagrangienne

La théorie de la dualité lagrangienne trouve son application la plus fréquente dans le cadre de la relaxation lagrangienne c'est-à-dire le fait d'appliquer la fonction de Lagrange pour un sous-ensemble bien choisi de contraintes. Considérons le $PM$ suivant :

$$\begin{cases} \min f(x) \\ g_i(x) \leq 0, i \in I \\ x \in S \subset \mathbb{R}^n \end{cases} \tag{2.9}$$

où $S$ est un ensemble non triviale pouvant être lui-même défini par des contraintes. On choisira d'appliquer la relaxation lagrangienne aux contraintes $g_i$ dans le cas où :

— le problème de minimisation de la fonction lagrangienne obtenue sur $S$ doit pouvoir être fait de manière efficace et exacte.

— l'ensemble des contraintes $g_i$ relaxée n'est pas trop important et bien choisie pour limiter le nombre de multiplicateur à manipuler.

L'objectif de cette relaxation est de donner de bonne borne d'évaluation du problème. Cependant il n'est pas rare que l'on obtienne également des solutions primales intéressantes. Un exemple très fréquent en pratique se situe dans le cadre des PLNE où certaines contraintes sont jugées "difficiles" à traiter. On peut alors écrire le PLNE de la façon suivante :

$$\begin{cases} \min c^T x \\ Ax \leq b \\ xd \\ x \in N^n \end{cases} \tag{2.10}$$

où les contraintes $Ax \leq b$ sont des contraintes que l'on voudrait ôter du programme : c'est principalement le cas si elles sont "couplantes". Pour mesurer l'efficacité de la relaxation lagrangienne, on va tenter de comparer la valeur d'évaluation obtenue par la relaxation linéaire du PLNE et celle obtenue par la relaxation lagrangienne $L$ des contraintes $Ax \leq B$, c'est-à-dire les valeurs de

$$\begin{cases} \min c^T x \\ Ax \leq b \\ Cx \leq d \\ x \in \mathbb{R}^n \end{cases} \tag{2.11}$$

et

$$\begin{cases} \max_\lambda \min_{x \in S} \{L(x, \lambda)\} \\ \text{avec } S = \{x \in N^n | Cx \leq d\} \end{cases} \tag{2.12}$$

En fait, on peut prouver le résultat suivant

**Lemme** En fait, on peut prouver le résultat suivant $(D_L)$ est égale à la valeur optimale du problème suivant :

$$\begin{cases} \min c^T x \\ Ax \leq b \\ x \in conv(S) \end{cases} \tag{2.13}$$

**Démonstration** On se limite ici au cas où $S$ contient un nombre ni de points que l'on note $S = \{x^1, x^2, ..., x^K\}$. Dans ce cas, la fonction de Lagrange est alors

$$w(\lambda) = \min_{k=1}^{K} (c^T x^k + \lambda^T (Ax^k - b)) \tag{2.14}$$

On peut voir que le dual lagrangien $(D)$ peut se réécrire alors comme un $PL$ en variables linéaires $(\lambda, z) \in R^{m+1}$ suivant

$$\begin{cases} \max z \\ z \leq c^T x^k + \lambda^T (Ax^k - b), \forall k \in \{1, ..., K\} \\ \lambda \in R_+^m \\ z \in R \end{cases} \tag{2.15}$$

En associant aux contraintes du $PL$ précédent les variables duales du $PL$ $u_k \geq 0$, on va prouver que l'on obtient le programme dual de la $PL$ du programme linéaire précédent :

$$\begin{cases} \min \sum_{i=1}^{K} (c^T x^k) u_k \\ \sum_{i=1}^{K} (Ax^k) u_k \leq b \\ \sum_{i=1}^{K} u_k = 1 \\ u_k \in \mathbb{R}^+, \forall k \in \{1, ..., K\} \end{cases} \tag{2.16}$$

En fait, le premier lot de contraintes correspond en fait aux variables $\lambda$ et la dernière contrainte à la variable $z$. Le lot de contraintes associé aux variables $\lambda$ n'est pas immédiat : pour l'obtenir, il faut remarquer que les contraintes de $(PL)$ peuvent chacune s'écrire

$$z - \sum_{i=1}^{m} \lambda_i (A^i x^k - b_i) \leq c^T x^k \tag{2.17}$$

où $A^i$ désigne le vecteur ligne correspondant à la ligne $i$ de $A$. Donc la contrainte associée à la variable $\lambda_i$ dans $(DL)$ est alors

$$-\sum_{k=1}^{K} (A^i x^k - b_i) u_k \geq 0 \text{ i.e. } \sum_{k=1}^{K} A^i x^k u_k \leq b_i \sum_{k=1}^{K} u_k$$

Par la dernière contrainte $\sum_{i=1}^{k} u_k = 1$, on obtient alors la contrainte correspondante dans $(DL)$. Considérons à présent un point $x$ quelconque de $conv(S)$, par définition il existe un vecteur $t \in R_+^K$ tel que $\sum_{i=1}^{k} t_k = 1$ et $x = \sum_{i=1}^{K} t_k x^k$. On remarque donc que le programme $(DPL)$ revient en fait à rechercher un point $x = \sum_{i=1}^{K} u_k x^k$ dans $conv(S)$. Par conséquent $(DPL)$ est exactement équivalent au programme donné dans l'énoncé du lemme. Ainsi dans le cas où $conv(S)\{x \in R^n | Cx \leq b\}$, ce qui est le cas plus fréquent en PLNE, on voit que la relaxation lagrangienne donne une meilleure évaluation que la relaxation linéaire.

## 2.3 Conditions d'optimalité en variables mixtes

Dans la suite, nous désignons par vecteur en variables mixtes un vecteur regroupant plusieurs types de variables :
• Des variables continues dont les valeurs peuvent varier continûment entre deux bornes réelles distinctes.
• Des variables entières astreintes à prendre des valeurs entières entre deux bornes entières distinctes.
• Des variables discrètes acceptant des valeurs discrètes réparties de façon quelconque entre deux bornes réelles distinctes.
Nous supposerons que ces différentes valeurs sont ordonnées de façon croissante.

Les composantes du vecteur de variables mixtes seront séparées en deux parties : les composantes continues et les composantes discrètes et entières. On notera ce vecteur de la manière suivante :

$$x = \{x_c, x_d\}^T \in \mathbb{R}^n$$

Où $x_c \in \mathbb{R}^{(n-d)}$ désigne les $(n-d)$ variables continues,
et $x_d \in E^d \subset \mathbb{R}^d$ les $d$ variables discrètes et entières.
Pour simplifier les notations nous regroupons l'ensemble des variables entières et discrètes sous l'appellation commune : $x_d$. La seule différence entre ces deux types de variables se situe au niveau de la répartition de l'ensemble des valeurs acceptables. Pour les variables entières il suffit de spécifier les bornes inférieures et supérieures alors que dans le cas des variables discrètes il faudra énumérer l'ensemble des valeurs possibles.
Le problème d'optimisation en variables mixtes s'écrira :

$$(P_c) \begin{cases} \min f(x) \\ s.t \\ c_j(x) \leq 0, j = 1...m \\ x \in \mathbb{R}^{(n-d)} \times E^d \end{cases}$$

avec :

$$f, c_j : x \in \mathbb{R}^{(n-d)} \times E^d \longrightarrow f(x), c_j(x) \in \mathbb{R}, j = 1, ..., m$$

On suppose que les fonctions $f(x)$ et $c_i(x)$ sont continues et différentiables sur $\mathbb{R}^n$, c'est à dire dans l'espace des variables continues étendu à tout le vecteur variable. De cette manière lorsque nous "relaxerons" les restrictions dues aux variables discrètes le problème $(P_c)$ pourra être considéré comme

un problème d'optimisation classique de $n$ variables continues. Le domaine des solutions de $(P_c)$ et toujours défini par :

$$D = \{x \in \mathbb{R}^{(n-d)} \times E^d / c_i(x) \leq 0, j = 1...m\}. \tag{2.18}$$

Le partage du vecteur variable mixte en variables continues et discrètes permet de considérer deux types de sous problèmes : l'un en variables continues et les variables discrètes étant supposées fixes, l'autre en variables discrètes et les variables continues restant fixes. Ces deux sous problèmes sont généralement étroitement liés, les valeurs des fonctions objectif et contraintes du sous problème en variables continues dépendront de celles des variables discrètes et vice-versa.
L'optimum en variables mixtes, $x^* = \{x_c^*, x_d^*\}^T$, sera atteint lorsque les composantes continues et discrètes du vecteur variable auront simultanément atteint leur valeur optimale. pour $x_d = x_d^*$, les composantes continues optimales $x_c^*$ satisferont les conditions d'optimalité de Kuhn et Tucker, sous réserve que le domaine des solutions soit qualifié en $x^* = \{x_c^*, x_d^*\}^T$. On a alors :
Pour $x_d = (x_d^*, x_c^*)$ est un optimum local continu du problème en variables mixtes $(P_c)$
s'il existe des réels $\lambda_j, j = 1...m$ positifs ou nuls tels que :

$$\begin{cases} \nabla_{x_c} f(x_c^*, x_d^*) + \sum_{j=1}^{m} \lambda_j \nabla_{x_c} c_j(x_c^*, x_d^*) = 0 \\ \lambda_j c_j(x_c^*, x_d^*) = 0, j = 1...m \end{cases}$$

Le voisinage discret de $x = \{x_c, x_d\}^T$ sera défini comme l'ensemble des points de $\mathbb{R}^{(n-d)} \times E^d$ dont les composantes discrètes prennent les valeurs discrètes "adjacentes" à celle de $x_d$ , excepté lui-même, les composantes continues restant fixes (figure suivante).

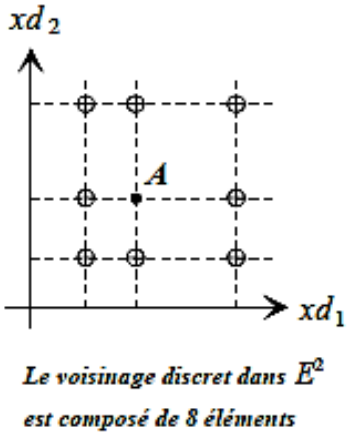



FIGURE 2.1 – Voisinage discret pour d=2

Si $d$ est le nombre de variables discrètes, le voisinage discret noté $v_d(x_d) \subset E^d$ est composé de $3^d - 1$ éléments.
Pour $x = \{x_c^*, x_d^*\}$ sera un optimum local discret pour le problème en variables mixtes $(P_c)$ si :

$$\forall x_d \in V_d(x_d) \text{ tel que } : c_j(x_c^*, x_d) \leq 0, j = 1...m$$

on a :

$$f(x_c^*, x_d^*) \leq f(x_c^*, x_d) \text{ et } c_j(x_c^*, x_d^*) \leq 0, j = 1...m$$

Cette définition ne permet pas d'établir des conditions mathématiques permettant de caractériser un optimum local discret. La seule façon envisageable pour statuer sur l'optimalité locale discrète est d'évaluer la fonction objectif pour l'ensemble des éléments du voisinage discret appartenant au domaine des solutions du problème $(P_c)$.
Cette méthode n'est envisageable que lorsque le problème comporte quelques variables discrètes. En effet, on voit dans le tableau ci-après, qu'à partir de 5 variables discrètes, l'évaluation de la fonction

objectif pour tous les éléments du voisinage discret conduit à un nombre de calculs considérable (on suppose pour simplifier que tous les éléments du voisinage discret appartiennent au domaine des solutions).

| **d** | 1 | 2 | 5 | 10 | 5 |
|---|---|---|---|---|---|
| $3^d - 1$ | 2 | 8 | 243 | 59048 | $1.43\ 10^7$ |

TABLE 2.1 – :Nombre de voisins discrets pour quelques valeurs de $d$.

Cette opération est d'autant plus lourde, qu'elle devra être recommencée pour chaque nouvelle valeur des composantes continues du vecteur en variables mixtes.

## 2.4 Méthodes de résolution

Pour des valeurs fixées des variables discrètes, les valeurs optimales correspondantes des variables continues d'un problème en variables mixtes peuvent être obtenues avec les méthodes de résolution décrites dans le chapitre précédent. Les méthodes de résolution en variables mixtes exploitent cette propriété, le problème étant alors de déterminer les valeurs optimales des variables discrètes. Cette opération peut s'effectuer de plusieurs façons :

1. Il est possible d'assimiler les restrictions dues aux variables discrètes à des fonctions contraintes et d'utiliser une méthode de pénalité pour ramener le problème en variables mixtes à un problème en variables continues.
2. On peut, comme dans le cas des variables continues, imaginer une méthode de "déplacement" dans l'espace des variables discrètes.
3. L'une des dernières possibilités est l'énumération explicite ou implicite de l'ensemble des points discrets appartenant au domaine des solutions.

### 2.4.1 Méthode des décompositions de Benders

**Principe de la méthode**

Considérons le problème suivant :

$$(P)\begin{cases} \min f(x,y) \\ s.t: \\ F(x,y) \leq 0 \\ x \in X \subset \mathbb{R}^{n-d}, y \in Y = E^d \subset \mathbb{R}^n \end{cases} \tag{2.19}$$

**Hypothèses :**

— X ensemble convexe.
— Pour chaque valeur fixée de $y \in Y$, $f(x,y)$ et $F(x,y) = (f_1(x,y), ..., f_m(x,y))^T$ sont convexes sur $X$.
— Hypothèses standards dans la théorie de la dualité lagrangienne :
  — $h(y) = \inf_{x \in X}\{f(x,y) : F(x,y) \leq 0\}$ prend une valeur finie.
  — ils existent des multiplicateurs optimaux pour le problème définissant $h(y)$
— (P) possède un domaine réalisable non vide et une solution optimale finie.

La projection du problème (P) sur l'espace des variables $y$ :

$$\min_{y \in Y}\{\inf_{x \in X}\{f(x,y) : F(x,y) \leq 0\}\} \qquad (P_1) \tag{2.20}$$

Dénotons par : $\Gamma_y = \{y : \exists x \in X$ tel que :$F(x,y) \leq 0\}$
De plus, puisque pour tout $y \notin \Gamma_y$ : $\{x \in X : F(x,y) \leq 0\} \neq \emptyset$ alors pour ces valeurs de $y \notin Y$, par convention : $\inf_{x\in X}\{f(x,y) : F(x,y) \leq 0\} = \infty$
Ainsi il n'y a pas de perte de généralité de restreindre à appartenir à $\Gamma_y$ dans $(P_1)$.
Donc le problème suivant $(P_2)$ est équivalent à $(P)$ :

$$(P_2) : \min_{y\in Y\cap\Gamma_y} \{\inf_{x\in X}\{f(x,y) : F(x,y) \leq 0\}\} \tag{2.21}$$

Dénotons par $h(y) = \inf_{x\in X}\{f(x,y) : F(x,y) \leq 0\}$

**Théorème ( Caractérisation de $\Gamma_y$)**
Supposons que $X$ est convexe et non vide, et les $f_1(x,y), ..., f_m(x,y)$ sont convexes sur $X$ pour tout $y \in Y$. De plus, supposons que pour tout $y \in Y$ : $Z_y = \{z \in \mathbb{R}^m, F(x,y) \leq z, \forall x \in X\}$ est fermé. Alors $\tilde{y} \in Y$ est un élément de $\Gamma_y$ si et seulement si $\inf_{x\in X}\{\lambda^T F(x,\tilde{y})\} \leq 0$ pour tout $\lambda \in \Lambda$, où $\Lambda = \{\lambda \in \mathbb{R}^m : \sum_{i=1}^m \lambda_i = 1, \lambda_i \geq 0\}$

**Preuve** Si $\tilde{y} \in \Gamma_y$, alors il existe $\tilde{x} \in X$ tel que : $F(\tilde{x},\tilde{y}) \leq 0$, par conséquent,

$$\inf_{x\in X} \lambda^T F(x,\tilde{y}) \leq 0 \text{ pour tout } \lambda \in \Lambda$$

Inversement, supposons que :

$$\inf_{x\in X}\{\lambda^T F(x,\tilde{y})\} \leq 0 \text{ pour tout } \lambda \in \Lambda$$

il s'ensuit que :

$$\sup_{\lambda\in\Lambda} \inf_{x\in X}\{\lambda^T F(x,\tilde{y})\} \leq 0$$

puisque la normalisation de $\lambda \in \Lambda$ n'influence pas le signe de la quantité $\inf_{x\in X}\{\lambda^T F(x,\tilde{y})$, alors

$$\sup_{\lambda\geq 0} \inf_{x\in X}\{\lambda^T F(x,\tilde{y})\} = 0$$

( car 0 est maintenant une valeur permise pour $\lambda$). Mais ceci revient à dire que le dual lagrangien par rapport aux contraintes $F(x,\tilde{y})$ du problème primal (défini en terme des variables $x$ )

$$\begin{cases} \min 0^T x \\ s.t : \\ F(x,\tilde{y}) \leq 0 \\ x \in X \end{cases} \tag{2.22}$$

a une valeur optimale égale à 0.
Se référant à la théorie lagrangienne, la fermeture de $Z_y = \{z \in \mathbb{R}^m, F(x,y) \leq z, \forall x \in X\}$ est suffisante pour assurer que le problème primal possède une solution réalisable. Donc $\tilde{y} \in \Gamma_y$.
**Remarque** La fermeture de $Z_y$ est assurée pour tout $y \in Y$ si $X$ est fermé et borné et si les $f_1(x,y), ..., f_m(x,y)$ sont continues sur $X$ pour tout $y \in Y$.
Le problème $(P_2)$ peut s'écrire alors sous la forme :

$$\begin{cases} \min h(y) \\ s.t : \\ \inf_{x\in X} \lambda^T F(x,y) \leq 0, \forall \lambda \in \Lambda \qquad (P_3) \\ y \in Y \end{cases} \tag{2.23}$$

où $h(y) = \inf_{x\in X}\{f(x,y) : F(x,y) \leq 0\}$ et $\Lambda = \{\lambda \in \mathbb{R}^m, \sum_{i=1}^m \lambda_i = 1, \lambda_i \geq 0\}$
Analysons maintenant $h(y) = \inf_{x\in X}\{f(x,y) : F(x,y) \leq 0\}$ en ayant recours à la théorie de la dualité lagrangienne.

**Théorème** Supposons que $X$ est fermé et non vide, et que $f(x,y), f_1(x,y), ..., f_m(x,y)$ sont convexes sur $X$ pour tout $y \in Y$. Supposons également que pour tout $y \in Y \cap \Gamma_y$ :

1. $h(y)$ prend une valeur finie
2. ils existent des multiplicateurs optimaux pour le problème définissant $h(y)$

alors, pour tout $y \in Y \cap \Gamma_y$, la valeur optimale de

$$\begin{cases} h(y) = \inf f(x,y) \\ s.t: \\ F(x,y) \leq 0 \\ x \in X \end{cases} \tag{2.24}$$

est égale à celle de son dual lagrangien par rapport aux contraintes $F(x,y) \leq 0$ :

$$h(y) = \sup_{u \geq 0}\{\inf_{x \in X}\{f(x,y) + u^T F(x,y)\}\} \tag{2.25}$$

**Preuve** Le résultat découle directement de la théorie de la dualité lagrangienne.
Introduisons la variable fictive $y_0$ :

$$y_0 = h(y) = \sup_{u \geq 0}\{\inf_{x \in X}\{f(x,y) + u^T F(x,y)\}\} \tag{2.26}$$

Alors le problème équivalent :

$$(P_3) \begin{cases} \min h(y) \\ s.t: \\ \inf_{x \in X}\{\lambda^T F(x,y)\} \leq 0, \forall \lambda \in \Lambda \\ y \in Y \end{cases} \tag{2.27}$$

peut s'écrire :

$$(P_4) \begin{cases} \min y_0 \\ s.t: \\ \sup_{u \geq 0}\{\inf_{x \in X}\{f(x,y) + u^T F(x,y)\}\} = y_0 \\ \inf_{x \in X}\{\lambda^T F(x,y)\} \leq 0, \forall \lambda \in \Lambda \\ y \in Y \end{cases} \tag{2.28}$$

ou encore

$$(P_5) \begin{cases} \min y_0 \\ s.t: \\ \inf_{x \in X}\{f(x,y) + u^T F(x,y)\} \leq y_0, \forall u \geq 0 \\ \inf_{x \in X}\{\lambda^T F(x,y)\} \leq 0, \forall \lambda \in \Lambda \\ y \in Y \end{cases} \tag{2.29}$$

Une approche toute indiquée pour résoudre $(P_5)$ est la stratégie de relaxation.
Il faut donc avoir un moyen simple de vérifier si une solution optimale $\tilde{y_0}$ , d'un problème relaxé est en fait une solution réalisable et optimale du problème équivalent $(P_5)$ , et lorsqu'elle ne l'est pas, pour générer une contrainte additionnelle pour raffiner la relaxation.
L'efficacité de l'approche vient de l'utilisation du sous-problème :

$$\begin{cases} h(\tilde{y}) = \min f(x,\tilde{y}) \\ s.t: \\ F(x,\tilde{y}) \leq 0, \\ x \in X \end{cases} \tag{2.30}$$

Lors de la résolution de ce sous problème, une des situations suivantes se présente :

1. Le problème possède une solution optimale $\tilde{x}$, alors $\tilde{y} \in \Gamma$. Si de plus, $h(\tilde{y}) \leq \tilde{y}_0$ alors $\tilde{y}$ est une solution réalisable optimale de $(P_5)$ puisque

$$\begin{array}{rcl} \tilde{y}_0 & \geq & h(\tilde{y}) \\ & = & \sup_{u \geq 0}\{\inf_{x \in X}\{f(x,\tilde{y}) + u^T F(x,\tilde{y})\}\} \\ & \geq & \inf_{x \in X}\{f(x,\tilde{y}) + u^T F(x,\tilde{y})\}, \forall u \leq 0 \end{array} \tag{2.31}$$

   Alors $\tilde{x} = \tilde{y}$ est une solution optimale du problème original (P).

