MASTER EN SCIENCES INFORMATIQUES

Prédiction de séries temporelles à l'aide de cartes auto-organisées: application à la prédiction des courbes tarifaires de gaz

Simon, Geoffroy

Award date:
2002

FUNDP
Institut d'Informatique

Rue Grandgagnage, 21
B - 5000 NAMUR (Belgique)

Prédiction de séries temporelles à l'aide de cartes auto-organisées

Application à la prédiction des courbes tarifaires de gaz

Geoffroy SIMON

Co-promoteurs : Pierre-Yves SCHOBBENS
(FUNDP - Institut d'Informatique, Namur)

Michel VERLEYSEN
(UCL - FSA / DICE, Louvain-la-Neuve)

Mémoire présenté pour l'obtention
du grade de maître en informatique

Juin 2002

Remerciements

Les premiers remerciements sont destinés aux deux co-promoteurs de ce mémoire, Messieurs Michel Verleysen et Pierre-Yves Schobbens, pour leur encadrement lors de la découverte de ce domaine passionnant et leurs précieux conseils scientifiques.

Un merci tout particulier doit être adressé à Monsieur Amaury Lendasse, pour ses nombreux conseils scientifiques et sa patience tout au long de la réalisation de ce travail.

Merci à Madame et Messieurs Geneviève et Gaëtan Frippiat-Simon, Christophe Lorent et Jacques Thys, pour le prêt de leurs ordinateurs grâce auxquels une partie des simulations a été effectuée.

Enfin, merci à mon entourage et plus spécialement à Mademoiselle Samia Baccari, pour tout et tout le reste.

Errata

- section 2.3.3.2, page 23 : Au lieu de : 'Notons α et *distmax* sont deux paramètres [...]', il faut lire : 'Notons ***qu'α*** et *distmax* sont deux paramètres [...]'.
- section 2.3.7, page 27 : Au lieu de : 'Le lecteur [...] est donc invité à consulter cette annexe A.2.', il faut lire : 'Le lecteur [...] est donc invité à consulter cette annexe ***B***'.
- section 2.3.7, pages 29 et 30 : Dans la description des paramètres des différentes simulations, il est fait mention d'un taux et d'un voisinage. Au lieu de : 'taux' et 'voisinage', il faut lire : '***taux d'apprentissage***' et '***distance de voisinage***', respectivement.
- section 2.3.11, page 34 : Au lieu de : 'Rappelons pas que le but principal [...]', il faut lire : 'Rappelons que le but principal [...]'.
- section 3.6.2.1, page 45 : Au lieu de : '[...] ce qui nous fait passer des données « brutes » [...] aux données « non brutes » x_i', il faut lire : '[...] ce qui nous fait passer des données « brutes » [...] aux données « non brutes » x_t'. Plus bas, au lieu de : '[...] *un seul* ensemble contenant *toutes* les déformations y_i [...]', il faut lire : '[...] *un seul* ensemble contenant *toutes* les déformations y_t [...]'.
- section 3.6.3, page 47 : Au lieu de : 'x_i, $1 \leq i \leq k$ [...]', il faut lire : 'x_i, $1 \leq i \leq n_k$ [...]'.
- section 3.7.2, pages 48 et 49 : Au lieu de : '$x_i = (x_{i1}, x_{i2}, \ldots, x_{ik}, \ldots, x_{im})$', il faut lire : '$x_t = (x_{t1}, x_{t2}, \ldots, x_{tk}, \ldots, x_{tm})$'. La suite est modifiée en conséquent. Au lieu de : '[...] x_{ik} est une composante creuse de la donnée x_t. Lors de la prédiction, on additionne x_t et $\bar{y}_t$, et donc on effectue l'opération intermédiaire $x_{jk} + \bar{y}_{jk}$. Puisque la valeur de x_{jk} est indéterminée [...]', il faut lire : '[...] x_{tk} est une composante creuse de la donnée x_t. Lors de la prédiction, on additionne x_t et $\bar{y}_j$, et donc on effectue l'opération intermédiaire $x_{tk} + \bar{y}_{jk}$. Puisque la valeur de x_{tk} est indéterminée [...]'.
- section 3.7.3, page 49 : Au lieu de : 'Comme nous utilisons l'algorithme de Kohonen [...] des cartes auto-organisatrices ?', il faut lire : 'Comme nous utilisons l'algorithme de Kohonen [...] des ***cartes auto-organisées*** ?'.
- section 4.6.1.1, page 57 : Au lieu de : 'Le programme développé dans le cadre de ce mémoire [...] mais il est également défini par d'autres critères [...]', il faut lire : 'Le programme développé dans le cadre de ce mémoire [...] mais il y est également défini d'autres critères [...]'.
- section 5.4.1.1, page 63 : Au lieu de : 'Brièvement, un test χ^2 sur une table de contingence offre permet de tester [...]', il faut lire : 'Brièvement, un test χ^2 sur une table de contingence permet de tester [...]'.

- section 5.4.1.2, page 64 : Au lieu de : 'Les fréquences d'apparition [...] notées $P(\bar{y}_j \dim \bar{x}_i)$ [...] un tableau, qui constitue une table de contingence puisque $P(\bar{y}_j \dim \bar{x}_i) = n_{ij}$', il faut lire : 'Les fréquences d'apparition [...] notées $P(\bar{y}_j|\bar{x}_i)$ [...] un tableau, qui constitue une table de contingence puisque $P(\bar{y}_j|\bar{x}_i) = n_{ij}$'.
- section 5.4.6, page 74 : Au lieu de : 'Cependant, cette non-périodicité de la série est en fait perdue [...]', il faut lire : 'Cependant, cette périodicité de la série est en fait perdue [...]'.
- section 5.4.6, page 74 : Au lieu de : '$\hat{x}(t+1)$ n'est jamais qu'une estimation de la véritable valeur suivante, qu'est $\hat{x}(t+1)$', il faut lire : '$\hat{x}(t+1)$ n'est jamais qu'une estimation de la véritable valeur suivante, qu'est $\mathbf{x}(t+1)$'.
- section 5.4.6, page 75 : Au lieu de : 'on commet une erreur E_i qui vaut : $E_i = \sum_{k=1}^{i} e_i$', il faut lire : 'on commet une erreur E_i qui vaut : $E_i = \sum_{k=1}^{i} e_{\mathbf{k}}$'.
- section B.3.1, page 96 : Au lieu de : 'un voisinage', il faut lire : '***une distance de voisinage***'.
- section B.3.4.3, page 104 : Au lieu de : 'Observons le comportement d'une carte carrée et d'une ficelle. Dans le cas d'une carte carrée [...]', il faut lire : 'Observons le comportement d'une carte ***rectangulaire*** et d'une ficelle. Dans le cas d'une carte ***rectangulaire*** [...]'.
- Bibliographie, page 107 : Référence [16] : Au lieu de : '`http://www.auto.ucl.ac/~lendasse/pdf/ohio.ppt`', il faut lire : '`http://www.auto.ucl.ac.be/~lendasse/pdf/ohio.ppt`'.

Résumé

La prédiction à long terme de l'évolution d'une série temporelle est abordée dans ce travail. La méthode de prédiction se base sur les cartes auto-organisées de Kohonen. Cette méthode a été modifiée et généralisée pour tenir compte d'une série de problèmes rencontrés lors de la modélisation de l'évolution du prix du gaz.

Abstract

Long term predictions of times series are considered here. The prediction method is based on Kohonen' self-organising maps. This method has been modified and generalized to take into account specifical problems encountered when modeling the gaz' prices evolution.

Table des matières

1 Introduction **5**
- 1.1 Préambule . . . 5
- 1.2 Le cadre général de ce travail . . . 5
- 1.3 Données et séries temporelles . . . 6
  - 1.3.1 Définitions . . . 6
  - 1.3.2 Notations . . . 7
  - 1.3.3 Généralisation . . . 7
- 1.4 La prédiction de données temporelles . . . 8
  - 1.4.1 Modélisation des séries temporelles . . . 8
  - 1.4.2 Prédiction . . . 9
- 1.5 Les réseaux de neurones artificiels . . . 9
- 1.6 Par la suite 10

2 Réseaux de neurones artificiels et cartes de Kohonen **12**
- 2.1 Introduction . . . 12
- 2.2 Les réseaux de neurones artificiels : les origines de la discipline et les principaux modèles . . . 12
  - 2.2.1 Les origines : le neurone biologique . . . 12
  - 2.2.2 Les premiers modèles . . . 13
    - 2.2.2.1 L'article fondateur et le premier modèle . . . 13
    - 2.2.2.2 L'apprentissage . . . 14
    - 2.2.2.3 Le perceptron . . . 14
    - 2.2.2.4 L'importance des couches multiples de neurones . . . 14
  - 2.2.3 Les modèles multicouches . . . 15
    - 2.2.3.1 Résoudre la limitation du « linéairement séparable » . . . 15
    - 2.2.3.2 Eléments sur les réseaux multicouches . . . 16
  - 2.2.4 Les différents modèles : comment s'y retrouver ? . . . 17
  - 2.2.5 Point de vue informatique . . . 18
- 2.3 Les cartes auto-organisées de Kohonen . . . 18
  - 2.3.1 Introduction . . . 18
  - 2.3.2 La quantification vectorielle . . . 19
  - 2.3.3 La règle d'apprentissage . . . 21
    - 2.3.3.1 Notions de Competitive Learning . . . 21
    - 2.3.3.2 La règle d'apprentissage . . . 22
    - 2.3.3.3 Deuxième version . . . 23
  - 2.3.4 Algorithme complet . . . 24
  - 2.3.5 Propriétés des cartes de Kohonen . . . 24

2.3.5.1 La quantification vectorielle 24
2.3.5.2 Le respect de la topologie 24
2.3.6 Les paramètres du modèle 26
2.3.6.1 Le taux d'apprentissage 26
2.3.6.2 La distance de voisinage 26
2.3.6.3 Le nombre de neurones 26
2.3.6.4 La forme du réseau 27
2.3.7 Comportement du modèle 27
2.3.8 Le cas des centroïdes morts 30
2.3.9 Les données creuses 31
2.3.10 Résumé 33
2.3.11 Concrètement 34

3 Présentation de la méthode de prédiction de données temporelles 35
3.1 Introduction 35
3.2 Présentation générale 35
3.3 La méthode de prédiction des données temporelles 36
3.3.1 Note sur le calcul du régresseur 36
3.3.2 Partie caractérisation 37
3.3.2.1 Application du régresseur aux données 37
3.3.2.2 Première application de l'algorithme de Kohonen 37
3.3.2.3 Deuxième application de l'algorithme de Kohonen 38
3.3.3 Partie prédiction 39
3.4 Illustration de la méthode 40
3.5 Les paramètres de la méthode 44
3.5.1 Les paramètres de l'algorithme de Kohonen 44
3.5.2 Les paramètres spécifiques de la double quantification 44
3.6 Variante de la méthode 45
3.6.1 Note préalable 45
3.6.2 La méthode, seconde version 45
3.6.2.1 Partie caractérisation 45
3.6.2.2 Partie prédiction 46
3.6.3 Avantages de la méthode seconde version 46
3.7 Concrètement 47
3.7.1 Les données non scalaires 47
3.7.2 Les données creuses 48
3.7.3 Les centroïdes morts 49
3.7.4 Autres améliorations 50

4 Sélection de modèle et critères d'erreur 51
4.1 Introduction 51
4.2 L'erreur de prédiction 51
4.3 La sélection de modèle 52
4.3.1 L'importance des paramètres dans la modélisation 52
4.3.2 Le choix d'un modèle 53
4.4 La cross-validation 54
4.4.1 Notions de cross-validation 54
4.4.1.1 Idée intuitive 54

4.4.1.2 Définition et construction des ensembles d'apprentissage et de validation 54
4.5 Cross-validation et simulation Monte-Carlo 55
4.6 Concrètement 57
4.6.1 Définition des critères d'erreur 57
4.6.1.1 Première définition 57
4.6.1.2 Critique 58
4.6.1.3 Deuxième définition 59
4.6.1.4 Interprétation 60
4.6.2 Concrètement 60

5 Application de la méthode : la série Santa Fe A **61**
5.1 Introduction 61
5.2 Présentation de la série temporelle 61
5.3 Présentation générale des simulations 62
5.3.1 Le fichier utilisé 62
5.3.2 Les simulations 62
5.4 Simulations et résultats 63
5.4.1 Etude des données 63
5.4.1.1 Rappel théorique : le test χ^2 63
5.4.1.2 Application 64
5.4.2 Le cas des cartes carrées 65
5.4.2.1 La méthode « version DICE » 66
5.4.2.2 La méthode « version Cottrell » 66
5.4.2.3 Simulation supplémentaire 67
5.4.3 Le cas des cartes en ficelle 68
5.4.3.1 La méthode « version DICE » 68
5.4.3.2 La méthode « version Cottrell » 69
5.4.3.3 Simulation supplémentaire 70
5.4.4 Notes 71
5.4.5 Sélection de modèle et performances 71
5.4.6 Prédiction à long terme 73
5.5 Conclusion 75

6 Application de la méthode : la série Gaz de France **77**
6.1 Présentation générale 77
6.2 Application 77
6.2.1 Les données 77
6.2.2 Les différentes séries 77
6.2.2.1 Série brute 77
6.2.2.2 Série calée 78
6.2.2.3 Série des différences de la série brute 78
6.2.2.4 Série des différences de la série calée 78
6.2.2.5 Conséquences 78
6.2.3 Les simulations 79
6.2.3.1 Analyse des données 79
6.2.3.2 Sélection de modèle 79
6.2.3.3 Evolution à long terme 79

6.3 Note globale sur les résultats obtenus 79

7 Conclusion **81**

A Tableau des températures **83**

B Comportement des cartes de Kohonen **84**
B.1 Présentation des données 84
B.2 Importance des paramètres variables 85
B.2.1 Paramètres constants 85
B.2.2 Paramètres variables 89
B.2.3 Lois de décroissance 92
B.3 Importance relative de chacun des paramètres 95
B.3.1 Distance de voisinage 95
B.3.2 Taux d'apprentissage 97
B.3.3 Le nombre de neurones 98
B.3.4 La forme de la carte 101
B.3.4.1 Les cartes à une dimension 101
B.3.4.2 Les cartes à deux dimensions 102
B.3.4.3 Quelle carte utiliser ? 104
B.3.5 Note sur la valeur des paramètres 105

Bibliographie **107**

Chapitre 1

Introduction

1.1 Préambule

Comment prédire l'avenir ? Telle pourrait être la question de base de ce travail. Mais, bien entendu, savoir à l'avance ce que sera le futur n'a guère de sens ... Même en utilisant des 'techniques' !

Le présent travail est loin de répondre à une aussi ambitieuse question. Par contre, il permet de résoudre partiellement le problème suivant : quelle pourrait-être l'évolution future d'un certain phénomène fonction du temps ? Bien qu'il s'agisse encore d'un problème de prédiction, réduit à un seul processus, nous pouvons affirmer que ce problème est partiellement résolu grâce à une méthode qui va être présentée, développée et appliquée dans le reste de ce travail. En effet, la prétention n'est pas de savoir exactement à l'avance ce qui va se passer, mais plutôt d'essayer de dégager une tendance potentiellement réalisable dans un futur proche.

Comment réaliser cette prédiction ? Concrètement, nous utilisons une méthode purement technique qui manipule une série de mesures connues d'un processus évoluant au cours du temps et qu'on se propose de prédire. Cette manipulation permet de résumer l'information concernant l'évolution passée de manière locale, à savoir sur certains intervalles dans le temps. Par après, ce « résumé » est utilisé pour reproduire l'évolution du processus en se basant sur tous ces intervalles caractéristiques d'une tendance rencontrée dans le passé. Autrement dit, on essaye de découper la dynamique passée du processus et de reprendre ces petits morceaux dans un ordre utile pour simuler une évolution possible dans un avenir proche.

1.2 Le cadre général de ce travail

Prédire l'évolution d'un processus peut s'avérer, dans certaines situations, relativement important. Citons par exemple le cas de la consommation électrique. L'énergie électrique ne peut être stockée. Mais elle doit, dans un même temps, être produite en quantité suffisante afin de rencontrer la demande des clients d'un producteur. Celui-ci doit donc pouvoir anticiper l'importance de la demande de ses clients, dans le double but d'éviter :

- une surproduction d'énergie, qui serait perdue puisque non consommée et non stockable,
- une rupture de l'alimentation en énergie de ses clients.

Un autre domaine où il est relativement important de pouvoir prédire, ce sont les marchés boursiers. Pour les opérateurs, savoir s'il est préférable d'acheter ou de vendre, savoir s'il vaut mieux agir aujourd'hui ou attendre demain ou encore un autre jour, constituent des informations relativement critiques.

C'est dans le cadre de cette problématique des indices boursiers que se situe ce travail. Gaz de France achète sur les marchés londoniens une partie du gaz naturel qu'il fournit à ses clients, Londres constituant le principal lieu de cotation, en Europe, des matières premières énergétiques telles que le pétrole (baril de Brent) et le gaz naturel. Gaz de France s'intéresse plus particulièrement au prix du gaz sur les marchés à terme. De façon simplifiée, les marchés à terme fonctionnent de la manière suivante : un prix est fixé aujourd'hui pour une livraison d'une certaine quantité de marchandise à un terme fixé dans le temps lors de la signature du contrat de vente. Par exemple, un prix est fixé pour une vente aujourd'hui d'un produit qui sera livré à l'acheteur demain, ou dans 5 jours ouvrables, ou encore dans un mois, ou même l'an prochain. La date de livraison est une des clauses du contrat entre le vendeur et l'acheteur.