2. Le problème possède une solution optimale $\tilde{x}$, alors $\tilde{y} \in \Gamma_y$.
   Si de plus, $h(\tilde{y}) > \tilde{y}_0$, alors utiliser un vecteur de multiplicateurs optimaux $\tilde{u}$ du sous problème pour générer une nouvelle contrainte

$$\inf_{x \in X}\{f(x,\tilde{y}) + \tilde{u}^T F(x,\tilde{y})\} \leq y_0$$

   à ajouter à la relaxation actuelle pour en générer une nouvelle.

3. Le sous problème n'est pas réalisable. Alors $y \notin \Gamma$, et par conséquent l'une des contrainte

$$\inf_{x \in X}\{\lambda^T F(x,y)\} \leq 0$$

   est violée pour certain $\tilde{\lambda}$. Il suffit d'identifier et d'ajouter la contrainte correspondante à la relaxation actuelle.

Notons qu'il est sous-entendu qu'il est facile d'identifier un tel $\tilde{\lambda}$

**Procédure de relaxation pour résoudre** $(P_5)$

Considérons le problème $(P_5)$ :

$$\begin{cases} \min y_0 \\ s.t: \\ \inf_{x \in X}\{f(x,y) + u^T F(x,y)\} \leq y_0, \forall u \geq 0 \\ \inf_{x \in X}\{\lambda^T F(x,y)\} \leq 0, \forall \lambda \in \Lambda \\ y \in Y \end{cases} \tag{$P_5$}$$

Le Procédure de relaxation pour résoudre $(P_5)$ est défini comme suit :

— **Initialisation :** Soit $\tilde{y} \in Y \cap \Gamma_y$ tel que $\{x \in X : F(x,\tilde{y})\} \neq \emptyset$
  Déterminer une solution optimale $\tilde{x}$ du sous-problème

$$\begin{cases} h(\tilde{y}) = \min f(x,\tilde{y}) \\ s.t: \\ F(x,\tilde{y}) \leq 0 \\ x \in X \end{cases} \tag{2.32}$$

  et un vecteur de multiplicateurs optimaux $\tilde{u}$.
  Poser $\mu = 1$ et $\varphi = 0$, et définir le premier problème relaxé avec la contrainte

$$\inf_{x \in X}\{f(x,y) + \tilde{u}^T F(x,y)\} \leq y_0$$

  Définir $UB = f(\tilde{x},\tilde{y})$ et un niveau de tolérance $\delta$.
  Résoudre le problème relaxé :

$$\begin{cases} \min y_0 \\ s.t: \\ \inf_{x \in X}\{f(x,y) + \tilde{u}^{iT} F(x,y)\} \leq y_0, j = 1, ..., \mu \\ \inf_{x \in X}\{\lambda^{jT} F(x,y)\} \leq 0, j = 1, ..., \varphi \\ y \in Y \end{cases} \tag{2.33}$$

Soit $\tilde{y} = \tilde{y}_0$ une solution optimale.
Si $UB - \tilde{y}_0 \leq \delta$, alors terminer la procédure avec la solution associée à $UB$ comme étant une solution $\delta-$optimale.

— Résoudre le sous-problème :

$$\begin{cases} h(\tilde{y}) = \min f(x, \tilde{y}) \\ s.t : \\ F(x, \tilde{y}) \leq 0 \\ x \in X \end{cases} \tag{2.34}$$

Si le sous-problème n'est pas réalisable, déterminer un $\tilde{\lambda} \in \Lambda$ tel que : $\inf_{x \in X} \tilde{\lambda}^T F(x, \tilde{y}) > 0$.
Augmenter $\varphi$ de 1 et introduire la contrainte $\inf_{x \in X} \tilde{\lambda}^T F(x, \tilde{y}) \leq 0$ dans le problème relaxé et reprendre l'étape 1.
Si $\tilde{x}$ est une solution optimale du sous-problème telle que $f(\tilde{x}, \tilde{y}) - \tilde{y}_0 > \delta$, alors déterminer un vecteur de multiplicateurs optimaux $\tilde{u}$ .
Augmenter $\mu$ de 1 et introduire la contrainte $\inf_{x \in X}\{f(x, y) + \tilde{u}^T F(x, y)\} \leq y_0$ dans le problème relaxé.
Poser $UB = \min\{UB, f(\tilde{x}, \tilde{y})\}$ et reprendre l'étape 1.

— Si la solution optimale $\tilde{x}$ du sous-problème

$$\begin{cases} h(\tilde{y}) = \min f(x, \tilde{y}) \\ s.t : \\ F(x, \tilde{y}) \leq 0 \\ x \in X \end{cases} \tag{2.35}$$

est telle que $f(\tilde{x}, \tilde{y}) - \tilde{y}_0 \leq \delta$ alors terminer la procédure avec $\tilde{x}, \tilde{y}$ une solution $\delta-$optimale.

**Remarque :** Au cours de premières itérations, il faut, au besoin, borner de façon artificielle le problème relaxé de l'étape 1. En effet, même si le problème original possède une solution optimale finie, le problème relaxé peut ne pas être borné.

**Convergence de la procédure**

La convergence de cette procédure ne découle pas nécessairement de celle de la stratégie de relaxation puisque le problème équivalent

$$(P_5) \begin{cases} \min y_0 \\ s.t : \\ \inf_{x \in X}\{f(x, y) + u^T F(x, y)\} \leq y_0, \forall u \geq 0 \\ \inf_{x \in X}\{\lambda^T F(x, y)\} \leq 0, \forall \lambda \in \Lambda \\ y \in Y \end{cases} \tag{2.36}$$

ne comporte pas nécessairement un nombre fini de contraintes. Il faut donc des hypothèses supplémentaires pour assurer la convergence de cette procédure. En voici deux mentionnées dans l'article de Geoffrion [13] :

i) $Y$ est un ensemble discret fini.
L'argument de convergence repose sur le fait que le même élément $\tilde{y} \in Y$ ne peut être solution optimale d'un prochain problème relaxé puisque la nouvelle contrainte ajoutée l'élimine. Ainsi, puisque $Y$ est un ensemble discret fini, le nombre d'itérations est donc fini.

ii) $Y \subset \Gamma$ et compact. $X$ est convexe et compact. $f$ et $f_1, ..., f_m$ sont convexes sur $X$ pour tout $y \in Y$ pour tout $X \times Y$. L'ensemble $U(y)$ des multiplicateurs optimaux du sous-problème est non vide pour tout $y \in Y$ et uniformément borné dans un voisinage de $y$.

L'argument est plus complexe et repose sur la convergence d'une sous-suite de points $y^k, y0^k$ en exploitant les propriétés des ensembles $X, Y, U(y)$, $\inf_{x \in X}\{\lambda^t F(x, y)\} \leq 0$ puisque $Y \subset \Gamma$.

**Commentaires sur l'utilisation de la méthode** Notons que le problèmes équivalent

$$(P_5)\begin{cases}\min y_0\\ s.t:\\ \inf_{x\in X}\{f(x,y)+u^TF(x,y)\}\leq y_0, \forall u\geq 0\\ \inf_{x\in X}\{\lambda^TF(x,y)\}\leq 0, \forall\lambda\in\Lambda\\ y\in Y\end{cases} \tag{2.37}$$

et ses relaxations ne sont pas simples à traiter en pratique puisque les contraintes sont spécifiées en termes de problèmes d'optimisation. Ainsi, à moins que ces problèmes soient facilement résolus, la méthode devient difficilement utilisable en pratique.

Dans la cas où il n'y a pas de contraintes de type $\inf_{x\in X}\{\lambda^TF(x,y)\}\leq 0$ ($i.e., Y\subset\Gamma$), à l'étape 1, nous pouvons remplacer les relaxation

$$\begin{cases}vr_\mu=\min y_0\\ s.t:\\ \inf_{x\in X}\{f(x,y)+u^jTF(x,y)\}\leq y_0, j=1,...\mu\\ y\in Y\end{cases} \tag{2.38}$$

par les approximations

$$\begin{cases}\tilde{v}\tilde{r}_\mu=\min y_0\\ s.t:\\ f(x^j,y)+(u^j)^TF(x^j,y)\leq y_0, j=1,...\mu\\ y\in Y\end{cases} \tag{2.39}$$

où $x^j$ était la solution optimale du sous-problème résolu à l'étape 2

$$\begin{cases}h(\tilde{y})=\min f(x,\tilde{y})\\ s.t:\\ F(x,\tilde{y}\leq 0\\ x\in X\end{cases} \tag{2.40}$$

à l'itération $j$.
Remplacer les relaxations

$$\begin{cases}vr_\mu=\min y_0\\ s.t:\\ \inf_{x\in X}\{f(x,y)+(u^j)^TF(x,y)\}\leq y_0, j=1,...\mu\\ y\in Y\end{cases} \tag{2.41}$$

par les approximations

$$\begin{cases}\tilde{v}\tilde{r}_\mu=\min y_0\\ s.t:\\ f(x^j,y)+(u^j)^TF(x^j,y)\leq y_0, j=1,...\mu\\ y\in Y\end{cases} \tag{2.42}$$

Or puisque $x^j\in X$ pour tout $j$ il s'ensuit que

$$\inf_{x\in X}\{f(x,y)+(u^j)^TF(x,y)\}\leq f(x^j,y)+(u^j)^TF(x^j,y) \tag{2.43}$$

ainsi, si

$$f(x^j,y)+u^jTF(x^j,y)\leq y_0 \tag{2.44}$$

alors

$$\inf_{x\in X}\{f(x,y)+(u^j)^TF(x,y)\}\leq f(x^j,y)+(u^j)^TF(x^j,y)\leq y_0 \tag{2.45}$$

Donc le domaine réalisable de toute relaxation est au moins aussi grand que celui de son approximation, et il s'ensuit que $vr_\mu\leq\tilde{v}\tilde{r}_\mu$.

### 2.4.2 Méthode utilisant un principe de pénalisation

Davydov et Sigal ([16]) proposent d'intégrer les restrictions dues aux variables entières grâce à une fonction contrainte égalité associée à chaque variable entière $x_{d_i}$ . L'expression retenue pour cette fonction contrainte est :

$$\begin{cases} \Psi(x_{d_i}) = (x_{d_i} - q_i)^2 \text{ si } x_{d_i} < q_i \\ (x_{d_i} - Q_i)^2 \text{ si } x_{d_i} > Q_i \\ \sin^2(\pi, x_{d_i}) \text{ si } x_{d_i} \in [q_i, Q_i] \end{cases} \tag{2.46}$$

où $q_i$ et $Q_i$ sont les bornes entières inférieures et supérieures de $x_{d_i}$

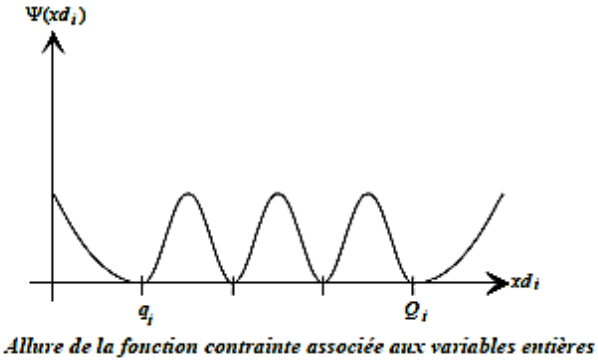



FIGURE 2.2 – Allure de la fonction contrainte associée aux variables entières

Ces fonctions contraintes, associées à un coefficient de pénalisation spécifique, sont ensuite ajoutées aux fonctions contraintes du problème dans une méthode de pénalité extérieure utilisant une fonction de pénalisation quadratique.
Gisvold et Moe [17] ont suggéré une approche similaire s'appliquant aux cas des variables discrètes avec une répartition quelconque des valeurs discrètes. Dans ce cas les restrictions dues aux variables discrètes sont intégrées dans une seule fonction contrainte dont l'expression est :

$$Q : x_d \in E^d \longrightarrow Q(x_d) \in \mathbb{R} \tag{2.47}$$

telle que :

$$Q(x_d) = \sum_{i-1}^{d} 4q_i(1 - q_i)^{\beta^k}; q_i = \frac{(x_{d_i}) - z_i^l}{z_i^u - z_i^l} \tag{2.48}$$

Avec :
$z_i^l, z_i^u$ : Bornes inférieures et supérieures des valeurs discrètes encadrant la valeur courante de $x_{d_i}$
$\beta^k \geq 1$ : Exposant de pénalisation.

Les auteurs de cette méthode utilisent une méthode de pénalité intérieure pour résoudre un petit problème d'optimisation de structure (10 variables dont 8 discrètes, 27 contraintes). A priori séduisantes, ces approches posent cependant quelques problèmes. L'introduction de ces fonctions contraintes dans une fonction de pénalisation produit un certain nombre de minimums locaux au niveau des valeurs entières et discrètes. L'obtention d'un optimum est alors délicate compte tenu des risques de convergence de la méthode de minimisation sans contrainte vers l'un de ces nombreux minimums locaux. De plus l'ajout des coefficients de pénalité associés à ces fonctions contraintes complique le choix des valeurs initiales et des stratégies d'évolution et augmente également les risques d'instabilité numérique.

### 2.4.3 Déplacement dans l'espace des variables discrètes

Pappas et Allentuch [19] proposent une méthode de recherche optimale discrète utilisant comme point de départ l'optimum en variables continues déterminé par une méthode de pénalité extérieure en relaxant les restrictions sur les variables discrètes. A partir de cet optimum la méthode consiste à se déplacer vers le point discret le plus proche et à procéder à une séquence de minimisation par rapport à une variable discrète les autres restant fixes. L'ordre des séquences de minimisation est celui des "sensibilités" décroissantes de chaque variable discrète. La sensibilité $s_i$ de la variable discrète $x_{d_i}$ est définie par :

$$s_i = \frac{[f^*(x_{d_i}) - f^*(x_{d_i} + \Delta)]}{\Delta_i} \tag{2.49}$$

Avec :
$\Delta$ différence entre la valeur courante $x_{d_i}$ et les valeurs discrètes adjacentes.
$f^*(x_{d_i}) = P(x_c^*, x_{d_1}, ..., x_{d_i}, ..., x_{d_d}, r)$
$f^*(x_{d_i}\Delta) = P(x_c^*, x_{d_1}, ..., x_{d_i} + \Delta, ..., x_{d_d}, r)$
La fonction $P : (x, r) \in \mathbb{R}^n \times \mathbb{R} \longrightarrow P(x, r) \in \mathbb{R}$ une fonction de pénalisation quadratique et $x_c^*$ la valeur optimale des variables continues pour $x_{d_1}, ..., x_{d_i}, ..., x_{d_d}$ fixé.
Cette minimisation par rapport à la variable discrète $x_{d_i}$, utilisant une méthode de dichotomie modifiée pour tenir compte des valeurs discrètes, est répétée autant de fois qu'il y a de variables discrètes. Pour chaque nouvelle valeur discrète de $x_{d_i}$, l'optimum $x_c^*$ est recalculé par la méthode de pénalité. Si l'ordre des sensibilités croissantes est modifié un nouveau cycle de minimisation est entamé. Lorsque cet ordre n'est plus modifié, ce processus est arrêté. On obtient alors généralement un optimum discret lorsque la fonction de pénalisation est unimodale. Les auteurs précisent que cette méthode n'est applicable que sur des problèmes de petites dimensions (4,5 variables) dans lesquels les effets de la discrétisation ne sont pas trop prononcés, car dans ce cas on peut espérer avoir un optimum discret dans le voisinage discret de l'optimum en variables continues.
Cha et Mayne [20],[21], ont modifié une méthode d'approximation quadratique récursive pour pouvoir l'appliquer sur des problèmes en variables mixtes. A partir d'un point de départ en variables mixtes situé à l'intérieur du domaine des solutions, la direction de recherche pour une minimisation unidimensionnelle est calculée en résolvant un sous-problème d'optimisation quadratique construit au point courant du processus.
La recherche unidimensionnelle effectuée dans cette direction tient compte des limites imposées par les fonctions contraintes, des éventuelles propriétés de monotonie locale et des restrictions dues aux variables discrètes. Le point obtenu est alors éventuellement arrondi au point discret le plus proche. Si cette étape ne parvient pas à améliorer la solution courante, une procédure de recherche locale discrète est appliquée. Les points discrets du voisinage discret situés du coté opposé au gradient de la fonction objectif sont considérés un à un.
Cette méthode permet d'obtenir une solution optimale en variable mixte pour 25 problèmes tests avec un nombre d'évaluations de fonction inférieur par rapport aux autres méthodes utilisées dans ce test, et notamment celles basées sur un principe de séparation et d'évaluation. Il semble toutefois que cette méthode soit réservée aux problèmes présentant un "pas de discrétisation" relativement faible et régulier. Dans le cas contraire, la procédure d'arrondi utilisée risquerait de compromettre la convergence de la méthode.

### 2.4.4 Méthodes d'énumération

Le moyen le plus sûr pour localiser un optimum discret est sans doute d'énumérer l'ensemble des points discrets du domaine des solutions et de résoudre pour chacun d'eux le problème d'optimisation en variables continues, les variables discrètes étant fixées par cette énumération. Il suffira alors de retenir comme solution le point discret donnant la plus petite valeur de la fonction objectif. Cependant nous avons vu dans le $x1.2$ de ce chapitre que cette approche n'est valable que pour certains cas particuliers, car généralement le nombre de calculs nécessaires est prohibitif. Grâce à un principe algorithmique s'appuyant sur une représentation particulière de l'ensemble des valeurs discrètes on peut éviter l'énumération de tous les points discrets du domaine des solutions.

**Principe d'une méthode de séparation et évaluation (Branch and Bound)**

Considérons le problème d'optimisation énoncé sous la forme très générale suivante : trouver :

$$f(s^*) = \min_{s \in S}\{f(s)\}$$

avec :

$$f : s \in S \longrightarrow f(s) \in \mathbb{R}$$

Soit un sous-ensemble $S_i$ inclus dans $S$. Le principe d'évaluation sera défini de la manière suivante : On dit que l'on sait "évaluer" le sous-ensemble $S_i$ si on peut déterminer un réel, dépendant de $S_i, g(S_i)$ tel que :

$$g(S_i) \leq f(s), \forall s \in S$$

Cette évaluation est dite "exacte" si on connaît un élément de $S_i, s_i^*$ tel que :

$$g(S_i) = f(s_i^*)$$

L'évaluation $g'(S_i)$ est meilleure que l'évaluation $g(S_i)$ si :

$$g'(S_i) > g(S_i)$$

L'évaluation associée à un sous-ensemble vide ($S_i = \oslash$) sera $g(S_i) = +\infty$ . On notera que la meilleure évaluation que l'on puisse obtenir est une évaluation exacte.
L'ensemble $S$ et dit "séparé" en $k$ sous-ensembles $S_i, i = 1...k$ lorsque :

1. Chacun des sous-ensembles $s_i$ est inclus dans $S$
2. L'union de tous ces sous-ensembles est égale à $S$.

On déduit immédiatement de ces deux définitions le résultat suivant :
Lorsque pour chacun des sous-ensembles $S_i$ tel que :

$$\begin{cases} S_i \subset S \text{ pour } i = 1...k \\ S_1 \cup S_2 \cup ... \cup S_k = S \end{cases}$$

on obtient une évaluation exacte, la solution optimale $s^*$ du problème d'optimisation est donnée par le sous-ensemble $S_i^*$ ayant l'évaluation $g(S_i^*)$ minimale, et on a :

$$g(S_i^*) = f(s^*) = \min_{i=1...k}\{g(S_i)\}$$

Supposons que l'on connaisse une solution $\widehat{s}$ Supposons que l'on connaisse une solution $s^*$ du problème d'optimisation. Par définition $\widehat{s}$ est telle que :

$$f(\widehat{s}) \geq f(s^*)$$

Si l'évaluation du sous-ensemble $S_i$ est telle que :

$$g(S_i) \geq f(\widehat{s}) \geq f(s^*) \tag{2.50}$$

Le sous-ensemble $S_i$ ne peut pas contenir de solution optimale, sinon cela contredirait le principe d'évaluation définit plus haut. Dans ce cas il sera inutile de décomposer (séparer) ce sous-ensemble en sous-ensembles plus petits. Ce sous-ensemble sera qualifié de "stérile". Lorsque l'évaluation pour un sous-ensemble donné est exacte, ce sous-ensemble peut être également "stérilisé".
**L'algorithme général d'une méthode par séparation et évaluation est le suivant :**

1. Au départ la valeur de $\widehat{s}$ est indéterminée et $\widehat{f} = f(\widehat{s}) = \infty)$
2. Tant qu'il reste des sous-ensembles à séparer (non stérile) effectuer les opérations suivantes :
   — Choix d'un sous-ensemble à séparer (au départ on sépare l'ensemble $S$ tout entier)
   — Calcul de l'évaluation :
   si l'évaluation est exacte alors $\widehat{s} = s_i^*$ et $\widehat{f} = f(\widehat{s}) = f(S_i^*) = g(S_i)$

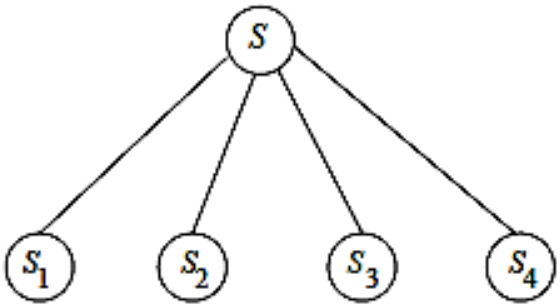
— Séparation du sous-ensemble
— Stérilisation des sous-ensembles $S_i$ tel que :

$$g(S_i) \geq f(\widehat{s}) \text{ où } f(s_i^*) = g(S_i) \tag{2.51}$$

La méthode sera entièrement définie lorsqu'on aura précisé :

[•]Le choix du sous-ensemble à séparer à chaque étape.Le principe de calcul de l'évaluation. La méthode de séparation des sous-ensembles.