Gaz de France a commandé à l'équipe de Madame Marie Cottrell, du Centre de Recherche en Statistiques Appliquées et MOdélisation Stochastique (SAMOS) de l'Université Paris I, Panthéon-La Sorbonne, Paris, une étude de faisabilité portant sur la prédiction de l'évolution des courbes des prix du gaz sur les marchés à terme. Le SAMOS a pris contact avec Monsieur Michel Verleysen, du Laboratoire d'Electronique et de Microélectronique (DICE) de l'Université Catholique de Louvain, Louvain-La-Neuve, afin de prendre part à l'étude de faisabilité. Cette collaboration nous a permis de contribuer à cette étude de faisabilité, dans le cadre du stage de dernière année de la maîtrise en informatique des Facultés Universitaires Notre-Dame de la Paix, Namur. Le présent document décrit le travail qui a été effectué au cours de ce stage et constitue le mémoire de fin de cycle de la maîtrise en informatique.

1.3 Données et séries temporelles

Les cotations en bourse évoluent de jour en jour, en fonction de l'offre et de la demande relatives aux titres cotés. Cette évolution au jour le jour (parfois beaucoup plus rapide) peut être vue comme une évolution au cours du temps, en faisant abstraction de tous les autres mécanismes qui influencent typiquement les marchés boursiers : effet de rumeur, présentations des chiffres trimestriels d'une entreprise, événements politiques nationaux / internationaux, ...

1.3.1 Définitions

Soit un processus continu, évoluant au cours du temps. Par exemple, le cours d'une action.

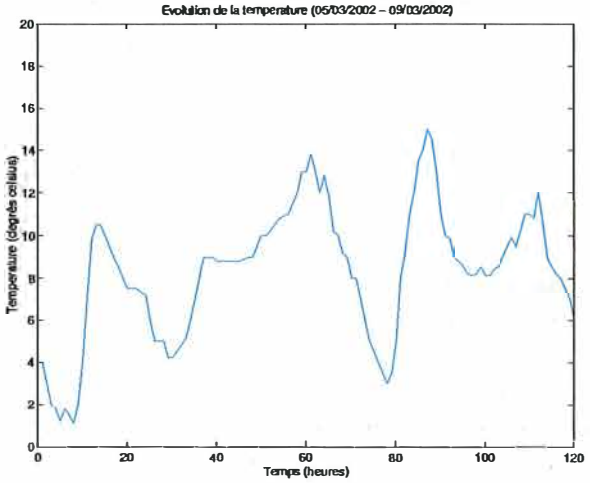
On appelle donnée temporelle une mesure de ce processus lors d'un instant t.

On appelle série de données temporelles, ou plus simplement série temporelle, un ensemble de données temporelles, classées selon l'ordre chronologique dans lequel elles ont été mesurées.

Notons que les mesures doivent être effectuées à intervalle fixe. Autrement dit, en général, créer une série temporelle revient à échantillonner un processus continu, selon une certaine fréquence.

Illustrons ces concepts sur base d'un exemple simple : le relevé de la température, processus qui évolue continuellement. Concrètement, la température a été mesurée entre le 5 mars 2002 à minuit et le 9 mars 2002, à 23 heures, à intervalle fixe d'une heure, dans la région gembloutoise. L'ensemble de ces 24 mesures sur 5 jours, soit un total de 120 données, sont reprises dans le tableau des températures en Annexe A, et constituent une série temporelle.

Pour chaque mesure, nous disposons de trois valeurs : le numéro de la mesure dans la série, l'heure à laquelle elle a été effectuée, et la température correspondante. Pratiquement, ne considérons plus le moment de la mesure mais plutôt son numéro, ou son indice, dans la série. Il est possible de dessiner un graphe de la température en fonction de cet indice représentant le temps :



Cette représentation permet de résumer en un clin d'œil la dynamique de l'évolution de la température au cours des 5 jours d'observation. Chaque relevé de la température constitue une donnée temporelle. L'ensemble des 120 mesures, ordonnées selon l'heure de la mesure qui varie de 1 (le 5 mars à minuit) à 120 (le 9 mars à 23h), constitue une série temporelle.

1.3.2 Notations

Par la suite, nous utiliserons les notations suivantes :

- x_t : une donnée temporelle, mesurée en t
- $x_1, x_2, x_3, \ldots, x_n$: une série temporelle, comprenant n données temporelles.

1.3.3 Généralisation

Bien entendu, les définitions et notations peuvent être généralisées. En effet, le processus étudié peut comporter plusieurs observations lors de chaque mesure.

Pour rester en météorologie, lors des mesures de températures effectuées pour l'exemple précédent, nous aurions également pu noter la direction du vent et le taux d'humidité de l'air. Nous aurions ainsi été en présence de données temporelles constituées de 3 mesures différentes.

Dans le cas général, une donnée temporelle est notée :

$$x_t = (x_{t1}, x_{t2}, x_{t3}, \ldots, x_{tm})$$

où m est le nombre de composantes observées lors de la mesure au temps t, aussi appelée dimension de la donnée temporelle ; la notation pour la série temporelle restant valable.

1.4 La prédiction de données temporelles

Intuitivement, pour réaliser la prédiction d'une série temporelle, il faut tout d'abord caractériser l'évolution connue de la série temporelle, pour essayer de comprendre le « fonctionnement » de la série sur base de son passé. Cette caractérisation est effectuée par une modélisation de la série, sur base des valeurs mesurées.

Une fois le modèle déterminé, on peut essayer de prédire l'évolution à venir. Notons que cette manière de faire se base sur l'hypothèse, relativement raisonnable, que le processus se comportera dans le futur de façon comparable à son évolution passée. Autrement dit, la dynamique de la série reste toujours semblable à elle-même.

1.4.1 Modélisation des séries temporelles

L'étude d'une série temporelle revient à modéliser l'évolution connue d'un processus continu dans le temps.

Plus formellement, le processus évolue selon sa propre dynamique, que nous notons $f(.)$. Par conséquent, lors de mesures effectuées à intervalles réguliers t, nous obtenons des valeurs x_t de la série temporelle sur base de la relation :

$$x_t = f(t).$$

Développer un modèle de ce processus revient à trouver une autre fonction du temps, $g(.)$, telle qu'en moyenne :

$$g(t) \cong f(t).$$

Malheureusement, puisque $g(.)$ n'est qu'un modèle de $f(.)$, l'égalité ci-dessus n'est pas stricte. En effet, quel que soit le niveau de complexité du modèle, il ne fait que reproduire 'aussi bien que possible' la véritable dynamique du processus. Dès lors, les valeurs données par le modèle $g(.)$ aux différents instants t sont :

$$\hat{x}_t = g(t).$$

Ces valeurs $\hat{x}_t$ constituent des approximations ou des estimations des vraies valeurs $x(t)$ données aux différents instants t par le processus observé $f(.)$. Ces estimations $\hat{x}_t$ des vraies valeurs x_t seront d'autant meilleures que le modèle $g(.)$ est proche de la véritable relation $f(.)$ qui caractérise la dynamique du processus. Par la suite, la notation $\hat{x}_t$ sera toujours utilisée pour décrire une estimation de x_t.

Comment peut-on dès lors obtenir ce modèle $g(.)$? Tout ce dont nous disposons, c'est de la série temporelle, qui constitue un échantillonnage de l'évolution passée, et donc connue, du processus observé. La valeur donnée par $g(t)$, estimation de la vraie valeur, sera donc une fonction des valeurs passées :

$$\hat{x}_t = g(x_1, x_2, x_3, \ldots, x_n, t)$$

où t est compris entre 1 et n, n étant le nombre total de mesures à notre disposition.

Cette formule est cependant un peu trop générale. Rien ne dit en effet que toutes les valeurs passées ont une influence sur la valeur actuelle. Le problème est alors de pouvoir déterminer quels sont les liens, s'il y en a, entre les éléments passés et la dynamique du processus, afin de déterminer le modèle $g(.)$ le plus proche possible de $f(.)$.

1.4.2 Prédiction

Une fois le modèle connu, la prédiction peut être abordée. Par prédiction, il faut entendre 'estimation de la (des) valeur(s) future(s)'. Notons que le plus important n'est pas de pouvoir déterminer exactement la (les) valeur(s) suivante(s), mais plutôt de déterminer la tendance à venir la plus probable. Il s'agit donc bien d'une prédiction, au sens large.

Si on considère $t = n + 1$, où n est le dernier instant où une mesure a été effectuée, alors la prédiction ne peut qu'être déduite du modèle $g(.)$, par la relation :

$$\hat{x}_t = g(x_1, x_2, x_3, \ldots, x_n, t).$$

Bien entendu, la prédiction peut être itérée pour connaître $\hat{x}_t$ en $t = n+2, t = n+3$, $\ldots$, $t = n + k$, et, par conséquent, obtenir la simulation de l'évolution à long terme. Il suffit pour cela, par exemple, d'inclure $\hat{x}_{n+1}$ au modèle et de demander une valeur pour $t = n + 2$; d'inclure $\hat{x}_{n+2}$ et de chercher $t = n + 3$; puis d'itérer jusqu'au terme final souhaité k, aussi appelé horizon de prédiction.