L'association d'une arborescence à la décomposition progressive de l'ensemble permet de visualiser la mise en œuvre d'une procédure de recherche par séparation et évaluation. Le sommet de ce graphe est représenté par l'ensemble $S$ tout entier, à chaque étape les sommets pendants correspondent aux sous-ensembles $s_i$. Les arcs reliant chaque sommet représentant les relations d'inclusion (figure suivante). Les performances d'une méthode de recherche de la solution optimale par séparation et évaluation




*Arborescence associée à la décomposition de S*

•

dépendent pour une très large part de la manière dont sont choisis les sous-ensembles à séparer et de la méthode de séparation de ces sous-ensembles, mais également de la qualité de l'évaluation. En pratique il faudra trouver un bon compromis entre un principe d'évaluation donnant une approximation assez grossière de la valeur minimale de la fonction objectif sur un sous-ensemble donné, mais rapide à calculer et une évaluation plus précise et donc souvent plus longue à calculer. Les stratégies pour le choix du sous-ensemble à séparer sont généralement les suivantes :

Sélectionner le sous-ensemble dont l'évaluation est la plus faible possible, avec l'idée que ce sous-ensemble a le plus de chances de contenir une solution optimale. Ce choix correspond à une stratégie d'exploration de l'arborescence dite de "largeur d'abord". Le risque ici est d'avoir à explorer une fraction importante de l'arborescence avant d'obtenir une évaluation exacte, donc une solution approchée $\widehat{s}$.

Sélectionner le sous-ensemble le plus récemment séparé. Il s'agit alors d'une exploration dite de "profondeur d'abord". Cette méthode permet d'obtenir le plus tôt possible une solution approchée $\widehat{s}$, par contre cette solution est généralement moins bonne que celle obtenue par une exploration de "largeur d'abord". L'inconvénient de ce type de choix est de ne pas tenir compte de la fonction d'évaluation, on est alors amené à traiter des sous-ensembles dont l'évaluation est très médiocre et qui ont peu de chance de contenir une solution optimale.

On a toujours intérêt à obtenir rapidement une solution approchée $\widehat{s}$, car celle-ci permet de stériliser très tôt les sous-ensembles ayant une évaluation supérieure à la valeur de la fonction objectif pour cette solution. On pourra stériliser d'autant plus de sous-ensembles que cette solution approchée est meilleure. Il n'existe pas de stratégies donnant systématiquement de meilleurs résultats, cela dépend du problème à traiter. L'approche la plus couramment utilisée consiste à débuter par une exploration de "profondeur d'abord", permettant d'obtenir rapidement une solution approchée $\widehat{s}$ et ensuite de poursuivre par une exploration de "largeur d'abord" pour améliorer cette solution. On remarque que la méthode présente une certaine indépendance vis à vis de la structure du problème d'optimisation. Plus précisément les liens entre une méthode procédant par séparation et évaluation et le type du problème à traiter (linéaire, non linéaire, avec ou sans fonction contrainte) sont définis par le principe d'évaluation et la technique de séparation de l'ensemble $S$.

**Mise en œuvre d'un principe de séparation et évaluation : résolution des problèmes linéaires en nombres entiers**

C'est pour résoudre ce type de problème que le principe de séparation et évaluation fut mis au point. Ce principe a ensuite été appliqué sur de nombreux cas de problèmes de graphes et d'optimisation combinatoire. Nous présenterons ici les particularités de cette mise en œuvre en précisant seulement le principe de séparation et celui d'évaluation spécifique aux problèmes linéaires en nombre entier. Un problème d'optimisation linéaire en nombres entiers peut s'écrire sous la forme :

$$\begin{cases} \min f(x) = c^T.x \\ \text{Sous les fonctions contraintes :} \\ A.x + b \leq 0 \\ x_i \in N, i = 1...n \end{cases} \tag{2.52}$$

avec $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ et $A$ matrice $m \times n$ de rang $n$. Nous supposerons que l'ensemble des solutions $S$ tel que :

$$S = \{x \in \mathbb{R}^n / A.x + b \leq 0, x_i \in N, i = 1...n\} \tag{2.53}$$

est borné. Soit deux vecteurs de $n$ composantes entières $u$ et $v$ tel que : $u < v$ Le sous-ensemble $S_{u,v}$ inclus dans $S$ et défini grâce à ces deux vecteurs est :

$$S_{u,v} = \{x \in S / u \leq x \leq v\} \tag{2.54}$$

Si on associe à $S_{u,v}$ le problème linéaire en nombres entiers suivant :

$$(P_{u,v}) \begin{cases} \min f(x) = c^T.x \\ s.t : \\ A.x + b \leq 0 \\ u \leq x \leq v', x_i \in N, i = 1...n \end{cases} \tag{2.55}$$

et le problème linéaire en variables continues et bornées :

$$(P'_{u,v}) \begin{cases} \min f(x) = c^T.x \\ s.t : \\ A.x + b \leq 0 \\ u \leq x \leq v \end{cases} \tag{2.56}$$

on peut facilement définir le principe d'évaluation pour ce sous-ensemble $S_{u,v}$ En effet il suffit devoir que l'ensemble des solutions de $(P_{u,v})$ , est inclus dans celui de $(P'_{u,v})$, et que par conséquent la valeur optimale de la fonction objectif de $(P'_{u,v})$ ,est nécessairement inférieure ou égale à celle de $(P_{u,v})$. D'autre part, lorsque la solution optimale de $(P'_{u,v})$ ,est "entière", la valeur de la fonction objectif de $(P'_{u,v})$, est égale à celle du problème $(P_{u,v})$. Donc la valeur optimale de la fonction objectif de $(P'_{u,v})$ constitue bien une évaluation (au sens de la définition du sous-ensemble $S_{u,v}$. Lorsque la solution optimale correspondante est entière cette évaluation est "exacte". On notera qu'il n'existe pas ici de meilleure évaluation. Le principe de séparation est le suivant : Soit $\overline{x}$ la solution optimale de $(P'_{u,v})$. Pour séparer $S_{u,v}$ on choisira un indice $r$ on choisira un indice $r^{ieme}$ composante de $\overline{x}$ ne soit pas entière. L'ensemble $S_{u,v}$ est alors séparé en deux sous ensembles disjoints $S_{u',v}$ et $S_{u,v'}$ où $u'$ et $v'$ sont deux vecteurs de $n$ composantes entières tels que :

$$u'_i = Ent(\overline{x}_r) + 1 \textbf{ si } i = r u_i \textbf{ si } i \neq r \tag{2.57}$$

et

$$v'_i = Ent(\overline{x}_r) \textbf{ si } i = r v_i \textbf{ si } i \neq r \tag{2.58}$$

On a alors :

$$\begin{cases} S_{u,v'} \cap S_{u',v} = \oslash \\ S_{u,v'} \cup S_{u',v} = S_{u,v} \\ \overline{x} \notin S_{u',v}; \overline{x} \notin S_{u,v'} \end{cases} \tag{2.59}$$

Il existe de nombreuses règles pour le choix de la variable de séparation, la plus couramment appliquée correspond au choix de $\overline{x}_r$ tel que la valeur de $\overline{x}_r$ soit le plus "fractionnaire possible". C'est à dire lorsque $|\overline{x}_r - Ent(\overline{x}_r)|$ est le plus proche possible de 0.5. L'algorithme de résolution est identique à celui du $x$2.3.1. Au départ de la méthode on affectera à $u$ et $v$ de grandes valeurs, et on choisira comme point de départ la solution optimale de $(P'_{u,v})$

**Application aux problèmes non linéaires en variables mixtes**

Gupta et Ravindran [22] ont réutilisé le principe ci-dessus pour l'appliquer au problème non linéaire en variables entières suivant :

$$(P_c) \begin{cases} \min f(x) \\ s.t : \\ c_j(x) \leq 0, j = 1...m \\ x_i \in N, i = 1...n \end{cases} \tag{2.60}$$

La méthode de séparation des sous ensembles est identique, seul le type du problème associé à chaque sous-ensemble est modifié. Dans ce cas pour évaluer le sous-ensemble défini par :

$$S_{u,v} = \{x_i \in N, i = 1..n/c_j(x) \leq 0, j = 1...m; u \leq x \leq v\} \tag{2.61}$$

il faut résoudre le problème d'optimisation en variables continues suivant :

$$(P_c) \begin{cases} \min f(x) \\ s.t : \\ c_j(x) \leq 0, j = 1...m \\ u \leq x \leq v \end{cases} \tag{2.62}$$

Dans l'algorithme proposé ce calcul est effectué avec une méthode de gradient réduit généralisé. L'étude statistique menée par les auteurs sur une vingtaine de problèmes tests a montré que pour un problème non linéaire en nombres entiers, la séparation des sous ensembles suivant la variable dont la valeur est la plus fractionnaire possible donne les meilleurs résultats.

Sandgren [23] propose une démarche similaire, applicable sur des problèmes en variables mixtes. Le principe de séparation pour les variables entières est identique. Dans le cas des variables discrètes, les bornes u et v sont données par les valeurs discrètes immédiatement supérieures et inférieures à la valeur "non discrète" de la variable de séparation. L'auteur suggère d'effectuer la séparation suivant la variable discrète provoquant la plus forte diminution de la fonction objectif. On retrouve ici l'idée de Pappas et Allentuch, d'effectuer les minimisations unidimensionnelles discrètes dans l'ordre des "sensibilités" décroissantes (Cf. $x$2.2). Les principaux inconvénients des méthodes procédant par séparation et évaluation de l'ensemble des solutions sont :
• Elles nécessitent généralement de nombreux calculs pour parvenir à une solution optimale. En effet le nombre de problèmes non linéaires à résoudre est d'autant plus important que l'arborescence comporte de sommets (ou sous ensembles) non stérilisés.
• Il est généralement impossible de connaître la taille de l'arborescence qui sera développée, donc la quantité de mémoire qui sera utilisée, pour la résolution d'un problème.

Elles possèdent cependant les avantages fondamentaux suivants :
— Ces méthodes permettent d'obtenir : Une solution optimale en variables mixtes (la première solution approchée pour laquelle on obtient une évaluation exacte, par exemple). Toutes les solutions optimales en variables mixtes en explorant l'ensemble de l'arborescence.

— Elles sont extrêmement souples au niveau de leur mise en œuvre, et peuvent prendre en compte les structures discrètes les plus variées.

## 2.5 Conclusion

Les problèmes d'optimisation non linéaires en variables mixtes sont des problèmes d'optimisation très difficiles à résoudre. Ces difficultés sont intrinsèquement liées à la présence des variables discrètes. En effet elles introduisent de nombreux optimums globaux même dans le cas de problèmes convexes. De plus ces optimums discrets ne sont pas nécessairement situés à proximité d'une solution optimale en variables continues. Dans ce type de situation les méthodes procédant par approximation ne donnent pas de bons résultats. L'absence de conditions d'optimalité permettant de statuer efficacement sur l'optimalité d'une solution couramment calculée constitue un sérieux écueil dans la mise au point de méthodes de résolution efficaces. En ce qui concerne les problèmes de conception optimale, dans lesquels la présence des variables discrètes est inévitable, les méthodes procédant par séparation et évaluation sont certainement les mieux adaptées à la recherche des solutions optimales pour ce type de problèmes. L'argument de poids en faveur de ces méthodes est sans aucun doute la souplesse de leur mise en œuvre et leur capacité à prendre en compte des variables discrètes avec de très larges pas de discrétisation, et réparties de façon quelconque. Comme généralement les problèmes de conception optimale sont de taille moyenne (une dizaine de variables), on peut espérer obtenir une arborescence associée à l'ensemble des solutions de taille raisonnable.

# Chapitre 3

# Réseaux neuronaux et apprentissage

## 3.1 Apprentissage automatique

Le Machine Learning, aussi appelé apprentissage automatique en français, est une forme d'intelligence artificielle permettant aux ordinateurs d'apprendre sans avoir été programmés explicitement à cet effet. Cette technologie permet de développer des programmes informatiques pouvant changer en cas d'exposition à de nouvelles données. Découvrez la définition, le fonctionnement et les secteurs d'applications du Machine Learning.

L'apprentissage est une méthode d'analyse de données permettant d'automatiser le développement de modèle analytique. Par le biais d'algorithmes capables d'apprendre de manière itérative, le Machine Learning permet aux ordinateurs de découvrir des insights cachées sans être programmés pour savoir où les chercher.

Aux origines, l'apprentissage est né grâce aux technologies de reconnaissance de pattern et à la théorie selon laquelle les ordinateurs peuvent apprendre sans être programmés pour effectuer des tâches spécifiques. Les chercheurs intéressés par l'intelligence artificielle souhaitaient vérifier si les ordinateurs pouvaient apprendre à partir de données. L'aspect itératif du Machine Learning est essentiel, car il permet aux modèles de s'adapter indépendamment lorsqu'ils sont exposés à de nouvelles données. Ils apprennent des précédents calculs pour créer des décisions et des résultats fiables et répétables.

Les méthodes d'apprentissage automatique ont pour but de comprendre la structure des données. Les modèles statistiques reposent sur une théorie prouvée mathématiquement derrière le modèle, mais les données doivent aussi correspondre à certaines caractéristiques. L'apprentissage automatique s'est développé à partir de la capacité à utiliser les ordinateurs pour structurer les données, même si on ignore à quoi cette structure s'apparente. Le test pour un modèle d'apprentissage automatique est la validation d'une erreur sur de nouvelles données, et non un test théorique qui prouve une hypothèse. L'apprentissage automatique repose généralement sur une approche itérative pour apprendre des données, et l'apprentissage peut donc être facilement automatisé. Les tests sont effectués jusqu'à ce qu'une pattern robuste soit trouvée.

Les algorithmes d'apprentissage ne sont pas une nouveauté, mais ce n'est que depuis peu qu'il est possible d'appliquer des calculs mathématiques complexes de plus en plus vite au Big Data. L'apprentissage est aujourd'hui utilisé dans de nombreux domaines, comme le développement de véhicules autonomes, les systèmes de recommandations en ligne comme ceux de Netflix et Amazon, l'analyse de sentiments de clients, ou encore la détection de fraude. Le regain d'intérêt pour l'apprentissage est lié aux mêmes facteurs que ceux qui ont suscité l'attention au tour du data mining et des technologies analytiques. Les données sont de plus en plus nombreuses et diversifiées, la puissance informatique est moins chère que jamais, et le stockage de données est désormais abordable. Par conséquent, il est désormais possible de produire rapidement et automatiquement des modèles capables d'analyser plus rapidement des volumes de données plus importants et plus complexes. Ce faisant, les entreprises ont

davantage de chances d'identifier des opportunités lucratives ou d'éviter les risques inconnus.

### 3.1.1 les secteurs d'application de L'apprentissage automatique

L'apprentissage automatique est utilisé dans de nombreuses industries, et même dans les domaines créatifs comme la peinture ou le cinéma. Les entreprises ont compris l'avantage compétitif procuré par la capacité de collecter des informations en temps réel à partir de données.

Les banques et autres entreprises de l'industrie de la finance utilisent l'apprentissage automatique pour découvrir des informations importantes au sein des données, et pour empêcher la fraude. Les insights permettent d'identifier les opportunités d'investissement, tandis que le Data Mining permet d'identifier les clients à haut risque. La cybersurveillance permet quant à elle de repérer les signes de fraude.

Les agences gouvernementales peuvent utiliser l'apprentissage automatique pour traiter les données en provenance de nombreuses sources qu'elles ont à disposition. En analysant les données de capteurs, par exemple, elles peuvent identifier les possibilités d'augmenter leur efficience et d'économiser de l'argent. L'apprentissage automatique permet également de détecter les fraudes et de réduire le nombre de vols d'identités.

L'apprentissage automatique est de plus en plus utilisé dans le secteur de la santé, notamment grâce à l'essor des objets connectés et autres capteurs permettant d'utiliser les données pour accéder aux données de santé d'un patient en temps réel. Cette technologie peut aussi aider les experts médicaux à analyser les données pour identifier des tendances alarmantes afin d'améliorer les diagnostics et les traitements.

Les sites web qui recommandent des produits basés sur les précédents achats de l'utilisateur utilisent l'apprentissage automatique pour analyser l'historique d'achat des clients et proposer des produits qui pourraient les intéresser. La capacité de collecter les données, de les analyser et de les utiliser pour personnaliser l'expérience de shopping représente le futur de la vente au détail.

L'apprentissage automatique permet également de trouver de nouvelles sources d'énergie, d'analyser les minéraux dans le sol, ou encore de prédire les pannes de capteurs dans les raffineries. Cette technologie permet de rendre la distribution de pétrole plus efficiente et plus économique.

Dans l'industrie des transports, les données sont analysées pour identifier des patterns et des tendances. Ainsi, les itinéraires sont plus efficients et les problèmes potentiels peuvent être prédits pour augmenter la rentabilité. L'analyse de données et les modèles d'apprentissage automatique sont utilisés comme de précieux outils par les entreprises de livraison, les transports publics et les autres entreprises de transport.

## 3.2 Les grandes classes d'apprentissage

Les deux méthodes d'apprentissage les plus couramment utilisées sont l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage profond ou de renforcement.

### 3.2.1 Apprentissage supervisé

L'apprentissage supervisé est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des « exemples » (en général des cas déjà traités et validés).

**Définition mathématique**

Une base de données d'apprentissage est un ensemble de couples entrée-sortie $(x_n, y_n)_{1 \leq n \leq N}$ avec

$x_n \in X$ et $y_n \in Y$ , que l'on considère être tirées selon une loi sur $X \times Y$ inconnue, par exemple $x_n$ suit une loi uniforme et $y_n = f(x_n) + w_n$ où $w_n$ est un bruit centré.
La méthode d'apprentissage supervisé utilise cette base d'apprentissage pour déterminer une représentation compacte de $f$ notée $g$ et appelée fonction de prédiction, qui à une nouvelle entrée $x$ associe une sortie $g(x)$.
Le but d'un algorithme d'apprentissage supervisé est donc de généraliser pour des entrées inconnues ce qu'il a pu « apprendre » grâce aux données déjà traitées par des experts, ceci de façon « raisonnable ».
On distingue deux types de problèmes solvables avec une méthode d'apprentissage automatique supervisée :

1. $Y \subset \mathbb{R}$ : lorsque la sortie que l'on cherche à estimer est une valeur dans un ensemble continu de réels, on parle d'un problème de régression.
2. $Y\{\in 1, ..., I\}$ : lorsque l'ensemble des valeurs de sortie est fini, on parle d'un problème de classification, qui revient à attribuer une étiquette à chaque entrée.

On peut citer quelques méthodes d'apprentissage supervisé les plus connues (et les plus efficaces) :

**Machine à vecteurs de support**

Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais Support Vector Machine, S.V.M) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les S.V.M sont une généralisation des classifieurs linéaires.
Les S.V.M ont été développés dans les années 1990 à partir des considérations théoriques de Vladimir Vapnik sur le développement d'une théorie statistique de l'apprentissage : la Théorie de Vapnik-Chervonenkis. Les S.V.M ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions, le faible nombre d'hyper paramètres, leurs garanties théoriques, et leurs bons résultats en pratique.
Les S.V.M ont été appliqués à de très nombreux domaines (bio-informatique, recherche d'information, vision par ordinateur, finance[1]...). Selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mixture gaussienne.
SVM est une méthode de classification particulièrement bien adaptée pour traiter des données de très haute dimension telles que les textes et les images.
L'idée principale des SVM consiste à projeter les données dans un espace de plus grande dimension appelé, espace de caractéristiques, afin que les données non linéairement séparables dans l'espace d'entrée deviennent linéairement séparables dans l'espace de caractéristiques. En appliquant dans cet espace la technique de construction d'un hyperplan optimal séparant les deux classes, on obtient une fonction de classification qui dépend d'un produit scalaire des images des données de l'espace d'entrée dans l'espace des caractéristiques.
Ce produit scalaire peut être exprimé, sous certaines conditions, par des fonctions définies dans l'espace d'entrée, qu'on appelle les noyaux. **Noyaux ( kernels)** : Le fait de représenter une fonction de décision complexe par un séparateur linéaire dans un espace de caractéristiques de très haute dimensionnalité peut engendrer un coût computationnel élevé. Fréquemment, l'étude des algorithmes d'apprentissage révèle que les seules opérations manipulant les vecteurs de caractéristiques consistent en des produits scalaires. La stratégie du noyau permet alors de substituer ces produits scalaires par une fonction $K(x, x')$ rapidement calculable. Le résultat de $K(x, x')$ doit être équivalent en produit scalaire entre $\phi(x)$ et $\phi(x')$ dans un certain espace de caractéristiques $F$ . Ainsi le noyau $K$ est associé à $\phi$ lorsque : $K(x, x') = \phi(x)\phi(x')$. D'après un théorème[4], une condition nécessaire et suffisante pour qu'une fonction $K : X \times X \to \mathbb{R}$ soit un noyau est :

— La matrice contenant les similarités entre tous les exemples de l'ensemble d'apprentissage $\Big(K(x_i, x'_j)\Big)_{1 \leq i,j \leq n}$ est une matrice symétrique, définie positive.