1.5 Les réseaux de neurones artificiels

Une question légitime à ce moment de l'introduction est de se demander quelle technique peut être utilisée pour modéliser une série temporelle.

Les outils généralement utilisés dans ce type d'étude sont des méthodes statistiques et / ou adaptatives. Dans le cadre de l'étude de faisabilitée demandée par Gaz de France, des méthodes adaptatives ont été envisagées : le SAMOS étudiait la série par des méthodes telles que les modèles auto-régressifs (AR), les modèles auto-régressifs à moyenne variable (ARMA), les modèles de Markov cachés, ..., alors que le DICE utilisait une autre méthode, basée sur une double quantification vectorielle par réseaux de neurones artificiels, de type cartes de Kohonen. Le lecteur désireux de mieux se familiariser avec les modèles AR, ARMA, ... lira avec intérêt [7].

Qu'entend-t-on par méthodes adaptatives ? De façon générale, il s'agit d'une famille d'algorithmes, appelés réseaux de neurones artificiels, qui peuvent être vus comme des méthodes non linéaires d'analyse de données.

Le domaine des réseaux de neurones artificiels, aussi appelé connexionisme, est une des nombreuses branches de l'intelligence artificielle. Mais il ne sera ici question d'utiliser les réseaux de neurones artificiels que pour ce qu'ils sont aujourd'hui : des outils algorithmiques. Le but de ce travail est loin de vouloir valider d'un point de vue biologique un modèle artificiel du mode de fonctionnement des neurones vivants, bien que le domaine trouve ses origines dans la biologie. Notre point de vue est d'utiliser une technique informatique particulière, que nous pensons particulièrement bien adaptée pour le problème posé par Gaz de France.

Les réseaux de neurones artificiels ont l'avantage d'une complexité raisonnable, et, une fois programmés, d'une utilisation relativement aisée vu qu'ils peuvent être considérés comme une boîte noire. C'est donc un outil, qui comporte une série de paramètres sur lesquels l'utilisateur peut avoir une influence. Cet outil est non linéaire. L'utilisation des paramètres est relativement délicate, dans la mesure où c'est elle qui permet de spécialiser le modèle très général du réseau de neurones au problème précisément étudié.

Y a-t-il une (des) raison(s) particulière(s) qui peut (peuvent) nous inciter à utiliser cet outil ? Outre la complexité raisonnable et la possibilité de modéliser des processus non linéaires, les réseaux de neurones artificiels ont les avantages suivants :

- capacité d' « apprentissage » par l'exemple. Les méthodes adaptatives peuvent « apprendre » des relations complexes sur simple présentation d'exemples décrivant ces relations.
- capacité de généralisation. Une fois l'apprentissage effectué, les réseaux de neurones artificiels peuvent étendre leurs « connaissances » à d'autres valeurs qui ne font pas partie de l'ensemble des exemples.
- théorème d'approximateur universel. Il a été démontré que les réseaux de neurones artificiels peuvent approximer d'aussi près que souhaité n'importe quelle relation.
- les réseaux de neurones artificiels sont des algorithmes. Ils sont donc aisément manipulables sur des ordinateurs. Par ailleurs, comme ils sont gourmands en temps de calcul, ils bénéficient des progrès réalisés au niveau de la puissance de calcul des processeurs, ce qui permet dans les faits de développer des modèles de plus en plus proches de processus complexes.

Ces avantages doivent être nuancés par les désavantages suivants :

- les réseaux de neurones artificiels ne constituent pas la solution miracle pour résoudre les problèmes jusqu'ici non résolus par les méthodes linéaires ou par les méthodes informatiques classiques. Il s'agit juste d'un ensemble de méthodes complémentaires, proposant une approche différente.
- le réglage des paramètres d'un réseau de neurones artificiels est une étape délicate, qui peut fortement pénaliser la qualité des résultats.
- le temps de calcul nécessaire lors de l'utilisation des réseaux de neurones artificiels est non négligeable et peut parfois constituer une limite à l'utilisation des méthodes adaptatives. Nous avons en effet fait l'expérience de simulations qui prennent plusieurs jours avant de donner des résultats.
- il existe relativement peu de résultats théoriques qui fondent la validité de la discipline. Les résultats obtenus, bien que de grande qualité, sont parfois expliqués par des données empiriques et non des preuves théoriques.

Notons que les réseaux de neurones artificiels ont déjà été appliqués, au DICE, à différents problèmes de prédiction, avec une certaine réussite. C'est donc fort de cette expérience acquise par le laboratoire que le choix s'est porté sur cet outil particulier.

Enfin, il faut savoir que la prédiction n'est pas le seul domaine d'application des réseaux de neurones artificiels, ceux-ci pouvant être appliqués en :

- approximation de fonction,
- reconnaissance automatisée, que ce soit la reconnaissance de caractères, de la parole ou encore en biométrie,
- compression de données,
- analyse de données en grandes dimensions, data-mining et extraction de caractéristiques
- séparation de sources sonores,

pour ne citer que quelques exemples. L'intérêt pour les réseaux de neurones artificiels commence seulement à se révéler et à s'étendre vers de nouveaux champs d'application, vu la jeunesse relative de la discipline.

1.6 Par la suite ...

Dans le chapitre 2, nous présenterons sommairement le domaine des réseaux de neurones artificiels. Nous étudierons ensuite plus précisément les cartes auto-organisatrices de

Kohonen, élément essentiel de la méthode de prédiction. Il est conseillé de parcourir, en parallèle de cette étude, l'annexe B portant sur le comportement des cartes de Kohonen.

Dans le chapitre 3, nous décrirons la méthode utilisée, ainsi qu'une variante de celle-ci. Certaines modifications apportées à la méthode seront également expliquées.

Dans le chapitre 4, nous expliquerons l'utilisation, dans le cadre de ce travail, d'une procédure classique pour sélectionner un « meilleur » modèle parmi tous les modèles possibles. Y sera également présentée la démarche qui nous a guidé dans le choix des critères utilisés pour évaluer l'erreur de prédiction, qui mesure l'écart entre le modèle $g(.)$ et la vraie fonction $f(.)$.

Dans le chapitre 5, nous appliquerons la méthode sur la série Santa Fe A, série bien connue en prédiction. Cette série nous a servi de test « grandeur nature », le but étant de valider le code de l'implémentation de la méthode avant de l'appliquer au problème de Gaz de France.

Dans le chapitre 6, nous introduirons l'application de la méthode aux données de Gaz de France. La présentation des résultats de la méthode, une analyse de ces résultats et quelques interprétations seront reprises dans l'annexe C, annexe confidentielle.

Le chapitre 7 conclura ce mémoire.

Chapitre 2

Réseaux de neurones artificiels et cartes de Kohonen

2.1 Introduction

Ce chapitre constitue une introduction brève et générale au domaine des réseaux de neurones artificiels. Il y sera rapidement présenté les origines de la discipline, son historique et les principaux modèles de réseaux de neurones artificiels. Le lecteur intéressé par le domaine pourra se référer aux ouvrages généraux [8], [5], [10] ; tous trois d'excellents ouvrages d'introduction ; ainsi qu'à l'incontournable [1], plus spécialisé et plus complet.

Parmi les différents modèles des réseaux de neurones artificiels, le modèle des cartes auto-organisées de Kohonen retiendra plus particulièrement notre attention. En effet, ce modèle est à la base de la méthode qui est développée dans le cadre de ce mémoire et est présentée au chapitre suivant. Une étude plus approfondie de ce type de réseaux de neurones sera donc bien utile, afin de comprendre en détail le fonctionnement de ce modèle, élément de base de la méthode de prédiction.

2.2 Les réseaux de neurones artificiels : les origines de la discipline et les principaux modèles

2.2.1 Les origines : le neurone biologique

L'idée des réseaux de neurones artificiels trouve son origine dans l'étude du cerveau humain. Le cerveau humain est en effet capable d'effectuer une multitude de tâches différentes : réflexion, raisonnement, parole, reconnaissance, abstraction, mémorisation, apprentissage, ... pour n'en citer que quelques-unes.

La question de fond était (et est toujours) de comprendre comment le cerveau fonctionne et sur quelle base il peut développer ces capacités peu communes. C'est en voulant comprendre le cerveau humain que l'intérêt des chercheurs s'est porté sur l'élément constitutif du cerveau, à savoir le neurone. Des modèles théoriques du fonctionnement du neurone biologique sont alors apparus. La discipline trouve donc ses fondements dans des observations et des études biologiques. Mais il est important de noter qu'aujourd'hui, les travaux effectués dans le domaine des réseaux de neurones artificiels n'ont plus grand-chose à voir avec la biologie, même si certains termes sont communs et même si des rapprochements entre les deux disciplines peuvent être effectués.

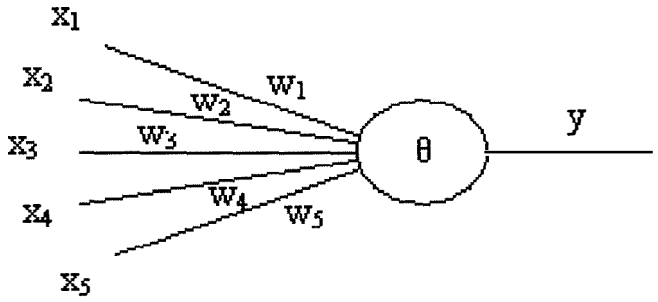
2.2.2 Les premiers modèles

2.2.2.1 L'article fondateur et le premier modèle

La première date importante dans le domaine des réseaux de neurones artificiels est 1943, date de parution de l'article de Mc Cullogh et Pitts qui présentait une étude des capacités des 'binary decision units'. Cet article est la première modélisation théorique du neurone naturel.

Dans cette étude, un modèle mathématique du mode de fonctionnement du neurone naturel était présenté. Ce modèle comportait une unité particulière, parfois appelée noyau, neurone ou encore tout simplement unité, qui effectue une somme pondérée des entrées pour fournir un signal en sortie. Développons quelque peu cette dernière phrase.

Soit le schéma simplifié d'un neurone :



où :

- il y a 5 entrées $x_i; 1 \leq i \leq 5$; qui correspondent aux dendrites d'un neurone naturel.
- il y a une unique sortie y, qui correspond à l'axone du neurone naturel, lui aussi unique.
- le noyau, ou neurone, est symbolisé par la figure ovale, comprenant la lettre θ.
- les valeurs $w_i; 1 \leq i \leq 5$; représentent ce qu'on appelle les coefficients synaptiques, qui correspondent à l'importance relative avec laquelle le noyau tient compte des signaux reçus sur les dendrites correspondants.

Sur ce schéma, le sens de propagation de l'influx nerveux va de gauche à droite. Au sein du noyau, une opération assez simple est effectuée. Il s'agit de la somme pondérée des entrées, déjà citée ci-dessus, et que nous pouvons écrire :

$$S = \sum_{i=1}^{5} w_i \, x_i$$

Cette somme S représente la somme des influx x_i reçus en entrée par le neurone, pondérés en fonction de l'importance relative de chaque connexion w_i. Le neurone va lui-même propager un influx si la somme S est supérieure à un certain seuil θ, appelé seuil d'activation. Dans ce cas, l'influx, ou encore le signal, est propagé le long de la sortie y.

Notons que pour Mc Cullogh et Pitts, les signaux tant en entrée qu'en sortie étaient binaires. Actuellement, certains modèles considèrent toujours des entrées et des sorties binaires, d'autres considèrent des entrées entières voire continues, voire des combinaisons entrées continues et sorties entières, etc.

2.2.2.2 L'apprentissage

Une des propriétés les plus intéressantes du neurone artificiel est sa capacité d'apprentissage. Expliquons brièvement en quoi consiste ce concept.

Si on présente un certain nombre d'exemples en entrée au neurone, avec en sortie la réponse attendue du neurone, celui-ci va adapter son comportement, suivant une règle d'apprentissage, afin de répondre au mieux à l'ensemble des données qui lui sont présentées en entrée.

Par « règle d'apprentissage », il faut comprendre qu'une loi générale de fonctionnement interne, le plus souvent une relation mathématique, va indiquer de quelle manière et avec quelle importance le neurone devra adapter son comportement.

Par « répondre au mieux », il faut entendre que le neurone va essayer d'adapter son comportement pour que la réponse qu'il propose pour une certaine entrée corresponde en moyenne à la réponse correspondante attendue en sortie; avec parfois une certaine erreur, idéalement minimale.

L' « adaptation du comportement » consiste en une modification des poids des entrées w_i ou, dit autrement, en un ajustement de l'importance relative associée à chacune des entrées : on augmente ou diminue le poids d'une (ou de plusieurs) entrée(s) pour que la sortie calculée se rapproche de la valeur attendue. Bien entendu, le neurone ne pourra pas donner une bonne réponse à chacune de ces données prises séparément, mais plutôt une réponse qui sera en moyenne satisfaisante.

Il est important de noter que l'apprentissage est en fait une capacité émergente des modèles de neurones artificiels, dans la mesure où rien dans ces modèles ne dit explicitement comment l'apprentissage se fait, en réalité. L'apprentissage résulte de l'adaptation du modèle à un ensemble de données, suivant une règle d'apprentissage qui est l'une des caractéristiques les plus importantes de chaque modèle. Cette règle permet au neurone de déterminer comment et dans quelle mesure adapter l'importance relative de chacune de ses entrées.

2.2.2.3 Le perceptron

Sur base du modèle décrit dans la section 2.2.2.1, Rosenblatt a développé en 1957 le modèle du perceptron, premier modèle dont les capacités d'apprentissage ont été étudiées.

Dans le cas du perceptron, le comportement du neurone est exactement le même que celui développé dans le modèle théorique de Mc Cullogh et Pitts : le neurone effectue également une somme pondérée de ses entrées.

La modification du comportement a lieu lorsque l'entrée, qui est passée au travers du neurone, produit une sortie différente de la sortie attendue. Il s'agit ici d'un type d'apprentissage particulier, appelé apprentissage supervisé, dans lequel les entrées et les sorties sont présentées en même temps au neurone. Par conséquent, lorsque le neurone a calculé sa sortie, il peut la comparer à la sortie désirée, ou attendue, correspondant à cette entrée. Si ces deux valeurs en sortie sont différentes, une modification du comportement a alors lieu.

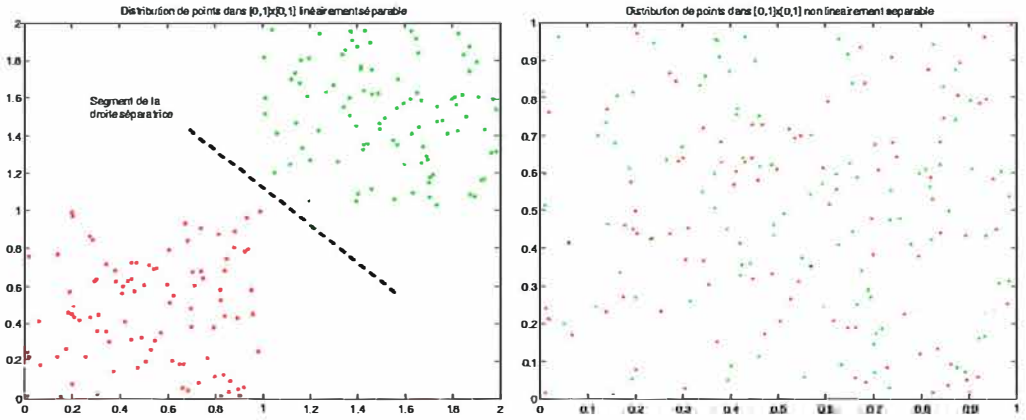
Mentionnons que d'autres modèles simples ont été développés à peu près à la même époque, tels que l'adaline et le madaline, qui sont des modèles linéaires.

2.2.2.4 L'importance des couches multiples de neurones

Le modèle du perceptron a eu un grand succès au cours des années 60, jusqu'à la parution en 1969 du livre 'Perceptrons', de Minsky et Papert, où il était démontré que le perceptron

ne pouvait en fait résoudre que des problèmes linéairement séparables.

Ce concept de séparabilité linéaire est à rapprocher du concept de convexité d'une région de l'espace. Plutôt que d'entrer dans les détails, observons deux exemples. Ces exemples comportent deux ensembles de points générés selon une loi aléatoire uniforme, et ont la propriété d'appartenir à une certaine classe parmi deux classes possibles. Une couleur est attachée aux points, afin de pouvoir déterminer s'ils appartiennent à la première ou à la deuxième classe (respectivement la rouge et la verte) :



On peut voir que les classes de points de la première région, à gauche, sont linéairement séparables : les points peuvent être classés en deux sous-régions séparées par une seule droite (un hyperplan dans le cas de données de dimension supérieure à 2). Cette droite est symbolisée, dans l'exemple ci-dessus, par un segment en pointillés. Dans le cas de la figure de droite, les points ne peuvent être séparés par une simple droite.

Notons que le cas de la fonction logique XOR n'est pas linéairement séparable. Un perceptron classique ayant par exemple deux entrées (les valeurs) et une sortie (la disjonction exclusive de ces valeurs) ne peut pas modéliser cette fonction, pourtant très simple.