Ce multiple choix de noyaux rend les SVM plus intéressantes et surtout plus riches puisqu'on peut toujours chercher de nouveaux noyaux qui peuvent être mieux adaptés à la tâche qu'on veut accomplir.

Les trois noyaux les plus utilisés sont : le noyau linéaire, le noyau polynomial et le noyau gaussien noté aussi RBF ( Radial Basis Function ).

**Réseau de neurones**

(L'objet du chapitre suivant)

**Classification naïve bayésienne**

La classification naïve bayésienne est un type de classification Bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance (dite naïve) des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des classifieurs Linéaires.

Un terme plus approprié pour le modèle probabiliste sous-jacent pourrait être « modèle à caractéristiques statistiquement indépendantes ».

En termes simples, un classifieur bayésien naïf suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques. Un fruit peut être considéré comme une pomme s'il est rouge, arrondi, et fait une dizaine de centimètres. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que le fruit est une pomme en considérant indépendamment ces caractéristiques de couleur, de forme et de taille.

Selon la nature de chaque modèle probabiliste, les classifieurs bayésiens naïfs peuvent être entraînés efficacement dans un contexte d'apprentissage supervisé. Dans beaucoup d'applications pratiques, l'estimation des paramètres pour les modèles bayésiens naïfs repose sur le maximum de vraisemblance. Autrement dit, il est possible de travailler avec le modèle bayésien naïf sans se préoccuper de probabilité bayésienne ou utiliser les méthodes bayésiennes.

Malgré leur modèle de conception « naïf » et ses hypothèses de base extrêmement simplistes, les classifieurs bayésiens naïfs ont fait preuve d'une efficacité plus que suffisante dans beaucoup de situations réelles complexes. En 2004, un article a montré qu'il existe des raisons théoriques derrière cette efficacité inattendue [24]. Toutefois, une autre étude de 2006 montre que des approches plus récentes (arbres renforcés, forêts aléatoires) permettent d'obtenir de meilleurs résultats [25].

L'avantage du classifieur bayésien naïf est qu'il requiert relativement peu de données d'entraînement pour estimer les paramètres nécessaires à la classification, à savoir moyennes et variances des différentes variables. En effet, l'hypothèse d'indépendance des variables permet de se contenter de la variance de chacune d'entre elle pour chaque classe, sans avoir à calculer de matrice de covariance.

**Fonctionnement de la méthode bayésien naïve**

Le modèle probabiliste pour un classifieur est le modèle conditionnel

$$p(C|F_1, \ldots, F_n)$$

où $C$ est une variable de classe dépendante dont les instances ou classes sont peu nombreuses, conditionnée par plusieurs variables caractéristiques $\{F_1, \ldots, F_n\}$. Lorsque le nombre de caractéristiques $n$ est grand, ou lorsque ces caractéristiques peuvent prendre un grand nombre de valeurs, baser ce modèle sur des tableaux de probabilités devient impossible. Par conséquent, nous le dérivons pour qu'il soit plus facilement soluble.

À l'aide du théorème de Bayes, nous écrivons :

$$p(C|F_1, \ldots, F_n) = \frac{p(C)\ p(F_1, \ldots, F_n|C)}{p(F_1, \ldots, F_n)}.$$

En introduisant l'hypothèse naïve : si chaque $F_i$ est indépendant des autres caractéristiques $F_{j \neq i}$ , conditionnellement à $C$ alors

$$p(F_1, \ldots, F_n|C) = p(F_1|C)\ p(F_2|C)\ p(F_3|C)\ \cdots\ p(F_n|C) = \prod_{i=1}^{n} p(F_i|C).$$

Le classifieur bayésien naïf couple ce modèle avec une règle de décision. Une règle couramment employée consiste à choisir l'hypothèse la plus probable. Il s'agit de la règle du maximum a posteriori ou MAP.

Le classifieur correspondant à cette règle est la fonction classe suivante :

$$\text{classe}(f_1, \ldots, f_n) = \underset{c}{\text{argmax}}\ p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).$$

où $(f_1, \ldots, f_n)$ est un échantillon donnée.

### 3.2.2 Apprentissage non supervisé

l'apprentissage non supervisé est un problème d'apprentissage automatique. Il s'agit, pour un logiciel, de trouver des structures sous-jacentes à partir de données non étiquetées. Puisque les données ne sont pas étiquetées, il n'est pas possible d'affecter au résultat de l'algorithme utilisé un score d'adéquation. Cette absence d'étiquetage (ou d'annotation) est ce qui distingue les tâches d'apprentissage non-supervisé des tâches d'apprentissage supervisé.
Quand le système ou l'opérateur ne dispose que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé (ou clustering). Aucun expert n'est disponible ni requis. L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données.
On distingue 3 principales tâches réalisables à partir de méthodes d'apprentissage non-supervisé :
Nous allons détailler de manière très succincte le fonctionnement de certains des algorithmes permettant d'effectuer ces tâches.

**Partitionnement des données ou " clustering " :**

Le clustering consiste à regrouper les données en groupes homogènes appelés classes ou clusters, de sorte que les éléments à l'intérieur d'une même classe soient similaires, et les éléments appartenant à deux classes différentes soient différents. Il faut donc définir une mesure de similarité entre deux éléments des données : **la distance**.

**Choix d'une distance :** Chaque élément donnée peut être défini par les valeurs de ses attributs, ou d'un point de vue mathématique par un vecteur ou un point. Le nombre d'éléments de ce vecteur est le même pour tous les éléments et est appelé la dimension du vecteur, notée $n$ ici. Étant donnés deux vecteurs $x_1$ et $x_2$, il faut définir la distance entre ces deux éléments $d(x_1, x_2)$. Parmi les choix les plus classiques, on retrouve :

— **La distance euclidienne**. C'est la distance à vol d'oiseau entre 2 points. Étant donné deux points $A$ et $B$ repérés par leurs coordonnées $X$ et $Y$, on a :

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.1}$$

— **La distance de Manhattan**. Le nom de cette distance est inspirée du fameux quartier de New York, constitué de nombreux gratte-ciels. Il est impossible de se rendre d'un point à un autre à vol d'oiseau, il faut contourner les bâtiments :

$$d(A, B) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.2}$$

— **La distance de Hamming** . Cette distance mesure la similitude entre 2 mots en comptant le nombre de caractères qui diffèrent. Par exemple, la distance de Hamming entre ‘ manche ’ et ‘ mouche ’ est égale à 2. Elle permet de quantifier la différence entre deux séquences de symboles. C'est une distance au sens mathématique du terme. À deux suites de symboles de même longueur, elle associe le nombre de positions où les deux suites diffèrent.

**Principe des algorithmes de clustering** Les algorithmes de clustering consistent à assigner des classes en respectant les règles suivantes :

— La distance entre les éléments d'une même classe (distance intra-classe) est minimale.
— La distance entre chaque classe (distance inter-classes) est maximale.

Toutefois, pour résoudre de manière exacte ce problème, il faudrait essayer toutes les combinaisons possibles d'assignations des éléments à des classes, et choisir la solution ayant les distances intra-classe minimales, et les distances inter-classes maximales. Pour assigner des classes à 27 éléments de manière optimale, il faudrait 115094595070 années à un processeur de 3 GHz pour réaliser cette tâche, soit environ 100 fois l'âge de l'univers... C'est pour cela que l'on utilise différents algorithmes appelés approximations ou heuristiques afin de trouver la solution la plus proche possible de la solution optimale en un temps raisonnable. Nous allons détailler quelques-uns de ces algorithmes.

**algorithme basé sur la représentation : illustration avec k-moyennes (k-means)**
Les algorithmes basés sur la représentation consistent à désigner un représentant pour chaque classe afin de calculer les distances plutôt que calculer des distances avec tous les éléments d'une même classe. Dans k-means, le représentant d'une classe est souvent le barycentre des points de cette classe (généralisation de la moyenne). Le fonctionnement de l'algorithme est le suivant :

**Algorithme de k-means**

**Initialisation** : le nombre de classes $K$ étant imposé, choisir $K$ points aléatoirement pour constituer initialement les représentants de chaque classe.

**Pour chaque point** :

— Calculer les distances entre ce point et les représentants des classes.
— Affecter à ce point la classe avec laquelle sa distance est minimale.
— **Mettre à jour** les représentants de chaque classe (par exemple, calcul de barycentre).

L'algorithme k-means a l'avantage d'être une méthode extrêmement simple à appliquer, mais elle est peu robuste car elle est très sensible aux outliers. Ainsi, ajouter un élément atypique peut complètement modifier le partitionnement des données.

**Clustering hiérarchique** Le fonctionnement du clustering hiérarchique ascendante est le suivant :

**Algorithme de CAH**

**Initialisation** : chaque point est une classe

**Tant qu**'il reste des points à classer :

— Calculer toutes les distances entre les points à classer et les représentants des classes
— **Fusionner** les 2 classes les plus proches
— **Fusionner** les classes les plus proches jusqu'à obtenir le nombre de classes voulu.

Le partitionnement final s'effectue en fusionnant progressivement les feuilles de l'arbre représenté dans la figure suivante ( figure ) :

**Clustering basé sur la densité** Plutôt que de considérer les distances entre les éléments, on peut introduire la notion de densité. Si un élément a de nombreux voisins, il aura une forte densité. Inversement, si tous ses voisins sont éloignés, sa densité sera faible (et par conséquent il sera considéré comme du bruit). Ces méthodes de clustering permettent de s'affranchir du choix du nombre de classes et permettent d'éliminer les éléments aberrants.

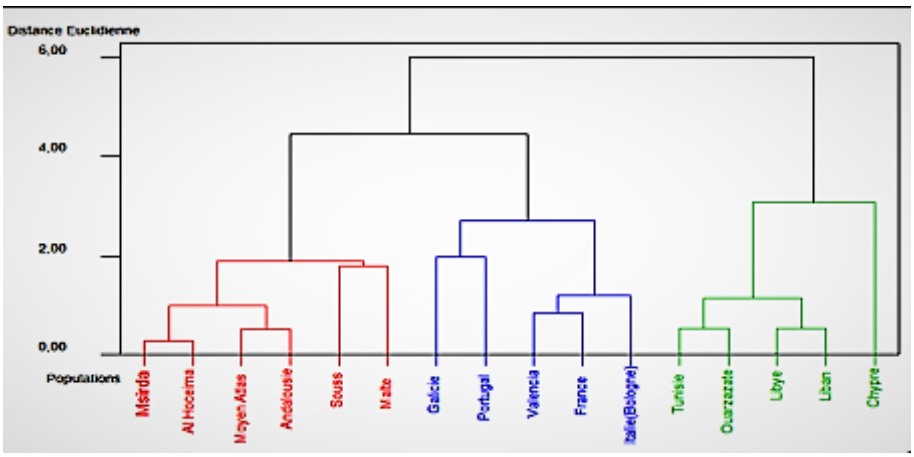



FIGURE 3.1 – Exemple d'un dendrogramme illustrant la méthode CAH

Formellement La densité $D$ d'un point $P$ est définie par le nombre d'éléments à une distance de $P$ inférieure $\epsilon$.
Pour parler de densité, il faut donc fixer le paramètre $\epsilon$, ce qui peut être assez empirique. Un des algorithmes utilisant la densité est DBSCAN :

**Algorithme de DBSCAN**

**Pour** chaque point non visité :
- Calculer sa densité
- **Si** cette densité est plus petite qu'un seuil à préciser, le point est considéré comme bruit, donc associé à aucun cluster
- **Sinon**, affecter ce point et tout son voisinage à la même classe

En particulier, on remarque que DBSCAN procède à une détection des éléments atypiques.

### La réduction de dimensions

Cette tâche apparaît dans le schéma global d'un processus de reconnaissance de formes, comme la sélection/extraction de variables. Cette phase est importante et incontournable, car elle conditionne le processus de classification . Ainsi,deux approches, illustrées à la figure , sont principalement utilisées pour réduire la dimension :

- Une réduction basée sur une sélection de caractéristiques qui consiste à sélectionner les caractéristiques les plus pertinentes à partir de l'ensemble de données des variables décrivant le phénomène étudié.
- Une réduction basée sur une transformation des données appelée aussi une extraction de caractéristiques et qui consiste à remplacer l'ensemble initial des données par un nouvel ensemble réduit,construit à partir de l'ensemble initial de caractéristiques.

### Réduction basée sur une transformation de données :

**La méthode ACP** L'analyse en Composantes principales (ACP) fait partie du groupe des méthodes descriptives multidimensionnelles appelées méthodes factorielles.
L'ACP est une technique qui permet de trouver des espaces de dimensions plus petites dans lesquels il est possible d'observer au mieux les individus.
Sa démarche essentielle consiste à transformer les variables quantitatives initiales, plus ou moins corrélées entre elles, en des variables quantitatives, non corrélées, combinaisons linéaires des variables initiales est appelées composantes principales.
Les composantes principales sont donc de nouvelles variables indépendantes, combinaisons linéaires

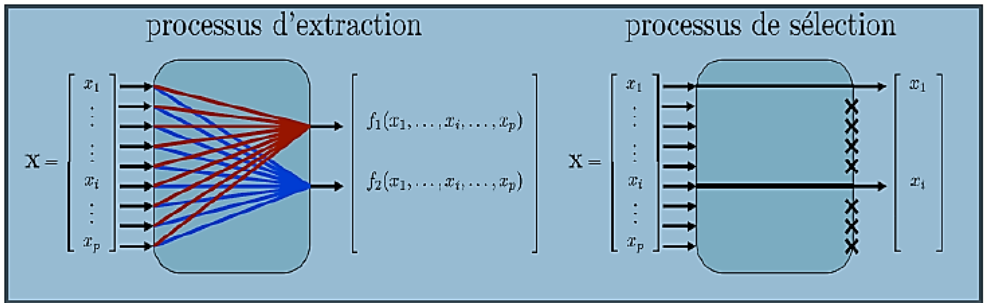



FIGURE 3.2 – Extraction des caractéristiques et sélection des variables

des variables initiales, possédant une variance maximale.
Globalement, l'ACP consiste à rechercher la direction suivant laquelle le nuage de points des observations s'étire au maximum. A cette direction correspond la première composante principale.
La seconde composante principale est déterminée de telle sorte qu'elle soit la plus indépendante possible de la première ; elle est donc perpendiculaire à celle-ci. Ces deux composantes forment le premier plan principal. Cette opération est réitérée de manière à trouver toutes les composantes principales expliquant le maximum de variance.
La figure montre à gauche, une exemple de données 3D qui se trouvent dans un plan 2D et à droite les deux premières composantes principales sur ces données. Supposons que nous avons un ensemble de

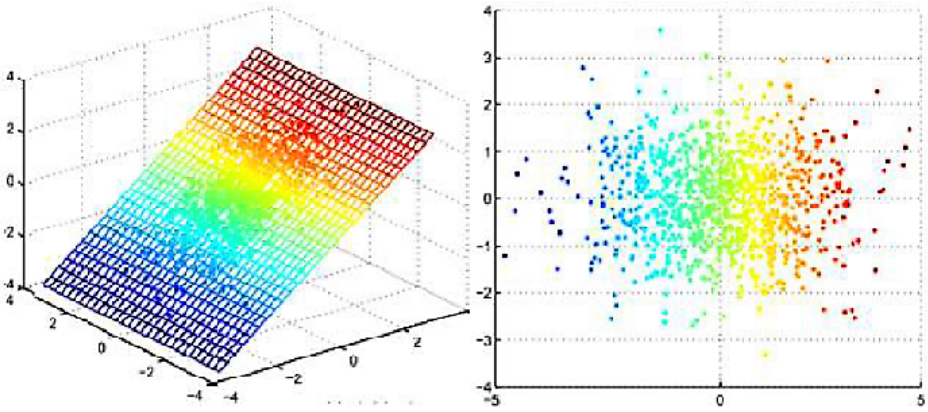


FIGURE 3.3 – ACP sur des données linéaires

données $X = \{x_1, x_2, ..., x_M\}$ composé de $M$ observations où chaque observation $x_i = (x_{i1}, x_{i2}, ..., x_{iN})$ est composée de $N$ caractéristiques. $X$ est associé à une matrice de données $A$ de taille $N \times M$ où chaque colonne représente une caractéristique. En pratique, le calcul de l'ACP pour la matrice $X$ revient à réaliser les opérations ci-dessous afin de trouver les composantes principales :

1. Calculer le vecteur $\mu = (\mu_1, \mu_2, ..., \mu_m)^T$ qui représente le vecteur moyen où $\mu_i$ est la moyenne de la ième composante des données.
2. Calculer la matrice $T$ en soustrayant le vecteur moyen à toutes les colonnes de A dans le but d'obtenir des données centrées.
3. Calculer la matrice $S$ (de taille $N \times N$) de covariance de $T$ avec ($S = TT^T$).
4. Calculer la matrice $U$ (de taille $N \times N$) qui est composée des coordonnées des vecteurs propres $u_j$ de $S$ triés par ordre décroissant des modules des valeurs propres $\lambda_j$ ( la première colonne de $U$ est le vecteur propre qui correspond à la plus grande valeur propre)
5. Garder les $R$ premières colonnes de $U$ pour former la matrice $U$ de dimension ($N \times R$) qui représente les $R$ premières composantes principales.

L'ACP étant une méthode de réduction de dimension, il est important de savoir qu'elle ne peut pas retenir la totalité de l'information contenue dans le nuage de points initial. Enfin , l'ACP prend uniquement en compte les dépendances linéaires entre les variables et ne peut donc pas fournir une projection pertinente pour une distribution non-linéaire de points. La figure montre à gauche, un exemple de données non-linéaires( non réparties dans un plan) et à droite le résultat de leur projection dans un plan générale par les deux premières composantes principales calculées sur ces données.

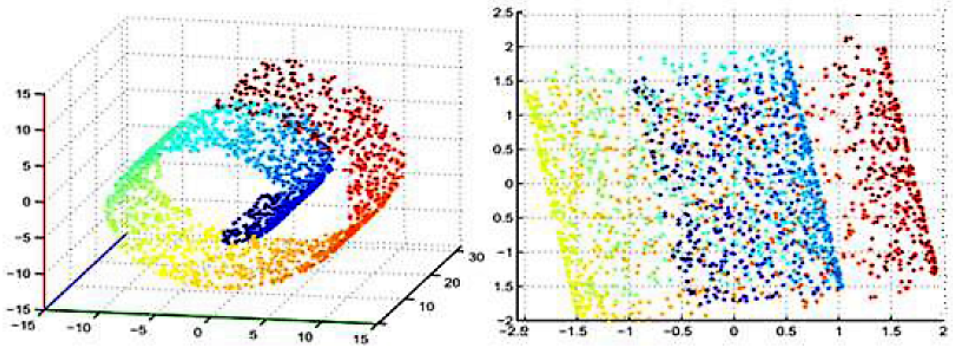


FIGURE 3.4 – ACP sur des données non linéaires

### 3.2.3 Apprentissage profond

D'après des études approfondies des aspects biologiques humaines concernant l'architecture des cerveaux humains, la représentations et la manières d'organisations des idées, ils sont arrivés à découvrir que les cerveaux humains ont une architecture profondes( possédant plusieurs couches cachées organisées d'une manière bien faite), les cerveaux humaines organisent leurs idées de manière hiérarchique et ils permettent aussi de représenter des données très complexes, ce qui a pousser les chercheurs de s'inspirer de cette architecture profonde, de ces manières d'organisation des idées et des données.
L'apprentissage profond, plus fréquemment désigné par l'expression Deep Learning est une branche du machine learning ou apprentissage automatique. Il consiste à apprendre des représentations de haut niveau des données en utilisant des réseaux de neurones. Ces méthodes ont été développées dans les années 1980 mais rapidement abandonnées car considérées comme peu prometteuses. Grâce à l'amélioration de la puissance de calcul des ordinateurs, de l'apparition de nouvelles bases de données plus grandes et plus riches, et de grands progrès dans les techniques d'optimisation, le Deep Learning a récemment permis d'obtenir des performances exceptionnelles pour différentes tâches [26], notamment :

• La classification d'images : Les réseaux de neurones convolutionnels popularisés par le réseau AlexNet [28] ont permis d'atteindre des résultats impressionnants en classification d'images. Celui-ci a gagné en 2012 la compétition ImageNet, ce qui a lancé la grande vague d'intérêt actuelle pour les réseaux de neurones. Les réseaux de convolution ont été successivement améliorés par la suite, notamment avec le VGGNet [29] et les Residual Networks [27].
• Modèles génératifs : Grâce au développement des Auto-encodeurs [30] et surtout récemment des réseaux adversariaux [31], on parvient à générer des images voire des sons extrêmement réalistes mais nouveaux, ou de modifier des images (ajouter des lunettes, vieillir ou rajeunir un visage sur une photo, ...)
• Reconnaissance de la parole : Ce problème a été l'une des premières applications industrielles du Deep Learning [32]
• Traitement automatique du langage naturel pour différents problèmes comme :
  — la réponse automatique à des questions posées en langage naturel, soit en faisant de la recherche dans des données externes [33] ou dans un texte [34], ou encore en devant apprendre à raisonner à partir des informations dans la question [35].