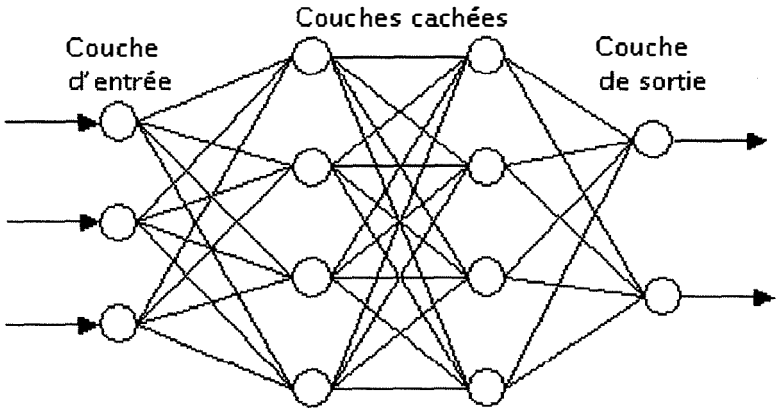
2.2.3 Les modèles multicouches

2.2.3.1 Résoudre la limitation du « linéairement séparable »

Pendant une petite quinzaine d'années, cette limitation a porté un coup sérieux à la recherche dans le domaine des réseaux de neurones artificiels. Le livre de Minsky et Papert a considérablement influencé les milieux politiques et militaires, notamment américains, qui ne voyaient dès lors plus l'utilité de financer des recherches sur le sujet.

Une des idées qui a permis de donner un nouvel essor au domaine a été de reprendre la sortie d'un neurone et d'en faire une entrée pour un (ou plusieurs) autre(s) neurone(s), créant ainsi une topologie en couches. Ces couches forment alors ce qu'on appelle un réseau de neurones artificiels. Cette nouvelle structuration des neurones a par ailleurs donné son nom au domaine.

Illustrons cette idée avec la figure suivante :



où :
- les ronds symbolisent les neurones,
- les segments de droite symbolisent les connexions entre les neurones,
- la première couche, à gauche, avec trois neurones, constitue la couche d'entrée,
- les deuxième et troisième couches, de quatre neurones chacune, sont ce qu'on appelle des couches cachées,
- la dernière couche, composée de deux neurones, est la couche de sortie.

Il faut remarquer que, dans cet exemple, les sorties d'un neurone sont reliées à chacun des neurones de la couche suivante. Cet exemple est donc appelé réseau multicouches complètement connecté. Il existe donc des modèles qui sont partiellement connectés, qui comportent des boucles, etc.

2.2.3.2 Eléments sur les réseaux multicouches

Bien entendu, ce modèle à plusieurs couches est plus complexe vu qu'un seul niveau, le dernier, peut être directement ajusté lors de l'apprentissage (c'est le seul niveau dont on connaît les valeurs attendues en sortie). Néanmoins, différentes techniques ont été développées pour obtenir des règles d'apprentissage pour toutes les couches du réseau, dont la plus connue est sans doute la rétro-propagation.

Plusieurs modèles multicouches ont alors été développés. Parmi tous ces modèles, les perceptrons multicouches (Multi-Layers Perceptrons ou MLP, dans la littérature) et les réseaux à fonction radiale de base (Radial Basis Function ou RBF, dans la littérature) sont les plus célèbres et aussi les plus utilisés en pratique.

Ces modèles ont été étudiés sur le plan théorique. Quelques propriétés et théorèmes ont été étudiés et démontrés dans ces deux grandes familles de réseaux. Le résultat le plus intéressant, au moins d'un point de vue théorique, est le théorème d'approximateur universel, qui dit que n'importe quelle fonction mathématique peut être approximée d'aussi près que souhaité par un réseau multicouches pour autant qu'il y ait un nombre suffisamment grand de couches de neurones et un nombre lui aussi suffisamment grand de neurones, à l'intérieur du modèle. Notons tout de même que ce théorème ne constitue qu'une preuve de l'existence d'un réseau de neurones artificiels approprié pour un problème donné, et non pas une méthode constructive pour obtenir le réseau de neurones artificiels en question ...

2.2.4 Les différents modèles : comment s'y retrouver ?

Bon nombre de modèles différents ont été développés depuis 1943, dont quelques-uns ont été très brièvement introduits ci-dessus.

Si, à la base, ils sont tous fort semblables, bien des différences peuvent être constatées entre les modèles. Bien qu'il ne soit ici question que d'une brève présentation générale du domaine, il peut être intéressant d'observer certains critères qui permettent de classer les différents modèles dans certaines catégories.

La première distinction qui peut être faite concerne les connexions à l'intérieur du réseau :

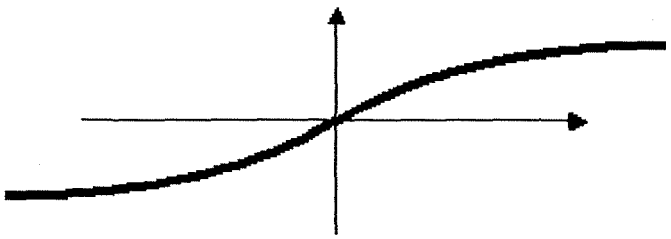
- les réseaux avec connexions de poids constant : le niveau d'activité des neurones est ajusté, sans changer les poids des connexions.
- les réseaux avec connexions de poids variable : les poids des connexions sont ajustés au cours d'une procédure d'apprentissage.

Le deuxième grande différence à observer concerne le type d'apprentissage. Jusqu'ici, seul l'apprentissage supervisé a été présenté. Mais il existe en fait trois catégories d'apprentissage :

- apprentissage supervisé : les entrées et les sorties sont présentées en même temps au réseau sous forme de couples, les sorties servant à la mise à jour des poids des connexions après observation de la différence ou non entre les sorties calculées par le réseau et les sorties attendues.
- apprentissage non supervisé : les entrées seules sont présentées au réseau. Il est évident que, dans ce cas-ci, la mise à jour des poids des connexions ne peut se baser sur des critères d'erreur entre sorties calculées et désirées. D'autres critères sont alors utilisés, généralement des relations de proximité définies au moyen d'une mesure de distance.
- apprentissage renforcé : le réseau est récompensé par son environnement en fonction de la réponse qu'il calcule pour une entrée ; la récompense pouvant être positive ou négative. Le principe est ici d'avoir un maximum de récompenses positives pour un minimum de récompenses négatives sur l'ensemble des données.

Un autre élément très important est la règle d'apprentissage qui est utilisée par le modèle. Beaucoup de règles différentes ont été étudiées, et elles ne seront pas décrites ici, afin de ne pas se limiter à une longue présentation d'équations sans leur donner un minimum d'explications.

L'élément le plus important pour chacune de ces règles d'apprentissage est le type de critère retenu pour la fonction d'activation. Ce critère est une fonction mathématique qui recouvre le concept de seuil d'activation présenté dans le modèle du neurone biologique. Là aussi, beaucoup de fonctions ont été utilisées, dans différents modèles. La plupart du temps, on utilise des fonctions continues, pour pouvoir profiter de toutes les propriétés liées à la continuité (caractère lisse, différentiabilité, ...). Généralement, on utilise des fonctions de type tangente hyperbolique ou sigmoïde, dont le graphe a pour forme générale :



Enfin, signalons que chaque modèle comporte une série de paramètres qui lui sont propres et qui permettent à l'utilisateur de ce modèle de jouir d'une certaine liberté dans la modélisation de son problème par un réseau de neurones artificiels. Citons notamment le nombre de couches de neurones dans le cas des réseaux multicouches ; le nombre de neurones, que ce soit par couche ou pour l'ensemble du réseau ; la règle d'apprentissage à utiliser ; le critère de mesure de l'erreur en sortie ; la fonction d'activation utilisée ; ...

2.2.5 Point de vue informatique

Par le passé, les réseaux de neurones artificiels étaient concrétisés sous la forme de puces électroniques. Mais les coûts de développement d'une puce et le temps nécessaire à sa réalisation sont non négligeables.

L'utilisation des ordinateurs classiques a permis de dépasser ces contraintes, et de développer non plus des puces mais des algorithmes qui simulent sur machine le fonctionnement de la puce. Les réseaux de neurones artificiels sont donc aujourd'hui, dans la plupart des cas, des programmes qui se comportent lors de leur exécution comme se serait comportée la puce.

En termes de programmation, un réseau de neurones n'est pas défini explicitement dans un programme. Ce qui est programmé, c'est un algorithme général qui permet de créer des réseaux de neurones d'un certain type et d'appliquer la règle d'apprentissage correspondante à ce type de réseaux. Cette règle particularisera le réseau pour qu'il soit adapté à l'ensemble des points utilisés lors de l'apprentissage.

2.3 Les cartes auto-organisées de Kohonen

Nous introduisons ici un nouveau modèle de réseaux de neurones artificiels dont il n'a pas encore été question, à savoir les cartes auto-organisées. Ce modèle est à la base de la méthode de prédiction qui est au centre du travail accompli dans le cadre de ce mémoire.

2.3.1 Introduction

Le modèle des cartes auto-organisées est un modèle bien différent des modèles cités jusqu'ici.

L'idée de base du modèle fait ici encore référence à la biologie. En effet, différentes études neuro-physiologiques ont permis de mettre en évidence le fait que tous les inputs reçus par des nerfs ayant des extrémités sensorielles dans deux membres ou régions du corps proches l'un de l'autre sont reçus, au bout du cheminement neuronal à travers l'organisme, dans

deux zones proches du cortex. Il y a donc, au niveau du cortex cérébral, un ordonnancement des signaux reçus selon une topologie proche de la réalité physique du corps.

Le premier développement théorique de cette idée a été effectuée par C. Von der Malburg, en 1973 [13], qui a proposé l'idée d'une activation semblable pour des neurones proches topologiquement, en réponse à des signaux en entrée proches en amplitude, en intensité, ... Bien entendu, cette activation de différents neurones proches n'a de sens que dans les limites d'une certaine distance maximale au-delà de laquelle les neurones n'ont plus d'influence les uns sur les autres. Cette même idée a été reprise, simplifiée puis développée par Teuvo Kohonen [11]. Depuis, les cartes auto-organisées sont également appelées cartes de Kohonen.

Une caractéristique importante de ce modèle est le type d'apprentissage utilisé, qui est dans ce cas non supervisé. Ceci revient à dire que les seules entrées sont présentées au réseau de neurones artificiels. L'adaptation du réseau ne peut donc être basée sur un critère d'erreur entre une sortie calculée et une sortie attendue. Cette adaptation est en fait gérée par le réseau lui-même, d'où ce nom de cartes auto-organisées.

Dans un premier temps, nous introduirons le concept de quantification vectorielle. Ensuite, nous allons étudier le modèle de Kohonen, décrire la règle d'apprentissage permettant cette auto-adaptation et analyser les propriétés des cartes de Kohonen. Nous observerons alors le comportement de ces cartes auto-organisées. Enfin, nous présenterons les améliorations apportées à cet algorithme.

2.3.2 La quantification vectorielle

Considérons un problème de la vie réelle dont on veut connaître la dynamique. Par exemple, le problème de l'évolution d'un cours de bourse, dont on souhaite comprendre l'évolution au jour le jour. Pour ce faire, on décide de « mesurer » le problème considéré, où une mesure est en fait une observation effectuée à un instant précis. On compare alors les mesures et on espère pouvoir trouver des similitudes entre certains évènements. Ces similitudes peuvent, en quelque sorte, nous renseigner sur la dynamique de l'évolution du problème, par exemple le cours de bourse étudié.

Les mesures effectuées sont récoltées dans une base de données, qui contient par exemple les cours d'un certain indice, tous les jours, à la fermeture. Ces mesures se rapportent chacune à un aspect du problème. Supposons maintenant que nous soyons non plus intéressés par un cours de bourse, mais par un ensemble de valeurs. Il peut alors être fait une série de mesures qui seront elles aussi reprises dans une base de données. Lorsque ce problème a fait l'objet d'un grand nombre de mesures, et que chacun des cours à étudier a lui-même été mesuré un grand nombre de fois, on se retrouve vite avec un nombre important de données. Malheureusement, l'information contenue au sein de ces données peut être relativement dispersée. Elle peut également vite devenir très importante, si on prend beaucoup de mesures. Il arrive même qu'elle soit trop volumineuse pour être appréhendée efficacement.

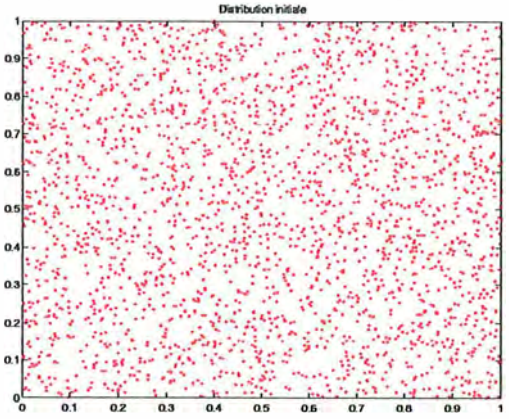
Une méthode classique pour réduire toute cette information est la quantification vectorielle. Le but de la quantification vectorielle est de diminuer la taille d'un ensemble de données tout en essayant de conserver la même information que celle contenue dans l'ensemble de départ. Dit autrement, il s'agit d'une *compression du volume* des données, en concédant un *minimum de perte* d'information.

Par exemple, considérons le temps. Considérons par ailleurs une horloge ou une montre. La montre réalise une quantification vectorielle du temps. En effet, pour tout un chacun, connaître avec exactitude la valeur d'un instant précis dans le temps requiert beaucoup de prudence dans la définition de la mesure utilisée, du point initial considéré, ... Par ailleurs, le

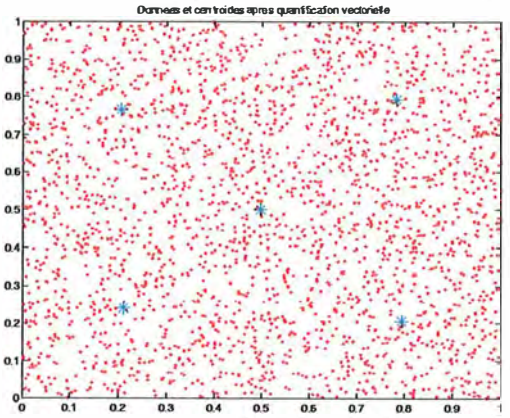
temps passe tellement vite que décrire chaque instant avec une grande précision engendrerait immédiatement un volume de données très important et très difficilement manipulable par l'esprit humain. C'est pourquoi, vu l'échelle de temps communément utilisée, la montre est généralement suffisante pour décrire la valeur actuelle du temps qui passe, bien qu'elle ne donne jamais qu'une information réduite par rapport à la valeur exacte de chaque instant. Par conséquent, la montre réalise bel et bien une réduction du volume de l'information, tout en limitant la perte d'information due à cette réduction.

Au niveau des résultats obtenus, la quantification vectorielle génère un ensemble réduit de données, représentatives des données de départ. A chacune de ces données représentatives est associée une région de l'espace de départ dans laquelle sont rassemblées toutes les données qui sont projetées sur cette donnée représentative, également appelé le centroïde de la région considérée. Généralement, ces régions sont appelées des zones ou des régions de Voronoï, du nom du mathématicien russe Georgy Voronoï qui, le premier, a étudié les propriétés géométriques de ces régions.

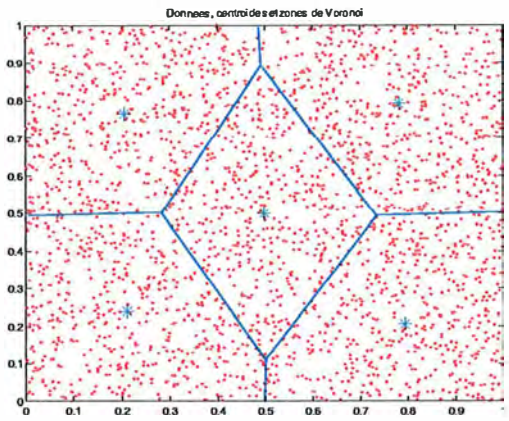
Illustrons maintenant ces concepts liés à la quantification vectorielle. Soit une région de $\mathbb{R}^2$ contenant 2500 points uniformément distribués :



Une quantification vectorielle de ces points pourrait être, si on considère par exemple cinq vecteurs représentatifs, ou centroïdes :



On peut alors dessiner cinq régions respectivement associées à un des vecteurs représentatifs :



Nous voyons clairement sur ce dernier schéma les cinq régions de Voronoï résultant de la quantification vectorielle avec, dans chaque région, le centroïde associé.

Notons que d'un point de vue théorique, un quantificateur vectoriel est un couple $(m,\ q)$ où :

- m est l'ensemble des N centroïdes, N étant choisi *a priori* :

$$m = \{y_k; 1 \leq k \leq N\},$$

- q est la fonction de quantification qui projète les points x_{i_k} de la région k sur le centroïde y_k associé à cette région :

$$q(.) \equiv x_{i_k} \rightarrow y_k,$$

où $1 \leq i_k \leq M_k$, M_k étant le nombre de points x_i de l'ensemble de départ compris dans la zone de Voronoï de centroïde y_k.

Bien évidemment, on a :

- $i_k \in I_k$, l'ensemble des indices des points de la k$^{\text{ième}}$ zone de Voronoï, avec $\bigcup_{k=1}^{N} I_k = I$, I étant l'ensemble des indices de la base de données.
- $\sum_{k=1}^{N} M_k = M$, où M est le nombre total de points quantifiés par les N centroïdes, $N < M$.

Dans la plupart des cas, la fonction q est une règle de type « plus proche centroïde » selon une mesure de distance, souvent euclidienne. C'est notamment le cas dans l'exemple ci-dessus.