— La traduction automatique a également été largement améliorée, où le deep learning a désormais remplacé dans tous les systèmes les plus modernes les méthodes développées par les linguistes fondées sur une fine étude de la grammaire [36].
— Description automatique d'images en langage naturel [37].

• Intelligence artificielle dans les jeux : La victoire très médiatisée de AlphaGo contre Lee Sedol au Go a montré l'efficacité du deep learning dans ce domaine [38].

l'apprentissage profond combine les avancées dans le domaine de la puissance informatique et les réseaux de neurones artificiels spécifiques pour apprendre des patterns complexes au sein de larges quantités de données. Les techniques de l'apprentissage profond sont actuellement utilisées pour l'identification d'objets dans les images, et de mots dans les sons. Les chercheurs visent à présent à appliquer la reconnaissance de patterns à des tâches plus complexes comme la traduction automatique de langage, les diagnostics médicaux et d'autres problèmes sociaux ou commerciaux importants.

### Les algorithmes d'apprentissage profond

Ils existent plusieurs méthodes d'apprentissage profond, supervisées et non supervisées, les plupart sont basées sur les réseaux de neurones artificiels classiques.

### Auto-encodeur

**Définition** Un auto-encodeur, ou auto-associateur est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé de caractéristiques discriminantes. L'objectif d'un auto-encodeur est d'apprendre une représentation (encodage) d'un ensemble de données, généralement dans le but de réduire la dimension de cet ensemble. Les auto-encodeurs (AE) sont des réseaux de neurones simples composés de trois couches. La première couche et la couche cachée forment l'encodeur, la couche cachée ainsi que la dernière couche formant le décodeur. L'encodeur calcule, à partir de $x$, le vecteur $h$ de taille $m$ (nombre de neurones cachés) ainsi : $h = \sigma(W^1 x + b^1)$ où $W^1$ est une matrice de taille $m \times n$ et $b^1$ un vecteur de biais de taille $m$. $\sigma$ est une fonction d'activation de type tangente hyperbolique définie par

$$\sigma(t) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Le décodeur cherche à reconstruire le vecteur $x$ à partir de la couche cachée $h$. Le résultat de cette reconstruction est le vecteur $\tilde{x}$ tel que : $v = \sigma(W^2 h + b^2))$ où $\tilde{x}$ est un vecteur de taille $m$, $W^2$ est une matrice de poids de taille $n \times m$ et $b^2$ est un vecteur de biais de taille $n$. Durant l'apprentissage, l'auto-encodeur tente de réduire une erreur de reconstruction $e$ entre $x$ et $\tilde{x}$. Il utilise l'erreur quadratique moyenne (M.S.E) :$(e_{MSE}(x, \tilde{x}) = \|x - \tilde{x}\|^2)$ de manière à minimiser l'erreur de reconstruction totale $E_{MSE}$ avec l'ensemble de paramètres $\theta = (W^1, b^1, W^2, b^2)$ :

$$E(x, \tilde{x}) = \frac{1}{d} \sum_{x \in D} (e_{MSE}(x, \tilde{x})$$

**Architecture d'un auto-encodeur simple** La forme la plus simple d'un auto-encodeur est un réseau de neurones non récurrents qui se propage vers l'avant, très semblable au perceptron multicouches ayant une couche d'entrée, une couche de sortie ainsi qu'une ou plusieurs couches cachées les reliant, mais avec toutefois **une couche de sortie possédant le même nombre de nœuds que la couche d'entrée**, son objectif étant de reconstruire ses entrées (plutôt que de prédire une valeur cible $y$ étant donné les entrées $x$). Par conséquent, un auto-encodeur est un modèle d'apprentissage non supervisé. Un auto-encodeur se compose toujours de deux parties, l'encodeur et le décodeur, qui peuvent être définies comme des transitions $\phi$ et $\psi$, telles que :

$$\begin{array}{rcl} \phi & : & X \to F \\ \psi & : & F \to X \\ \phi, \psi & = & arg \min_{\phi,\psi} \|x - (\psi \circ \phi)x\|^2 \end{array}$$

Dans le cas où il n'y a qu'une seule couche cachée, l'étape d'encodage prend l'entrée $x \in \mathbb{R}^d = X$ et l'associe à $z = \sigma(Wx + b) \in \mathbb{R}^p = F$ , $z$ est généralement appelé code, variables latentes ou

Pour chaque entrée $x$,

Effectuer un passage vers l'avant afin de calculer les activations sur toutes les couches cachées, puis sur la couche de sortie pour obtenir une sortie $\mathbf{x}'$ .

Mesurer l'écart entre $x'$ et l'entrée $x$, généralement en utilisant l'erreur quadratique.

représentation latente. $\sigma$ est une fonction d'activation, ( sigmoïde),$W$ une matrice de poids et $b$ un vecteur de biais. Ensuite, l'étape de décodage associe $z$ à la reconstruction $x' = \sigma'(W'z + b')$ de forme identique à $x$.
où $\sigma', W'$, et $b'$ pour le décodeur peut différer de $\sigma, W$, et $\mathbf{b}$ pour l'encodeur, selon la conception de l'auto-encodeur.
Un auto-encodeur est aussi entrainé pour minimiser l'erreur de reconstruction,( erreur quadratique) :

$$L(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b')\|^2$$

où $x$ est généralement la moyenne d'un ensemble de données d'apprentissage.
Si l'espace caractéristique $F$ possède une dimension inférieure à celui de l'espace d'entrée $X$ , alors le vecteur caractéristique $\phi(x)$ peut être considéré comme une représentation compressée de $x$ . Si les couches cachées possèdent une taille plus grandes que celle de la couche d'entrée, l'auto-encodeur peut potentiellement apprendre la fonction identité et devenir sans intérêt. Toutefois, des résultats expérimentaux ont montré que dans ce genre de cas, l'auto-encodeur pouvait malgré tout apprendre des caractéristiques utiles.
Différentes techniques existent pour empêcher un auto-encodeur d'apprendre la fonction identité et améliorer sa capacité à apprendre des représentations plus riches :

**Auto-encodeur débruiteur** : Un auto-encodeur débruiteur prend une entrée partiellement corrompue et apprend à récupérer l'entrée originale débruitée. Cette technique a été introduite avec une approche spécifique d'une bonne représentation. Une bonne représentation est celle qui peut être obtenue de manière robuste à partir d'une entrée corrompue et qui sera utile pour récupérer l'entrée débruitée correspondante. Cette définition contient les hypothèses implicites suivantes :

- Les représentations de haut-niveau sont relativement stables et robustes à la corruption de l'entrée ;
- Il est nécessaire d'extraire des caractéristiques qui soient utiles pour la représentation de la distribution d'entrée.

Pour entrainer un auto-encodeur à débruiter des données, il est nécessaire d'effectuer un mappage stochastique préliminaire $x \to \tilde{x}$ afin de corrompre les données et d'utiliser $\tilde{x}$ comme entrée d'un auto-encodeur classique, avec comme seule exception de calculer la perte pour l'entrée initiale $L(x, \tilde{x}')$ au lieu de $L(\tilde{x}, \tilde{x}')$.

**Auto-encodeur contractif** : L'auto-encodeur contractif ajoute une régulation explicite dans sa fonction objectif qui force le modèle à apprendre une fonction robuste aux légères variations des valeurs d'entrées. Cette régulation correspond à la norme de Frobenius de la matrice Jacobienne des activations de l'encodeur par rapport à l'entrée. La fonction objectif finale à la forme suivante :

$$L(x, x') + \lambda \sum_i \|\nabla_x h_i\|^2. \tag{3.3}$$

**Apprentissage d'Auto-encodeur** L'algorithme d'apprentissage d'un auto-encodeur peut être résumé comme suit :
Un auto-encodeur est bien souvent entrainé en utilisant l'une des nombreuses variantes de la rétropropagation, par exemple : méthode du gradient conjugué, algorithme du gradient. Bien que cela fonctionne de manière raisonnablement efficace, il existe des problèmes fondamentaux concernant l'utilisation de la rétropopagation avec des réseaux possédant de nombreuses couches cachées. Une fois les erreurs

rétropropagées aux premières couches, elles deviennent minuscules et insignifiantes. Cela signifie que le réseau apprendra presque toujours à reconstituer la moyenne des données d'entrainement. Bien que les variantes de la rétropropagation soient capables, dans une certaine mesure, de résoudre ce problème, elles résultent toujours en un apprentissage lent et peu efficace. Ce problème peut toutefois être résolu en utilisant des poids initiaux proches de la solution finale. Le processus de recherche de ces poids initiaux est souvent appelé pré-entrainement.
Les auto-encodeurs sont souvent Utilisés pour :
• compresser de l'information ( moins de neurones sur la couche cachée que sur la couche d'entrée).
• extraire les caractéristiques (composante) d'une base de données (plus de neurones sur la couche cachée que sur la couche d'entrée ).

## 3.3 Réseaux de Neurones Artificiels (RNA)

Dans les années 40, les travaux de Mc Cullochet et Pitts sont l'origine des réseaux de neurones artificiels, ces travaux montraient qu'assembler des entités de calcul simples permet de réaliser des opérations complexes. Dans les années 50, les premières machines ont été apparues, ces machines sont capable d'auto-régler le calcule qu'elle réalisent à partir des exemple qu'on leur présente. C'est l'apparition de perceptron et de l'adaline. Le modèle du neurone formel s'inspire de neurone biologique chez l'être humain. Le neurone formel est un modèle mathématique simplifié du neurone biologique. Les cellules nerveuses, appelées neurones, sont les éléments de base du système nerveux central. Celui-ci en posséderait environ cent milliards. Les neurones possèdent de nombreux points communs dans leur organisation générale et leur système biochimique avec les autres cellules ils présentent cependant des caractéristiques qui leur sont propres et se retrouvent au niveau des cinq fonctions spécialisées qu'ils assurent :

1. Recevoir des signaux en provenance de neurones voisins.
2. Intégrer ces signaux.
3. Engendrer un influx nerveux.
4. Le conduire.
5. Le transmettre à un autre neurone capable de le recevoir.

**Structure des neurones**

Un neurone est constitué de trois parties :

1. Le corps cellulaire.
2. Les dendrites.
3. L'axone.

**corps cellulaire :**
Il contient le noyau du neurone et effectue les transformations biochimiques nécessaire à la synthèse des enzymes et des autres molécules qui assurent la vie du neurone, il a une forme pyramidales ou sphérique dans la plus part des cas.
**Les dendrites :**
Chaque neurone possède une « chevelure » de dendrites Celles-ci sont de fines extensions tubulaires, de quelques dixième de microns. De diamètre et d'une longueur de quelques dizaine de microns. Elles sont les récepteurs principaux du neurone pour capter les signaux qui lui parviennent
**L'axone :**
L'axone, qui est à proprement parler la fibre nerveuse, sert de moyen de transport pour les signaux émis par le neurone Pour former le système nerveux, les neurones sont connectés les uns aux autres suivants des répartitions spatiales complexes, les connexions entre deux neurones se font en des endroits appelés synapses où ils sont séparés par un petit espace synaptique de l'ordre d'un centième de micron. D'une façon simple, on peut dire que le soma du neurone traite les courants électriques qui lui proviennent de ses dendrites, et qu'il transmet le courant électrique (sous forme d'impulsions

chaque une de durée d'environ 1ms et une amplitude d'environ $100mv$) résultant de ce traitement aux neurones aux quels il est connecté par l'intermédiaire de son axone.

Le schéma classique présenté par les biologistes est celui d'un soma effectuant une sommation des influx nerveux transmis par ses dendrites Si la sommation dépasse un certain seuil, le neurone répond par un influx nerveux au potentiel d'action qui se propage le long de son axone. Si la sommation est inférieure à ce seuil, le neurone reste inactif :

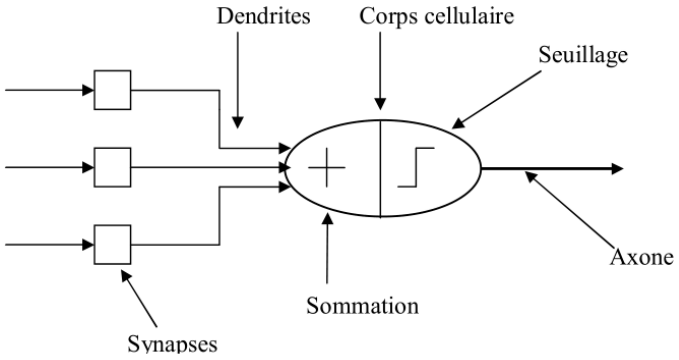



FIGURE 3.5 – Modèle d'un neurone biologique

**Neurone formel** Conçu par Mc Cullochet et pitts, le modèle de neurone formel dérivé du modèle biologique a connu un essor important vers les années 1980. Un neurone formel est une unité de traitement qui reçoit des données en entrée, sous la forme d'un vecteur $x = (x_1, x_2, ..., x_n)$, et produit une sortie $y$. Où :

- les $x_i$ désignent les entrées du neurone,
- les $w_i$ désignent les pondérations ou les connexions. Aussi appelées coefficients synaptiques d'après le terme biologique correspondant,
- $f$ est la fonction d'activation,
- $y$ est la sotie du neurone,

Les neurones formels effectuent deux transformations :

- une sommation pondérée de ses entrées pour calculer le potentiel $a$( appelé aussi l'activation )

$$a = \sum_{i=1}^{n} w_i x_i$$

- Une transformation de l'activation $a$ par une fonction non linéaire $f(.)$, pour calculer la sortie $y$ du neurone.

La fonction d'activation (ou de transfère) calcule la valeur de l'état du neurone. Il existe de nombreuses formes possibles pour la fonction d'activation, las plus courantes sont présentées sur la figure suivante :
a : fonction à seuil( S est la valeur du seuil),

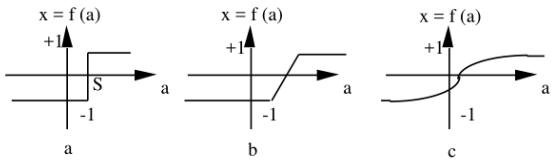



FIGURE 3.6 – Différents types de fonctions d'activation pour le neurone artificiel

b : fonction linéaire par morceaux,

c : fonction sigmoïde.
La plupart des fonctions d'activations sont continues, offrant une infinité de valeur possibles comprises dans l'intervalle $[0, 1]$ ou $[-1, 1]$.
La sortie du neurone est une fonction non linéaire d'une combinaison $\{x_i\}$ pondérées par les paramètres $\{w_i\}$, qui sont souvent désignés sous le non de pondérations.

### 3.3.1 Différentes architectures neuronales

Les connections entre les neurones qui composent le réseau décrivent la topologie ou l'architecture du modèle neuronale. Elles peuvent être quelconque, mais le plus souvent, il est possible de distinguer une certaine régularité. La détermination de l'architecture neuronale optimale influence la convergence des algorithme d'apprentissage : le choix du nombre des couches cachées et des neurones artificiels en chaque couche, la fonction d'activation... etc. Les paramètres optimaux de l'architecture des réseaux de neurones artificiels sont ceux qui donnent une erreur minimale en considération du temps de convergence du processus d'apprentissage.

#### Fonction de transfert du réseau

Quelle que soit l'architecture du réseau, nous considérons qu'il reçoit l'information de l'extérieur par des cellule d'entrés, la propage via des cellule internes qu'il produit une sortie sur une certaine nombre de cellules de sorties, qui constitue la réponse au signal d'entrée.
On peut donc parler de la fonction de transfert du réseau $g(x)$ réalisée par la structure interne( l'architecture ) du réseau. Différentes architectures définissent des fonctions de transfert différentes. On peut ainsi parler de la famille des modèles neuronaux au sein de laquelle on peut distinguer des sous ensembles de réseaux( que nous noterons $F_{A,W}$ caractérisés par une même architecture $A$ et un même espace de pondérations ou de paramètres $W$, c'est à dire les poids du réseau, le réseau calcule des fonctions différentes. De manière générale, on peut voir un RMC comme une fonction de l'espace d'entrée ( de dimension p) vers l'espace de sortie (de dimension c) i.e $g(x) \in F_{A,W}$ où $g : \mathbb{R}^p \to \mathbb{R}^c$.
Prenons le cas particulier d'un réseau de type " feed-forword " de $\mathbb{R}^p$ dans $\mathbb{R}^c$ avec une couche de cellules cachées la fonction de transfert d'un tel réseau est alors le vecteur $g(x) = (g_1, g_2, ..., g_k, ..., g_c)$ avec la $k^{ieme}$ composante de sortie écrite :

$$\begin{aligned} g_k(x) &= f\left(\sum_{j=1}^{h} w_{kj} y_j + w_{k0}\right) \\ &= f\left(\sum_{j=1}^{h} w_{kj} f\left(\sum_{i=0}^{p} w_{ji} x_i\right) y_j + w_{k0}\right) \end{aligned}$$

où $w_{ji}$ et $w_{ki}$sont les poids des première et du deuxième couche respectivement.
Nous dénotons dans la suite l'ensemble des poids d'un réseau par un vecteur $W \in \mathbb{R}^V$, où $V$ est le nombre total des poids. Nous noterons par $p$ la dimension de l'espace d'entrée et $n_h$ est le nombre de neurones cachées. Enfin avec notre notation un poids $w_{ji}$ représente le poids d'une connexion liant le neurone i d'une couche avec le neurone j de la couche suivante, le sens de propagation de l'information étant du neurone i vers le neurone j. Dans la suite, nous ne nous intéressons qu'aux réseaux de type " feed-forward" avec des connexions entre couches. Les fonctions de transition des neurones de la couche cachée et de la couche de sortie seront soit linéaires ou non linéaires.

#### Propriétés d'approximation universelles

Les propriétés d'approximation universelles des RMC ont été formulées et démontrées par de nombreux auteurs. Les réseaux de neurones à trois couches de neurones avec des sorties linéaire, des neurones cachées saturantes croissantes( ce qui inclut les fonctions sigmoïdes) peuvent approcher avec une précision arbitraire toute fonction Borel-mesurable d'un espace de dimension finie dans un autre. En d'autres termes, toute fonction continue peut être approchée avec une précision arbitraire par un réseau de neurones à une couche cachée, pourvu qu'il ait un nombre suffisant de cellules cachées avec fonctions d'activations sigmoïdes[59]. Les PMC constituent donc un outil universel pour l'approximation des fonctions continues d'un espace dans un autre

Cette propriété justifie l'utilisation de l'architecture présentée précédemment. Comme le montre ce théorème, le nombre de neurones cachés doit être choisi convenablement pour obtenir la précision voulue.

**Le perceptron**

Le perceptron est un type des réseaux de neurones à propagation avant les plus utilisés pour des problèmes d'approximation, de classification et de prédiction. Le perceptron peut être considéré comme le premier des réseaux de neurones (Rosenblatt,1958). Le but du perceptron est d'associer des configurations présentées en entrée à des réponses.

**Perceptron simple** Le perceptron simple se compose de deux couches essentielles. La première couche est appelée la rétine du perceptron et elle correspond à la couche d'entrée. La deuxième couche, qui représente la couche de sortie, donne la réponse du perceptron correspondant à la configuration donnée en entrée

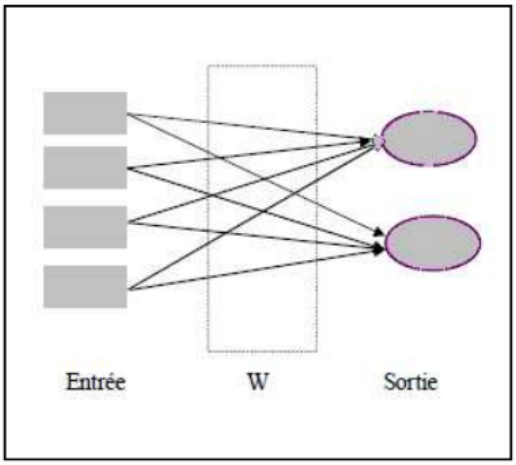



FIGURE 3.7 – Exemple de perceptron simple

Les neurones de la couche d'entrée (rétine) sont reliés aux neurones de la couche de sortie grâce à des pondérations d'intensité variable.
Les neurones de sortie effectuent une somme des intensités des neurones actives suivant la formule :

$$a_j = \sum_{i=1}^{n} w_{ji} x_i$$

•$a_j$ : Activation du $j^{eme}$ neurone de sortie,
•$x_i$ : Valeur de sortie du i ème neurone de la rétine,
•$w_{ji}$ : Intensité de la connexion entre la $i^{ieme}$ neurone de la couche d'entrée et le $j^{eme}$ neurone de le couche de sortie,
Les neurones de sortie deviennent actifs si leur degré d'activation dépasse un seuil fixé noté $\theta j$. Le plus souvent, le seuil $\theta j$ est fixé à zéro, mais sans que cela soit nécessaire avec une formule :

$$a_j = \begin{cases} 1 \text{ pour } a_j \geq \theta_j \\ 0 \text{ pour } a_j < \theta_j \end{cases}$$

Pour se ramener à un seuil nul, on fait intégrer un neurone supplémentaire noté $x_0$ toujours actifs avec $w_{j0} = -\theta_j$, dont $x_0 = 1$ puisque il est toujours actif. L'activation peut donc se réécrire sous la forme est :

$$a_j = \sum_{i=0}^{n} w_{ji} x_i = w_{j0} + \sum_{i=1}^{n} w_{ji} x_i$$

En conséquence, la règle de décision est :

$$a_j = \begin{cases} 0 \text{ pour } a_j \leq 0 \\ 1 \text{ pour } a_j > 0 \end{cases}$$

**Perceptron Multi-couches PMC** Le Perceptron Multi-couches est un réseau orienté de neurones artificiels( a propagation avant) organisés en couches, où l'information voyage dans un seul sans, de la couche d'entrée vers la couche de sortie. chaque couche du réseau regroupe un ensemble de neurones artificiels dont le nombre peut varier d'une couche à une autre. Il est habituellement constitué d'au moins trois couches totalement connectés.
Voici un exemple d'un PMC comporte trois couches : une couche d'entrée, une couche cachée et couche de sorties.

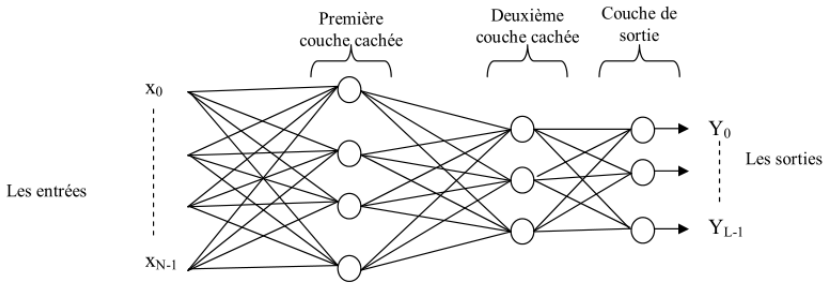



FIGURE 3.8 – Exemple de perceptron multi-couche

— La couche d'entrée : ne joue généralement que le rôle d'interface avec l'extérieur, c'est à dire qu'elle n'effectue pas de traitement sur l'information.
— Une ou plusieurs couches cachées : appelées aussi couches de traitement ou couches intermédiaires, leurs intérêt est d'augmenter le nombre de connexions, ce qui accroît la capacité d'un réseau à extraire l'information des données fournies en entrée.
— La couche de sortie : traite également l'information qu'elle reçoit de la couche précédente mais elle ne communique pas son résultat aux autres neurones. L'information fournie par chacune de ses unités constitue le vecteur de sortie du réseau. Celui-ci est donc le résultat du traitement d'un vecteur d'entrée par l'ensemble du réseau.

Les neurones des couches consécutives sont interconnectés. Chaque neurone calcule la somme pondérée des valeurs de ses entrées, et calcule en sortie cette somme modifiée par une fonction d'activation, qui est souvent la fonction sigmoïde :

$$f(x) = \frac{1}{1 + e^{-x}}$$

Le réseau de neurone réalise en fait une fonction, qui est déterminée par les pondérations qui sont associées aux connexions entre les neurones ; Le plus grand atout des réseaux neuronaux réside dans leur aptitude à l'apprentissage. Cela veut dire qu'il existe des algorithmes qui déterminent les poids tels que le réseau fonctionne d'une façon souhaitée. En effet, dans beaucoup de situations, on ne connait pas explicitement une fonction mais on possède des paires d'entrées /sorties qui sont représentatives du comportement recherché. les algorithmes d'entrainement utilisent cette base d'apprentissage pour déterminer les pondérations. après un apprentissage, le réseau reproduit la bonne sortie non seulement pour les points de la base, mais aussi pour d'autres entrées, c'est ce que l'on appelle la généralisation. Dans la suite nous utiliserons la terminologie suivante. les paramètres associés aux connexions, les poids, seront notés $W$. Le type de connexions, le nombre de neurones utilisés, ainsi que la nature de ces neurones et ses fonctions de transfert définissent l'architecture du réseau, que nous noterons $A$.

## 3.4 Apprentissage du PMC

### 3.4.1 Algorithme de Rétro-propagation du gradient

Le problème de l'apprentissage dans les perceptrons multi-couches est de connaitre la contribution de chaque poids dans l'erreur globale du réseau. L'algorithme de rétro-propagation de l'erreur permet de faire cela :

1. Propagation de l'entrée jusqu'à la sortie
2. Calcul de l'erreur en sortie
3. Rétro-propagation de l'erreur jusqu'aux entrées

Nous allons maintenant déterminer les équations de mise à jour des pondérations :
Pour la couche de sortie : Considérons un neurone $j$ de la couche de sortie. Lorsque l'exemple numéro $n$ lui est présenté, il produit la sortie $y_j(n)$ alors que c'est la valeur $d_j(n)$ qui est attendue. Notons

$$e_j(n) = d_j(n) - y_j(n)$$

l'erreur associée à ce neurone. L'erreur globale du réseau sur la pattern $n$ est

$$E(n) = \frac{1}{2}\sum_{j=1}^{m} e_j^2(n)$$

avec $m$ est le nombre de neurones dans la couche de sortie.
Le but de l'apprentissage est de minimiser l'erreur moyenne correspondante aux $N$ exemples d'apprentissage, c'est à dire minimiser

$$E_{moy} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

Soit

$$u_j(n) = \sum_i w_{ij}(n)y_i(n) - b_j(n)$$

l'activité interne d'un neurone dont la fonction d'activation est $f_j$ . Sa sortie s'écrit alors sous forme de

$$y_j(n) = f_j(u_j(n))$$

Le mécanisme de rétro-propagation est basé sur une correction $\Delta w_{ji}(n)$ et $\Delta b_j(n)$. En ce qui concerne les poids, on utilisera la règle dite du delta(delta rule), c'est à dire que la modification sera proportionnelle au gradient suivant :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_(n)}{\partial u_j(n)}\frac{\partial u_j(n)}{\partial w_{ji}(n)}$$

qui détermine la direction dans laquelle on recherche les valeurs de $w_{ij}(n)$. Nous appellerons taux d'apprentissage le facteur de proportionnalité et nous le noterons $\eta$ par la suite. D'après l'équation 2.2,on peut écrire :

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n)$$

de plus

$$\frac{\partial e_j(n)}{\partial y_j(n)} = f_j'(u_j(n))$$

par suite

$$\frac{\partial y_(n)}{\partial u_j(n)} = y_i(n)$$

Donc, on défini $\delta_j(n)$ le gradient local par

$$\delta_j(n) = -\frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}$$

d'après ce qui précède ceci est aussi égal à

$$\delta_j(n) = e_j(n) f'_j(u_j(n))$$

La correction appliquée au poids $W_{ji}(n)$ sera donc la suivante (règle du delta) :

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} = \eta \delta_j(n) y_i(n)$$

De même, la correction apportée au biais s'écrira sous la forme

$$\Delta b_j(n) = \eta \frac{\partial E(n)}{\partial b_j(n)}$$

en suivant un raisonnement identique à ce qui précède on obtient

$$\begin{aligned} \frac{\partial E(n)}{\partial b_j(n)} &= \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j(n)} \frac{\partial u_j(n)}{\partial b_j(n)} \\ &= -\delta_j(n) \frac{\partial u_j(n)}{\partial b_j(n)} \\ &= -\delta_j(n) \end{aligned}$$

D'où

$$\Delta b_j(n) = -\eta \delta_j(n)$$

Ces deux équations de mise à jour sont valables pour la couche de sortie d'un réseau de neurones car nous avons utilisé l'erreur de sortie $e_j$.
Pour les couches cachées : En ce qui concerne les autres couches, on ne peut utiliser cette formulation pour l'erreur, voyons alors quel les vont être les nouvelles équations de mise à jour. Pour cela considérons un neurone $j$ sur la dernière couche cachée, nous pouvons définir le gradient local par

$$\begin{aligned} \Delta b_j(n) &= \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j(n)} \\ &= \frac{\partial E(n)}{\partial y_j(n)} f'_j(u_j(n)) \end{aligned}$$

En utilisant la formule de l'erreur, on peut écrire

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j} &= \textstyle\sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_k(n)} \\ &= \textstyle\sum_k e_k(n) \frac{\partial e_k(n)}{\partial u_k(n)} \frac{\partial u_k(n)}{\partial y_j}(n) \end{aligned}$$

Or le neurone $k$ est sur la couche de sortie, d'où

$$\begin{aligned} \frac{\partial e_k(n)}{\partial u_k(n)} &= \frac{\partial (d_k(n) - y_k(n)}{\partial u_k(n)} \\ &= -f'_k(u_k(n)) \end{aligned}$$

de plus, puisque $\dfrac{\partial u_k(n)}{\partial y_j(n)} = w_{kj}(n)$, nous obtenons alors

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j} &= -\textstyle\sum_k e_k(n) w_{kj}(n) f'_k(u_k(n)) \\ &= -\textstyle\sum_k \delta_k(n) w_{kj}(n) \end{aligned}$$

Le gradient local pour un neurone de la dernière couche cachée est alors donné par :

$$\delta_j(n) = \left( -\sum_k \delta_k(n) w_{kj}(n) \right) f'_k(u_k(n))$$

Ceci pouvant se généraliser aux autres couches internes, nous avons ainsi obtenu les règles de mise à jour des poids et des biais avec la méthode de rétro-propagation du gradient. Ainsi, les poids et les biais doivent être actualisés en utilisant :

$$\begin{array}{rcl} w_{ji} & \longleftarrow & w_{ji} + \eta\delta_j(n)y_i(n) \\ b_j & \longleftarrow & b_j - \eta\delta_j(n) \end{array}$$

avec

$$\delta_j(n) = e_j(n)f'_j(u_j(n))$$

et

$$e_j(n) = \begin{cases} d_j(n) - y_j(n) \text{ si } j \text{ est un neurone de sortie} \\ \sum_k \delta_k(n)w_{kj} \text{ si } j \text{ est un neurone d'une couche cachée} \end{cases}$$

Ainsi, l'erreur de sortie se trouve propagée vers l'entrée (c'est à dire dans le sens inverse de celui qui est utilisé pour propager un signal au travers d'un réseau) afin de corriger de couche en couche les valeurs des poids et des biais.

**Algorithme de retro-propagation du gradient** Considérons qu'il existe une base de donnée étiqueté $(x(n), d(n))_{n=1,...,N}$. L'algorithme de retro-propagatiopn du gradient est donné par :

| **Algorithme de retro-propagatiopn du gradient** |
|---|
| **1.Initialisation** :<br>initialiser tous les poids et les biais du réseau de neurones. |
| **2.Présentation d'un exemple en entrée :**<br>Présenter un exemple en entrée $x(n) = (x_1(n), x_2(n), ..., x_p(n))$ ainsi que la sortie désirée correspondante $d(n) = (d_1(n), d_2(n), ..., d_m(n))$. |
| **3. Calculer les sorties :**<br>calcules successivement les sorties des différentes couches pour l'entrée $x$. |
| **4. La mise à jour des poids et des biais :**<br>modifier récursivement les poids et les biais du réseau suivant les règles détaillées avant :<br>$\begin{array}{rcl} w_{ji} & \longleftarrow & w_{ji} + \eta\delta_j(n)y_i(n) \\ b_j & \longleftarrow & b_j - \eta\delta_j(n) \end{array}$<br>avec<br>$\delta_j(n) = e_j(n)f'_j(u_j(n))$<br>et<br>$e_j(n) = \begin{cases} d_j(n) - y_j(n) \text{ si } j \text{ est un neurone de sortie} \\ \sum_k \delta_k(n)w_{kj} \text{ si } j \text{ est un neurone d'une couche cachée} \end{cases}$ |
| **5. Critère d'arrêt :**<br>Répéter les étapes 2,3 et 4 jusqu'à un nombre maximum d'itérations ou jusqu'à ce que l'erreur quadratique moyenne soit inférieure à un certain seuil. |

### 3.4.2 Problème de surajustement et Méthodes pour le limiter

**Problème de surajustement**

**Définition** Si l'on considère un ensemble d'apprentissage et une fonction de coût quadratique, en vertu de la propriété d'approximation universelle exposée précédemment , il est toujours possible

d'obtenir une fonction de coût aussi petite que l'on veut sur l'ensemble d'apprentissage, à condition de mettre suffisamment de neurones cachés. Cependant, le but de l'apprentissage n'est pas d'apprendre exactement la base d'apprentissage, mais le modèle sous-jacent qui a servi à engendrer les données. Or, si la fonction apprise par le réseau de neurones est ajustée trop finement aux données, elle apprend les particularités de la base d'apprentissage au détriment du modèle sous-jacent : le réseau de neurones est surajusté.

**Méthodes pour limiter le surajustement**

On distingue deux familles de méthodes pour prévenir le surajustement : les méthodes passives et les méthodes actives. Les philosophies de ces deux familles de méthodes sont différentes.

— Les méthodes passives Elles essayent de détecter le surajustement à posteriori pour supprimer les mauvais modèles. Parmi les méthodes les plus classiquesfigurent l'utilisation d'une base de validation pendant l'apprentissage, et les mesures de critère d'information.

— Les méthodes actives interviennent pendant la phase d'apprentissage pour empêcher le modèle de faire du surajustement. Les méthodes de régularisation comme l'arrêt prématuré ou la pénalisation entrent dans ce cadre.

**Méthodes passives** Les méthodes passives, issues de la description du problème de surajustement en termes de biais-variance ne propose comme solution qu'une limitation de la complexité du modèle par l'intermédiaire d'une limitation du nombre de neurones cachés.

**Méthodes actives : les méthodes de régularisation** Les méthodes de régularisation, par opposition, peuvent être qualifiées d'actives, car elles ne cherchent pas à limiter la complexité du réseau, mais elles contrôlent la valeur des poids pendant l'apprentissage.Il devient possible d'utiliser des modèles avec un nombre élevé de poids et donc un modèle complexe, même si le nombre d'exemples d'apprentissage est faible.
Bartlett (1997 )a montré que la valeur des poids était plus importante que leur nombre afin d'obtenir de modèles qui ne sont pas surajustés. Il montre, que si un grand réseau est utilisé et que l'algorithme d'apprentissage trouve une erreur quadratique moyenne faible avec des poids de valeurs absolues faibles, alors les performances en généralisation dépendent de la taille des poids plutôt que de leur nombre. Plusieurs méthodes de régularisation existent dans la littérature, à savoir :
**a) L'arrêt prématuré( early stopping) :**
Comme l'apprentissage consiste à minimiser, grâce à un algorithme itératif, une fonction de coût calculée sur la base d'apprentissage. La méthode de l'arrêt prématuré (early stopping) consiste à arrêter les itérations avant la convergence de l'algorithme. Si la convergence n'est pas menée à son terme, le modèle ne s'ajuste pas trop finement aux données d'apprentissage :le surajustement est limité.
**b) Weight Decay :**
Lorsque les poids du réseau sont grands en valeur absolue, les sigmoïdes des neurones cachés sont saturées, si bien que les fonctions modélisées peuvent avoir des variations brusques. Pour obtenir des fonctions régulières, il faut travailler avec la partie linéaire des sigmoïdes, ce qui implique d'avoir des poids dont la valeur absolue est faible. La méthode de régularisation du weight decay limite la valeur absolue des poids en faisant l'apprentissage en minimisant l'erreur régularisée suivante :

$$E' = E + \frac{\alpha}{2} \sum_{i=1}^{p} w_i^2$$

où $p$ est le nombre de poids que comporte le réseau.

## 3.5 Conclusion

Les réseaux de neurones s'avèrent un outil très puissant que ça soit pour des problèmes de classification, de régression et de prédiction dans divers domaines d'applications grâce à leurs propriétés

universelles.
Dans ce chapitre nous avons présenté les concepts généraux d'un réseau de neurones,et en particulier le modèle perceptron multicouches ainsi que son algorithme d'apprentissage. Nous avons également énoncé le problème de sur-ajustement ainsi que quelques méthodes pour y éviter.

# Chapitre 4

# Application au problème de classification

## 4.1 Le problème de classification

### 4.1.1 Introduction

La classification automatique des données consiste à attribuer automatiquement une classe à un élément de donnée à l'aide d'un système de classification. On retrouve ainsi la classification d'objets, de scènes, de textures, la reconnaissance de visages, d'empreintes digitale et de caractères... Comme on a vu dans le chapitre précédent, il existe deux principaux types de classification : la classification supervisée et la classification non-supervisée. Dans l'approche supervisée, chaque élément des données est associée à une étiquette qui décrit sa classe d'appartenance. Dans l'approche non supervisée les données disponibles ne possèdent pas d'étiquettes. Dans toute la suite on s'intéresse de l'approche supervisée.

### 4.1.2 Méthodes de classification

#### Méthodes supervisées

L'objectif de la classification supervisée est principalement de définir des règles permettant de classer des objets dans des classes à partir de variables qualitatives ou quantitatives caractérisant ces objets. On dispose au départ d'un échantillon dit d'apprentissage dont le classement est connu. Cet échantillon est utilisé pour l'apprentissage des règles de classement.
Il est nécessaire d'étudier la fiabilité de ces règles pour les comparer et les appliquer, évaluer les cas de sous apprentissage ou de sur apprentissage (complexité du modèle). On utilise souvent un deuxième échantillon indépendant, dit de validation ou de test.

#### Méthodes non supervisées

Procède de la façon contraire. C'est à dire ne nécessitent aucun apprentissage et aucune tâche préalable d'étiquetage manuel. Elle consiste à représenter un nuage des points d'un espace quelconque en un ensemble de groupes appelé Cluster. Il lié généralement au domaine de l'analyse des données comme l'ACP. Un « Cluster » est une collection d'objets qui sont « similaires » entre eux et qui sont « dissemblables » par rapport aux objets appartenant à d'autres groupes( voir chapitre réseau de neurones artificiels et apprentissage).

### 4.1.3 Indicateurs de performance en classification

#### Matrice de confusion

Prenons l'exemple d'un classifieur binaire, c'est-à-dire, qui prédit 2 classes notées classe 0 et classe 1.

Pour mesurer les performances de ce classifieur, il est d'usage de distinguer 4 types d'éléments classés pour la classe voulue :

1. **Vrai positif $\mathbf{V}_p$** : $\acute{E}lémentdelaclasse1correctementprédit$, Ces informations peuvent être rassemblés et visualisés sous forme de tableau dans une matrice de confusion. Dans le cas d'un classifieur binaire, on obtient : En particulier, si la matrice de confusion est diagonale, le classifieur est parfait.

| | | Classe prédite | |
|---|---|---|---|
| | | Classe 0 | Classe 1 |
| Classe réelle | Classe 0 | VN | FN |
| | Classe 1 | FP | VP |

2.

FIGURE 4.1 – Matrice de confusion dans le cas binaire

Notons que la matrice de confusion est aussi généralisable lorsqu'il y a $k > 2$ classes à prédire.
Il est possible de calculer plusieurs indicateurs résumant la matrice de confusion. Par exemple si nous souhaitons rendre compte de la qualité de la prédiction sur la classe 1, on définit :

— **Précision.** Proportion d'éléments bien classés pour une classe donnée :

$$P_{c_1} = \frac{V_p}{V_p + F_p}$$

— **Rappel.** Proportion d'éléments bien classés par rapport au nombre d'éléments de la classe à prédire :

$$R_{c_1} = \frac{V_p}{V_p + F_n}$$

— **F-mesure.** Mesure de compromis entre précision et rappel :

$$M_{c_1} = \frac{2 * P_{c_1} * R_{c_1}}{P_{c_1} + R_{c_1}}$$

Il est possible de calculer tous ces indicateurs pour chaque classe. La moyenne sur chaque classe de ces indicateurs donne des indicateurs globaux sur la qualité du classifieur.

$$\begin{array}{rcl} P & = & \frac{1}{k}\sum_{i=1}^{k}\frac{V_{pi}}{V_{pi}+F_{pi}} \\ R & = & \frac{1}{k}\sum_{i=1}^{k}\frac{V_{pi}}{V_{ni}+F_{ni}} \\ M & = & \frac{2*P*R}{P+R} \end{array}$$

**Courbe ROC (Received Operating Characteristic) :**

Dans le cas d'un classifieur binaire, il est possible de visualiser les performances du classifieur sur ce que l'on appelle une courbe ROC. La courbe ROC est une représentation du taux de vrais positifs en fonction du taux de faux positifs. Son intérêt est de s'affranchir de la taille des données de test dans le cas où les données sont déséquilibrées.
Cette représentation met en avant un nouvel indicateur qui est l'aire sous la courbe. Plus elle se rapproche de 1, plus le classifieur est performant.

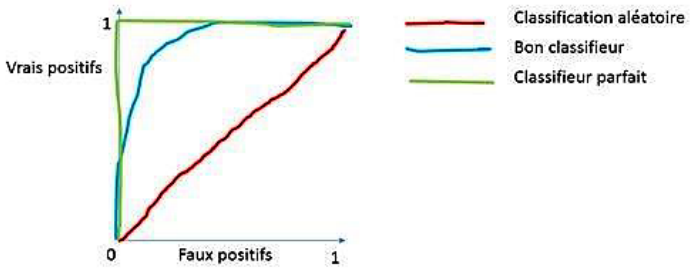



FIGURE 4.2 – Courbe ROC

### 4.1.4 Classification des données par le PMC

**Les bases utilisées**

Dans cette application, afin de tester la performance du PMC comme classifieur supervisé sur données non linéairement séparables, on va travailler avec trois bases, les paramètres de ces données se résument dans le tableau(4.1) : Les attributs des trois bases seront codées pour qu'elles soient utilisées

| **La base** | Le nombre d'attributs | le nombre de classes |
|---|---|---|
| **Iris** | 150 | 3 |
| **La base Post-Operative Patient** | 8 | 3 |
| **Wine** | 13 | 3 |

TABLE 4.1 – Description des bases utilisées

comme entrées au perceptron multicouche correspondant dans la phase d'apprentissage et dans la phase de test.

**Description des bases**

**La base Iris**

il s'agit d'une base donnée peut être la plus connue dans la littérature sur la reconnaissance des formes, l'ensemble des donnés contient 3 classes de 50 instances chacune, où chaque classe réfère à un type d'usine d'iris(un type des plantes ). Ces derniers ne sont pas linéairement séparables les uns des autres, pour cela, nous proposons de de faire une classification de ces types de plantes selon leurs formes.
La base de donnée Iris contient 4 informations concernant la forme de l'iris : la longueur et la largeur des sépales et les pétales de l'iris, ces information sont représentées dans un vecteur de 4 composantes, sous la forme :

* $x_1$ : longueur du sépale,
* $x_2$ : largeur du sépale,
* $x_3$ : longueur du pétale,
* $x_4$ : largeur du pétale,

La base d'iris contient 3 types différentes, donc on a besoin de 3 neurones dans la couches de sortie, de telle sorte que chaque neurone représente un type.
La sortie désirée associée a un vecteur des caractéristiques d'une image d'iris de type i est le vecteur nulle sauf dans la case i qui va contenir 1, i.e, la sortie désirée associée au type 1 est définie par le vecteur $(1, 0, 0)$ 'Iris-setosa', la sortie désirée associée au type 2 est définie par le vecteur $(0, 1, 0)$ 'Iris-versicolor', et celle du type 3 est définie par le vecteur $(0, 0, 1)$ 'Iris-virginica'.
**Architecture du réseau utilisé :**

- Le vecteur des caractéristiques est de taille 4, alors on a besoin de 5 neurones dans la couche d'entrée, de telle sorte que chaque neurone représente une caractéristique, plus un neurone contient les biais de la couche cachée.
- Un vecteur de dimension $k$, issu d'une couche cachée,
- Le vecteur de sortie de taille 3 contenant :
  — $c_1$ : la classe d'iris Iris Setosa
  — $c_2$ : la classe d'iris Versicolour
  — $c_3$ : la classe d'iris Virginica

L'architecture du réseau utilisé est représentée dans la figure (4.3) :

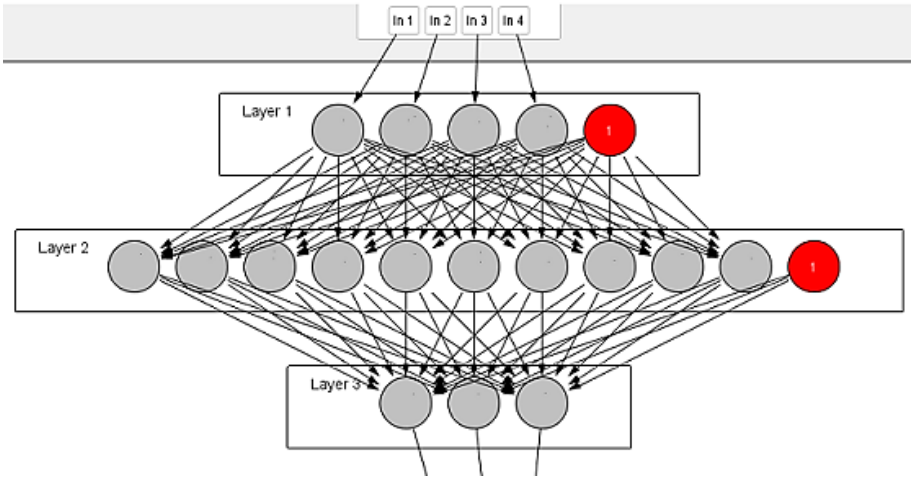



FIGURE 4.3 – Perceptron utilisé pour classifier la base d'Iris

**La base Post-Operative Patient**

Cette base contient des informations sur des patients (les constantes vitales des patients), ces informations se décrivent par les variables suivantes :

— La température interne du patient, elle sera élevée si cette température vaut une valeur supérieur à $37C$, moyenne si elle prend une valeur comprise entre $37C$ et $36C$, faible si cette dernière prend une valeur inférieur à $36C$.
— La température de surface du patient, elle sera élevée si cette température vaut une valeur supérieur à $36.5C$, moyenne si elle prend une valeur comprise entre $36.5C$ et $35C$, faible si cette dernière prend une valeur inférieur à $35C$.
— Le taux de saturation en oxygène, ce taux sera excellent s'il est supérieur à 98% , bon si ce dernier compris entre 98% et 90% , il va être passable s'il est compris entre 90% et 80%, médiocre s'il est inférieur à 80% .
— Une mesure récente de la pression artérielle, cette mesure sera élevée si elle est supérieur à $\frac{130}{90}$, elle sera moyenne cette dernière est comprise entre les valeurs $\frac{130}{90}$ et $\frac{90}{70}$, et elle est faible si cette dernière vaut une valeur inférieur à $\frac{90}{70}$.
— La stabilité au niveau de la température de surface du patient, cette dernière sera soit stable, soit mod stable, soit instable.
— La stabilité de la température centrale du patient, cette dernière sera aussi soit stable, soit mod stable, soit instable.
— La stabilité de la pression artérielle du patient, cette dernière sera aussi soit stable, soit mod stable, soit instable.
— Le confort perçu par le patient, il s'agit d'une mesure comprise entre les valeurs 0 et 20.

Cette base contient trois cas des patients, de décision ADM-DECS (décision de décharge) :

— I (patient envoyé à l'unité de soins intensifs),
— S (patient préparé pour rentrer à la maison),
— A (patient envoyé à l'étage général de l'hôpital)

donc l'architecture du réseau utilisée doit contenir de 3 neurones dans la couches de sortie, de telle sorte que chaque neurone représente un type.
La sortie désirée associée à un vecteur des caractéristiques d'un exemple de type i est le vecteur nulle sauf dans la case i qui va contenir 1, i.e, la sortie désirée associée au type 1 est définie par le vecteur $(1, 0, 0)$ 'I', la sortie désirée associée au type 2 est définie par le vecteur $(0, 1, 0)$ 'S', et celle du type 3 est définie par le vecteur $(0, 0, 1)$ 'A'.

**Architecture du réseau utilisé :**

- Le vecteur des caractéristiques est de taille 8, alors on a besoin de 9 neurones dans la couche d'entrée, de telle sorte que chaque neurone représente une caractéristique, plus un neurone contient les biais des neurones de la couche cachée.
- Un vecteur de dimension $k$, issu d'une couche cachée,
- Le vecteur de sortie contient :
  — $c_1$ : La classe des patients envoyés à l'unité de soins intensifs,
  — $c_2$ : La classe des patients préparés pour rentrer à la maison,
  — $c_3$ : La classe des patients envoyés à l'étage général de l'hôpital.

Le vecteur des caractéristiques contient :

— $x_1$ : L-CORE (température interne du patient en C) : élevé (de code 1) (supérieur à 37), moyen (de code 2) ( entre 36 et 37), faible (de code 3)( inférieur à 36),
— $x_2$ : L-SURF (température de surface du patient en C) : élevé (de code 1) (supérieur à 36,5), moyen (de code 2)(entre 36,5 et 35), faible (de code 3)(inférieur 35),
— $x_3$ : L-O2 (saturation en oxygène en) : excellent (de code 1)(supérieur à 98), bon (de code 2)(entre 90 et 98), passable (de code 3)(entre 80 et 90), médiocre (de code 4)(inférieur à 80),
— $x_4$ : L-BP (dernière mesure de la pression artérielle) : élevé (de code 1) (supérieur à 130/90), moyen (de code 2)(entre 130/90 et 90/70), faible (de code 3)(inférieur à 90/70),
— $x_5$ : SURF-STBL (stabilité de la température de surface du patient) : stable (de code 1), mod stable (de code 2), instable (de code 3),
— $x_6$ : CORE-STBL (stabilité de la température centrale du patient) stable (de code 1), mod stable (de code 2), instable (de code 3),
— $x_7$ : BP-STBL (stabilité de la pression artérielle du patient) stable (de code 1), mod stable (de code 2), instable (de code 3),
— $x_8$ : CONFORT (confort perçu du patient à la sortie, mesuré un nombre entier compris entre 0 et 20).

Donc, un vecteur d'entrés est un vecteur de 8 composantes.
concernant la sortie ou la décision ADM-DECS (décision de décharge), les données sont étiquetées par : I (de code (1 0 0))(patient envoyé à l'unité de soins intensifs),
S (de code (0 1 0))(patient préparé pour rentrer à la maison),
A (de code (0 0 1))(patient envoyé à l'étage général de l'hôpital). L'architecture du PMC utilisé dans ce cas est données par la figure (4.4) :

**Base Wine**

Les données contenues dans cette base sont le résultat d'une analyse chimique de vins cultivés dans la même région en Italie mais issus de trois cultivars différents. L'analyse a déterminé les quantités de 13 constituants trouvés dans chacun des trois types de vins.

La base Wine contient 178 attributs étiquetés, Chaque attribut de cette base contient 13 composantes, à savoir :

- $x_1$ : Alcohol
- $x_2$ : Malic acid
- $x_3$ : Ash
- $x_4$ : Alcalinity of ash
- $x_5$ : Magnesium
- $x_6$ : Total phenols
- $x_7$ : Flavanoids
- $x_8$ : Nonflavanoid phenols
- $x_9$ : Proanthocyanins

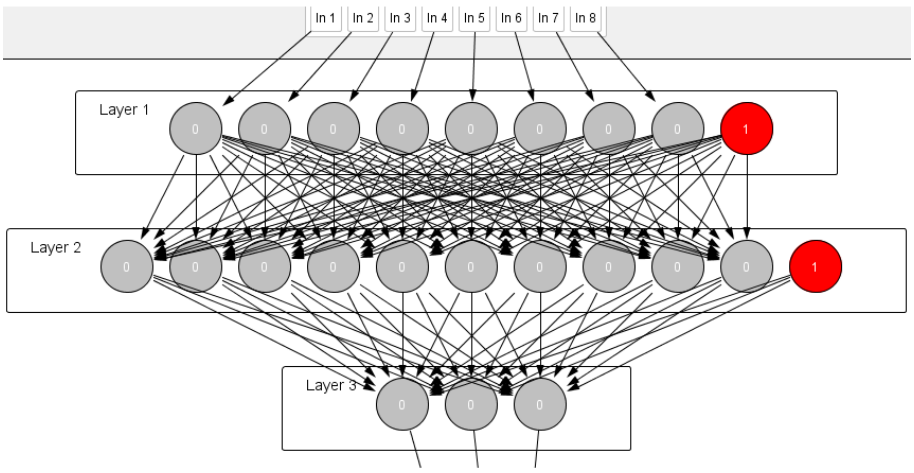



FIGURE 4.4 – Perceptron utilisé pour classifier la base Post-Operative Patient

- $x_{10}$ : Color intensity
- $x_{11}$ : Hue
- $x_{12}$ : OD280/OD315 of diluted wines
- $x_{13}$ : Proline

Elle contient aussi trois classes de vins $c_1$, $c_2$, et $c_3$. Ces classes seront codées de la même manière que la base Iris. Donc le PMC utilisé pour classifier cette base est (figure (4.5) :

— Une couche d'entrée contient 14 neurones.
— Une couche cachée de k neurones.
— une couche de sortie de 3 neurones.

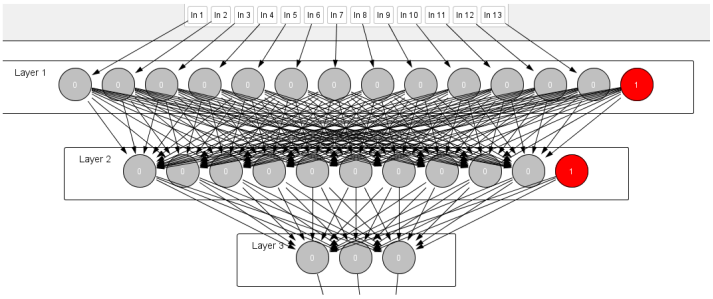



FIGURE 4.5 – Perceptron utilisé pour classifier la base Wine

**Remarques** Notons qu'on va faire la classification des 3 bases de données représentés ci-dessus. Après l'étape d'analyse de données et de calcule de la matrice de co-ocurrence associée à cette matrice, et après l'étape d'extraction des caractéristiques pertinentes de la matrice de co-ocurrence avec une méthode convenable (ACP par exemple), il reste que le codage de ces vecteurs caractéristiques pour qu'elle soient prêts à entrainer notre réseau neuronale. Les étapes avant apprentissage sont résumées dans la figure (4.6) :

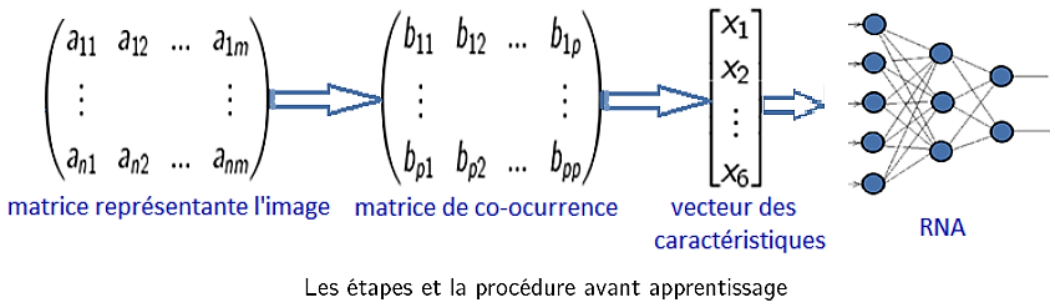



FIGURE 4.6 – Les étapes nécessaires avant l'apprentissage

### 4.1.5 Phase d'apprentissage

Afin d'évaluer la performance du PMC en tant que classifieur supervisé nous nous proposons d'étudier le taux de classification et le taux d'apprentissage dûs par le PMC. Ces deux critères de performance nécessitent deux ensembles de donnés du même base : une partie pour faire l'apprentissage du PMC, et une autre pour faire les test pour mesurer le taux de classification. Le tableau (4.2) résume la partitions des bases de données : Après avoir coder les vecteurs des caractéristiques

| **La base** | Le nombre d'attributs utilisés pour faire l'apprentissage | Le nombre d'attributs utilisés pour faire le test |
|---|---|---|
| **Iris** | 75 | 75 |
| **La base Post-Operative Patient** | 50 | 40 |
| **Wine** | 100 | 78 |

TABLE 4.2 – Description des ensembles d'apprentissage et de test utilisés

pertinentes données par chaque base, on utilise ces derniers comme entrées de notre réseau, puis on fait quelques études concernant l'architecture du PMC et la vitesse de convergence de l'algorithme de rétro-propagation du gradient.

#### Étude de l'effet d'architecture du réseau sur le temps d'apprentissage la vitesse de convergence

Dans cette application, pour une base de données fixées "Iris" on va étudier l'effet du changement de nombre de neurones de la couche cachée sur l'apprentissage, plus précisément, sur le temps d'apprentissage d'une part, et sur la vitesse de convergence de l'algorithme de rétro-propagation du gradient durant l'entrainement du réseau, Pour cela, on fait varier le nombre de neurones $k$ de la couche cachée de notre PMC dans un premier temps, en fixant le pas d'apprentissage $\alpha$ et l'erreur $\epsilon$ à ne pas dépasser. Puis, en varie les valeurs fixées "$\alpha$" dans un deuxième temps.
Les tableaux suivantes résument les résultats :

**Cas 1 :** Dans ce cas, on fixe les paramètres : pas d'apprentissage "$\alpha$" et l'erreur souhaité "$\epsilon$" aux valeurs suivantes :

— $\alpha = 1$,

— $\epsilon = 0.1$,

Les résultats obtenus dans ce cas pour plusieurs valeurs de $k$" nombre de neurones de la couche cachée" figurent dans le tableau (4.3) :

Les figure (4.7), (4.8) et (4.9) décrivent l'évolution du l'erreur généré par le procédure d'apprentissage

| **Nbr de neurones de la couche cachée** | k=10 | k=20 | k=30 | k=40 | k=50 | k=60 |
|---|---|---|---|---|---|---|
| **Le temps d'apprentissage** | 0.0119 | 0.0159 | 0.0358 | 0.1443 | 1.5497 | 6.2637 |
| **Nbr d'itérations nécessaires pour atteindre** $\epsilon$ | 22 | 51 | 107 | 455 | 4218 | 14406 |

TABLE 4.3 – L'effet de la variation du nombre de neurones de la couche cachée sur le temps d'apprentissage et le nombre d'itérations nécessaire pour pour converger avec une erreur $\epsilon$ dans le cas $\alpha = 1$

en fonction du nombre d'itérations nécessaires pour pour converger avec une erreur $\epsilon = 0.1$ pour un pas $\alpha = 1$ dans le cas $k = 20$, $k = 40$ et $k = 60$

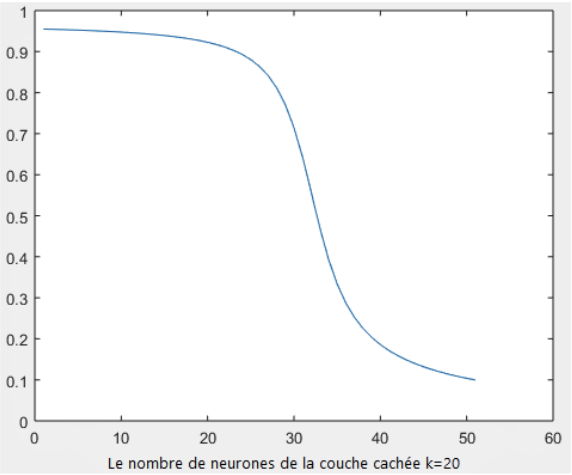



FIGURE 4.7 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ k=20.

**Remarques** D'après ces résultats, on constate que :

— Pour un nombre de neurone $k \leq 30$, le temps d'apprentissage reste raisonnable .

— Quand $k$ dépasse 30, le temps d'apprentissage augmente d'une manière très vite .

— Le nombre d'itérations nécessaires pour atteindre l'erreur $\epsilon$ augmente d'une manière explosive avec l'augmentation du nombre de neurones de la couche cachée.

**Cas 2 :** Dans ce cas, on va changer le paramètre : pas d'apprentissage "$\alpha$" et on va garder la même valeur de l'erreur "$\epsilon$" , c'est à dire :

— $\alpha = 0.5$,

— $\epsilon = 0.1$,

Les résultats obtenus dans ce cas pour les mêmes valeurs de $k$" prise dans le premier cas " se résument dans le tableau (4.4) :

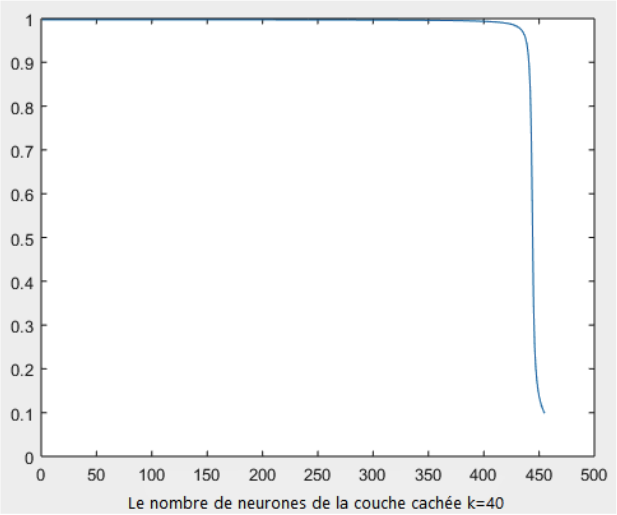



FIGURE 4.8 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ et k=40.

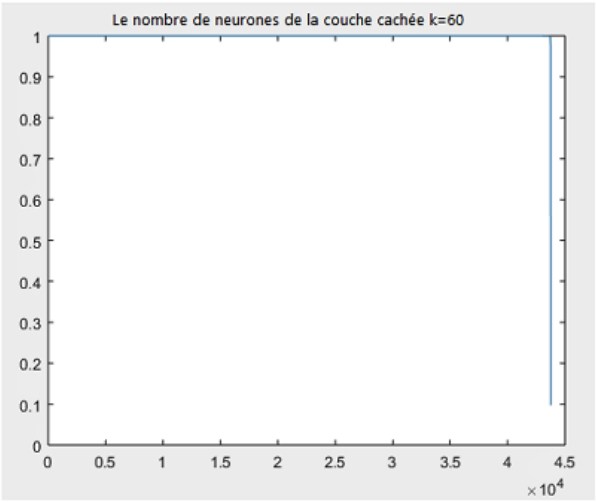



FIGURE 4.9 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 1$ et k=60.

| **Nbr de neurones de la couche cachée** | k=10 | k=20 | k=30 | k=40 | k=50 | k=60 |
|---|---|---|---|---|---|---|
| **Le temps d'apprentissage** | 0.0150 | 0.0205 | 0.1399 | 0.4971 | 1.0858 | 19.7572 |
| **Nbr d'itérations nécessaires pour atteindre** $\epsilon$ | 36 | 89 | 570 | 1627 | 2961 | 45942 |

TABLE 4.4 – L'effet de la variation du nombre de neurones de la couche cachée sur le temps d'apprentissage et le nombre d'itérations nécessaire pour atteindre l'erreur $\epsilon$ dans le cas $\alpha = 0.5$

**Remarques** D'après ces résultats, on remarque bien que :

— Pour un nombre de neurone $k \leq 30$, le temps d'apprentissage reste raisonnable mais supérieur à celui du cas précédent.

— Quand $k$ dépasse 30, le temps d'apprentissage augmente d'une manière très vite par rapport à celui du cas précédent.

— Le nombre d'itérations nécessaires pour atteindre $\epsilon$ est très grand par rapport au cas précédent. Les figures (4.10),(4.11) et (4.12) décrit l'évolution du l'erreur généré par le procédure d'apprentissage en fonction du nombre d'itérations nécessaires pour atteindre la valeur de l'erreur $\epsilon = 0.001$ pour un pas d'apprentissage $\alpha = 0.5$dans les cas $k = 20, k = 40$ et $k = 60$.

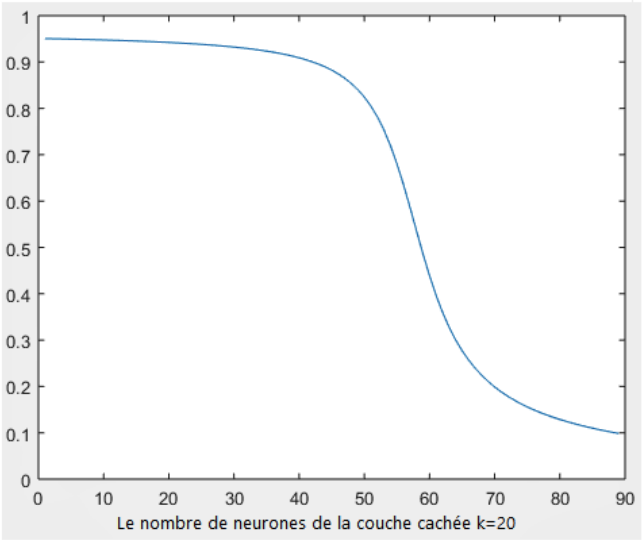



FIGURE 4.10 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 20$.

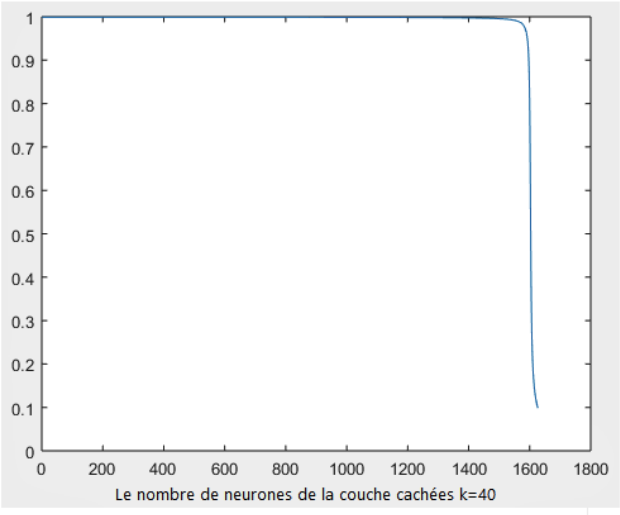



FIGURE 4.11 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 40$.

**Conclusion**

D'après les résultats si dessus, on peut affirmer que :

— Comme on est remarqué dans les deux cas, le nombre de neurones de la couche cachée influence beaucoup sur le temps d'apprentissage : plus que le nombre de neurone de la couche cachée augmente, plus que le temps d'apprentissage du réseau augmente aussi.

— De même, on peut dire que dans les deux cas, le nombre de neurones de la couche cachée influence beaucoup sur le nombre d'itérations nécessaires pour pour converger avec une erreur

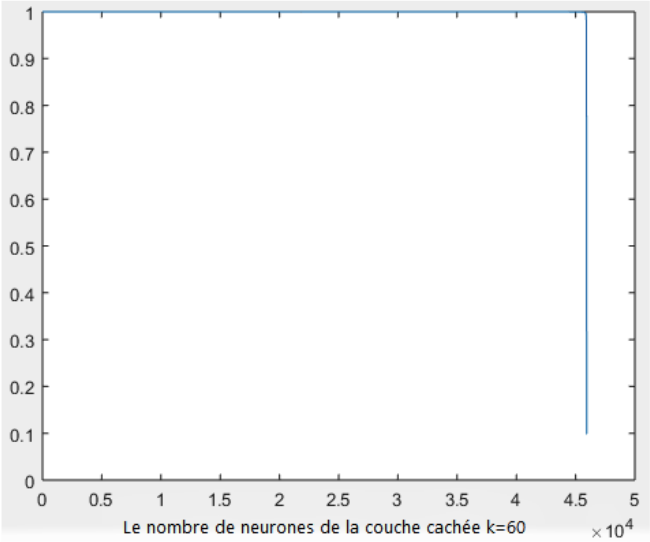



FIGURE 4.12 – L'évolution de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ pour les différentes valeurs de $k = 60$.

$\epsilon$ fixée : plus que le nombre de neurone de la couche cachée augmente, plus que ce nombre d'itérations augmente aussi.

— En comparant les temps d'apprentissage pour les différentes valeurs de $\alpha$, on affirme que le temps d'apprentissage devient plus grand si le pas d'apprentissage $\alpha$ diminue. De même pour le nombre d'itérations nécessaires.

**Étude de l'effet du pas d'apprentissage sur le temps d'apprentissage et la vitesse de convergence**

Maintenant, on va fixer les paramètres : k= nombre de neurones de la couche cachée en k=20, le nombre d'itérations maximales $I_{max} = 20$ ainsi que l'erreur $\epsilon = 0.001$, et on fait varier le pas d'apprentissage $\alpha$ selon plusieurs valeurs, afin d'étudier son effet sur la convergence de la méthodes de la rétro-propagation de gradient déjà proposé pour entrainer le PMC.
On résume les résultats obtenus dans le tableau (4.5) :

| $\alpha$ | 1 | 0.5 | 0.1 | 0.05 | 0.01 |
|---|---|---|---|---|---|
| **Le temps d'apprentissage** | 0.0496 | 0.0860 | 0.9826 | 1.5785 | 3.5487 |
| **Nbr d'itérations nécessaires pour atteindre** $\epsilon$ | 307 | 468 | 6327 | 10078 | 22606 |

TABLE 4.5 – L'effet du pas d'apprentissage sur le temps d'apprentissage et la vitesse de convergence.

**Remarque** L'effet du pas d'apprentissage est très clair d'après les résultats obtenus, en effet, on remarque bien que plus que le pas d'apprentissage diminue, plus que le temps d'apprentissage augmente, de même pour le nombre d'itérations nécessaires pour atteindre l'erreur souhaiter $\epsilon$.

**Conclusion**

D'après ces résultats, on peut affirmer que le pas d'apprentissage influence beaucoup sur la vitesse de convergence de la méthodes de rétro-propagation du gradient.
Les figures (4.13),(4.14) et (4.15) illustrent la variation de l'erreur quadratique moyenne en fonction de nombre d'itération pour trois valeur de $\alpha$ .

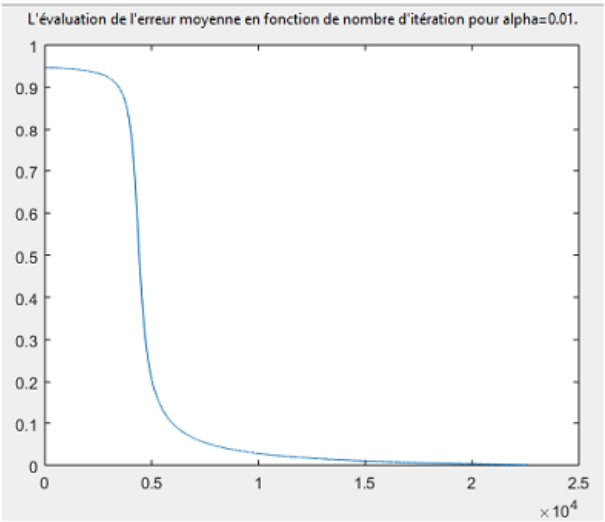



FIGURE 4.13 – L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.01$ et pour $k = 20$.

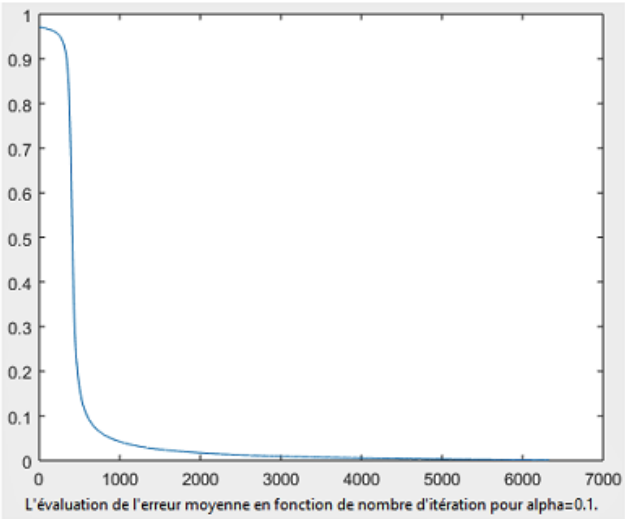



FIGURE 4.14 – L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.1$ et pour $k = 20$.

### Étude de performance du PMC en tant que classifieur supervisé

Afin d'étudier la performance du PMC dans la classification des données (taux de classification), on va entrainer le PMC pour faire classifier plus qu'une base de données, i.e, on faire une classification des trois différentes base de données déjà décrites par le PMC, et on va faire un test sur chacune de ces bases pour qu'on peut conclure à propos de la performance (l'erreur générée par le PMC dans la phase du test) du PMC comme classifieur supervisé.
Pour cela, on va utiliser la partie restante de l'ensemble de donnés pour faire ce test.
Pour un pas d'apprentissage $\alpha = 0.2$ et une erreur $\epsilon = 10^{-3}$ les résultats de ce test pour les différentes valeurs du $k$ se résument dans le tableau (4.6) :

On peut remarquer que le PMC est un bon classifieur supervisé.
La figure (4.16) illustre l'évolution de la performance du PMC en fonction du nombre de neurones de

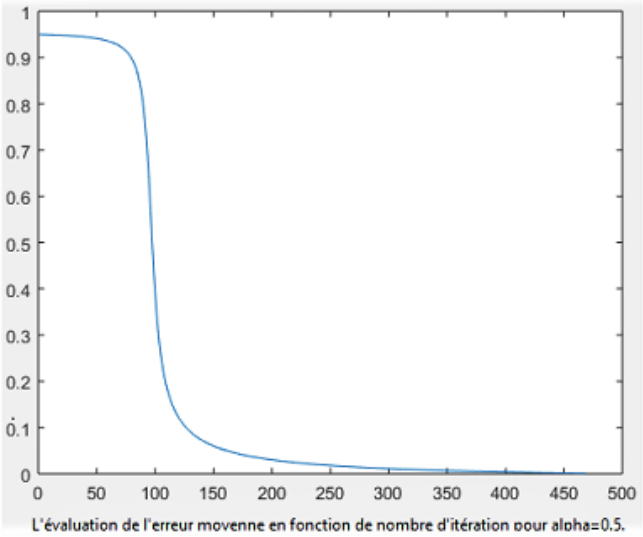



FIGURE 4.15 – L'évaluation de l'erreur moyenne en fonction de nombre d'itération pour $\alpha = 0.5$ et pour $k = 20$.

| **nombre de neurones de la couche cachée** | k=6 | k=12 | k=18 | k=24 |
|---|---|---|---|---|
| **Le taux de classification de la base Iris** | 0.99 | 0.96 | 0.98 | 0.98 |
| **Le taux de classification de la base Post-Operative Patien** | 0.73 | 0.87 | 0.78 | 0.81 |
| **Le taux de classification de la base La base Wine** | 0.91 | 0.97 | 0.96 | 0.98 |

TABLE 4.6 – Taux de classification des trois base déjà décrites.

la couche cachée effectuée sur la base Iris.

**Conclusion**

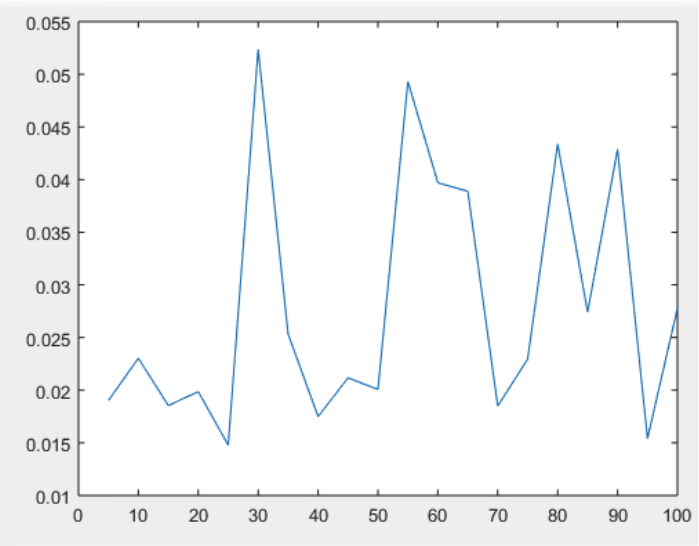



FIGURE 4.16 – L'évolution de l'erreur moyenne en fonction du nombre de neurones de la couche cachée.

D'après les résultats obtenus, on peut affirmer que le PMC est bien un classifieur supervisé bien adapté au problème de classification, concernant l'évolution de l'erreur moyenne en fonction du nombre de neurones de la couche cachée ci-dessus, on ne peut pas affirmer une règle bien correcte et générale concernant la relation existante entre l'architecture du PMC (nombre de neurones de la couche cachée) et le taux de classification et la performance du PMC.

### 4.1.6 Conclusion générale

Les réseaux de neurones artificiels est un outil puissant pour simuler des complexes problèmes, dans plusieurs domaines : économie, gestion, planification...
Plusieurs autres types de réseau neuronaux s'existent dans la littérature,ils se diffèrent selon l'architecture de ce réseau (le réseau de Hopfild, la machine de Boltzman, les réseaux bayésiens...)
Les études faites autour du PMC et son algorithme d'apprentissage (rétro-propagation du gradient) nous permet de conclure que ce dernier entraine bien un perceptron multicouches "PMC", en adaptant les paramètres de cet algorithme, en particulier le pas d'apprentissage $\alpha$ qui influencent beaucoup sur la convergence de cet algorithme d'apprentissage, ainsi que sa vitesse.
Nous pouvons également affirmer que le PMC est un classifieur supervisé non linéaire performant, bien adapté aux problèmes de types classification ou régression.
Des études d'amélioration du performance du PMC a été faites dans les dernières années au niveau de l'architecture du PMC (nombre optimal de couches cachées, nombre optimal des neurones de chacune des couches cachées...), elles permettent de modéliser ce problème en un problème d'optimisation non linéaire à variables mixtes.
Les perspectives de ce travail sont nombreuses : dans un premier temps, nous souhaitons améliorer l'algorithme de rétro-propagation du gradient et intégrer d'autres méthodes dans l'apprentissage des réseaux de neurones et faire une comparaison entre ces méthodes. Dans un deuxième temps, nous souhaitons améliorer la performance du PMC è travers l'optimisation de l'architecture de ce dernier tout en exploitant la théorie d'optimisation non linéaire en variables mixtes .

# Conclusion générale

Dans ce mémoire, on a traiter, dans un premier chapitre, le cadre théorique des problèmes d'optimisation non linéaire continu non contraint et contraint et des méthodes de résolutions de ces derniers. Puis, dans le deuxième chapitre, on a toucher la difficulté des problèmes d'optimisation non linéaire à variables mixtes et les conditions "méchantes" d'optimalités de ces derniers. Ensuite, dans le troisième et le quatrième chapitre qu'ils illustrent un cadre applicatif de la théorie d'optimisation dans la théorie d'apprentissage, en particulier, dans l'apprentissage du PMC par l'algorithme de rétropropagation du gradient et dans la classification des données par le PMC.
D'après les résultats obtenus dans le quatrième chapitre, on peut constater que l'algorithme de rétropropagation du gradient est plus adapté pour entrainer le PMC et pour la classification supervisée, reste à découvrir la relation existante entre l'architecture du PMC et la performance de ce dernier qui est un problème encore ouvert devant les chercheurs et que l'on va laisser pour le moment comme perspective de ce travail

# Bibliographie


[1] Ionel Sorin CIUPERCA, OPTIMISATION

[2] Frédéric de Gournay  Aude Rondepierre, Introduction à l'Optimisation Numérique.

[3] JAN A. SNYMAN, PRACTICAL MATHEMATICAL OPTIMIZATION, 2005. An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms.

[4] Stéphane Mottelet, Optimisation non-linéaire, Printemps 2003.

[5] Radosław Pytlak, Nonconvex Optimization and Its Applications, 2009, Conjugate Gradient Algorithms in Nonconvex Optimization.

[6] F. Gardi, K. Nouioua, Local search for mixed-integer nonlinear optimization : a methodology and an application, In Proceedings of EvoCOP,2011.

[7] S. Godskesen, T. Sejr, J. N. Kjeldsen, R. Larsen, Solving a real-life large-scale energy management problem, 2010.

[8] M. Porcheron, A. Gorge, O. Juan, T. Simovic, G. Dereu. 2010, a large-scale energy management problem with varied constraints EDF RD, Clamart, France February 2010.

[9] V. Le Diagon, Optimisation des grands arrêts de ranerie, Rapport de recherche ASSETSMAN www.assetsman.com

[10] I.F.P., L'IFP et le développement durable, http ://www.ifpenergiesnouvelles.fr, 2007.

[11] K. Rana, R.G. Vickson, Routing Container Ships Using Lagrangean Relaxation and Decomposition, 1991.

[12] R. Acuna-Agost, M. Boudia, S. Gabteni, N. Jozefowiez, C. Mancel, F. Mora- Camino. Méthodes de recherche opérationnelle pour la gestion des perturbations dans le domaine aérien, 2010.

[13] A.M.Geoffrion, Generalized Benders Decomposition, Journal of Optimization Theory and Applications 10, 1972.

[14] H.Marchand, A.Martin, R. Weismantel et L. Wolsey, Cutting Planes in Integer and Mixed Integer Programming, 2003.

[15] M. Minoux, Programmation mathématique : théorie et algorithmes,2007.

[16] Pierre Fouilhoux, Optimisation Combinatoire : Programmation Linéaire et Algorithmes, 2015.

[17] GISVOLD K.M., MOE J., A method for nonlinear mixed-integer programming and its application to design, ASME Journal of Engineering for Industry problems, 1972.

[18] FLETCHER R., POWEL M.J.D., A rapidly convergent descent method for minimization, Computer Journal, 1961.

[19] PAPPAS M., ALLENTUCH A., Mathematical programming procedures for mixed discrete-continuous design problems, ASME Journal of Engineering for Industry, 1974.

[20] CHA J.Z., MAYNE R.W., Optimisation with discrete variables via recursive quadratic programming : Part 1 - Concepts and definitions, ASME Journal of Mechanisms, Transmissions, and Automation in Design, 1989.

[21] CHA J.Z., MAYNE R.W., Optimisation with discrete variables via recursive quadratic programming : Part 2 - Algorithm and results, ASME Journal of Mechanisms, Transmissions, and Automation in Design, 1989.

[22] Nonlinear integer programming and discrete optimization, ASME Journal of Mechanisms, Transmissions, and Automation in Design, 1983.

[23] SANDGREN E., Nonlinear integer and discrete programming in mechanical design optimization, ASME Journal of Mechanical Design , 1990.

[24] Harry Zhang "The Optimality of Naive Bayes". Conférence FLAIRS2004.

[25] Caruana, R. and Niculescu-Mizil, A. : "An empirical comparison of supervised learning algorithms". Proceedings of the 23rd international conference on Machine learning, 2006.

[26] LeCun, Y., Bengio, Y., and Hinton, G. (2015), Deep learning. Nature, 521(7553) :436-444., 2015.

[27] He, K., Zhang, X., Ren, S., and Sun, J. (2016), Deep Residual Learning for Image Recognition, In Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[28] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems, 2012.

[29] Simonyan, K. and Zisserman, A. (2014), Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv preprint arXiv :1409.1556, 2014.

[30] Kingma, D. P. and Welling, M. (2013, )Auto-Encoding Variational Bayes, arXiv preprint arXiv :1312.6114, 2013.

[31] Courville, A., and Bengio, Y. (2014), Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Generative Adversarial Networks. In Advances in Neural Information Processing Systems.2014.

[32] V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012), Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, Deep Neural Networks for Acoustic Modeling in Speech Recognition : The Shared Views of Four Research Groups. IEEE Signal Processing Magazine, 29(6) :82–97,2012.

[33] Bordes, A., Weston, J., and Chopra, S. (2014), Question Answering with Subgraph Embeddings, arXiv preprint arXiv :1406.3676, 2014.

[34] Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., Teaching Machines to Read and Comprehend. and Blunsom, P. (2015), 2015.

[35] Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015), End-To-End Memory Networks., 2015.

[36] Sutskever, I., Vinyals, O., and Le, Q. V. (2014), Sequence to Sequence Learning with Neural Networks, Advances in Neural Information Processing Systems., 2014.

[37] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio,Y. (2015), Show, Attend and Tell : Neural Image Caption Generation with Visual Attention.2015.

[38] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G.,Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016), Mastering the game of Go with deep neural networks and tree search, Nature, 529, 2016.