2.3.3 La règle d'apprentissage

2.3.3.1 Notions de Competitive Learning

Puisque l'apprentissage est ici non supervisé, il ne peut se baser sur un critère d'erreur entre une sortie calculée par le réseau et une sortie souhaitée. L'apprentissage est donc tout autre et doit être mis en rapport avec le « Competitive Learning », algorithme simple de la famille des règles d'apprentissage non supervisé.

Brièvement, en « Competitive Learning », lorsqu'une donnée est présentée au réseau, une comparaison sous forme de compétition, comme l'indique le nom, commence entre toutes les sommes pondérées des différents neurones. Seul un neurone sort vainqueur de cette compétition, à savoir celui dont la somme pondérée est la plus élevée. C'est ce neurone qui sera activé pour la donnée correspondante en entrée. Lui seul verra donc son comportement adapté lors de cette étape de l'algorithme. Bien entendu, il faut ensuite présenter au réseau les autres données de l'ensemble pour compléter l'apprentissage en cours ; d'autres neurones l'emportant pour d'autres données en entrée, faisant ainsi évoluer le comportement global du réseau.

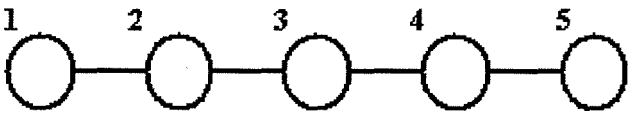
Enfin, il faut savoir que cet algorithme de « Competitive Learning » permet de quantifier vectoriellement un ensemble de points, les neurones jouant le rôle des centroïdes.

2.3.3.2 La règle d'apprentissage

L'algorithme de Kohonen utilise ce concept de « Competitive Learning », avec mise à jour du neurone vainqueur, mais aussi avec modification *d'autres neurones*, voisins du vainqueur et situés à une distance inférieure à une certaine limite supérieure. Cet algorithme comporte donc lui aussi deux étapes : le choix du vainqueur et la mise à jour du vainqueur *et de ses voisins.*

Cette notion de voisinage recouvre la relation physique des neurones entre eux, au niveau de la carte. Cette relation nous indique qu'il existe une certaine importance des positions respectives des neurones les uns par rapport aux autres.

Par exemple, supposons que les neurones du modèle peuvent être représentés selon une structure géométrique, que nous choisissons comme étant une ligne. L'importance de la position des neurones les uns par rapport aux autres peut être assimilée à un ordre, ou une numérotation :



Sur le schéma ci-dessus, les neurones sont symbolisés par des ronds, les liens entre neurones étant la concrétisation de la relation physique « est voisin de ». Considérons le neurone central, numéroté 3. On définit les neurones situés à une distance de voisinage de 1 du neurone 3 les neurones 2 et 4. De même, les neurones situés à une distance de deux sont les neurones 1 et 5. On dit également que le neurone 1 est à une distance de voisinage de 4 du neurone 5. Cette notion de voisinage peut être généralisée à une structuration quelconque de neurones, et non plus uniquement au cas d'une topoplogie en ligne. Intuitivement, la notion de distance de voisinage peut être mise en relation avec la longueur du plus court chemin reliant deux neurones entre eux, le chemin étant composé exclusivement de liens physiques entre neurones.

Nous pouvons maintenant écrire les équations à la base de l'algorithme de Kohonen :

- choix du vainqueur :

$$y_k = min_i(dist(w_i, x))$$

où

- x est la donnée présentée au réseau, de dimension n,

- y_k désigne le neurone vainqueur au niveau de la carte,
- w_i, également de dimension n, représente les poids des n connexions différentes, en entrée, propres au neurone y_i, $1 \leq i \leq N$, où N est le nombre de neurones,
- $dist(w_i,\ x)$ est la distance utilisée qui est, dans notre cas, la distance euclidienne.

Intuitivement, le poids w_i désigne la position du neurone y_i à l'intérieur de la distribution des points x de l'ensemble d'apprentissage, tandis que y_i désigne le neurone au niveau de la topologie de la carte.

- mise à jour des poids :

$$\begin{cases} w_i(t+1) = w_i(t) + \alpha(x - w_i(t)) & si\ d(y_k,\ y_i) \leq distmax \\ w_i(t+1) = w_i(t) & si\ d(y_k,\ y_i) > distmax \end{cases}$$

où

- t représente l'itération en cours,
- α est le taux d'apprentissage,
- d(., .) est la distance entre neurones *sur la carte*,
- $distmax$ est la distance maximale de voisinage.

Notons α et $distmax$ sont deux paramètres dont la valeur doit être fixée.

Les autres notations sont les mêmes que celles utilisées ci-dessus.

2.3.3.3 Deuxième version

Comme le montre l'annexe B.2.1, la convergence de la carte, qui peut être définie comme la manière dont la carte se déplie à l'intérieur de la distribution des points, n'est pas très bonne, dans le cas de paramètres constants.

Pour résoudre ce problème, des paramètres variants en fonction du temps ont été préférés aux paramètres constants. Présentons rapidement cette deuxième version de l'algorithme de Kohonen :

- choix du vainqueur :

$$y_k = min_i(dist(w_i,\ x))$$

- mise à jour des poids :

$$\begin{cases} w_i(t+1) = w_i(t) + \alpha(t)(x - w_i(t)) & si\ d(y_k,\ x_i) \leq distmax(t) \\ w_i(t+1) = w_i(t) & si\ d(y_k,\ x_i) > distmax(t) \end{cases}$$

où

- $distmax(t)$ est la fonction de la distance de voisinage, décroissante au cours du temps,
- $\alpha(t)$ est la fonction du taux d'apprentissage, elle aussi décroissante au cours du temps.

Par décroissant au cours du temps, il faut comprendre que la valeur de ces paramètres diminue en fonction du nombre d'itérations déjà effectuées, une itération étant la présentation d'une donnée au réseau.

L'annexe B.2.2 présente l'évolution d'une carte de Kohonen dans des conditions comparables au cas précédent, excepté pour les paramètres « taux d'apprentissage » et « distance de voisinage », ici variables. On peut y constater une meilleure convergence de la carte.

Généralement, les fonctions décroissantes utilisées sont de type hyperbolique ou exponentielle pour le taux d'apprentissage; linéaire ou exponentielle pour la distance de voisinage. Le choix des fonctions décroissantes à utiliser est un paramètre supplémentaire de la méthode. L'annexe B.2.3 présente les lois de décroissance que nous avons utilisées, et l'impact de l'utilisation, pour chaque paramètre, de différentes lois de décroissance.

2.3.4 Algorithme complet

L'algorithme complet peut être résumé selon la procédure effective suivante :

Initialiser la position des neurones.
Répéter autant de fois que de présentations de l'ensemble des points :
    Pour chacun des points de l'ensemble :
        Déterminer le neurone vainqueur,
        Modifier la position du vainqueur et de ses voisins, le cas échéant.

La première partie de l'algorithme initialise les positions des neurones au sein de la distribution des points. Bien entendu, les neurones ont la même dimension que ces points. Notons que plusieurs techniques sont possibles pour l'initialisation : prendre des valeurs à l'intérieur de l'espace des points, de façon aléatoire; prendre des valeurs à l'intérieur des points dont on dispose; prendre la moyenne de tous les points; ... L'initialisation pourrait donc être considérée comme un des paramètres de l'algorithme. Ce n'est toutefois pas le cas, simplement parce que dans le cadre de ce travail, nous considérons toujours une initialisation des neurones avec les valeurs de certains points, choisis aléatoirement.

La suite de l'algorithme consiste en une présentation des données à la carte, ce qui provoque une mise à jour du poids de certains neurones, après détermination du vainqueur.

2.3.5 Propriétés des cartes de Kohonen

Les cartes de Kohonen possèdent deux propriétés intéressantes que nous allons maintenant décrire.

2.3.5.1 La quantification vectorielle

Le concept de quantification vectorielle a été introduit à la section 2.3.2 de ce chapitre. Nous avons ensuite pu énoncer le fait que l'algorithme de « Competitive Learning » permet de quantifier vectoriellement un ensemble de données.

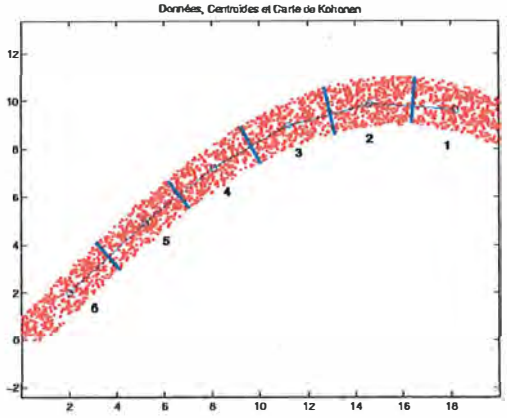
Puisqu'on sait que l'idée de l'algorithme de Kohonen est relativement proche de celle du « Competitive Learning », il est légitime de penser que dans ce cas également, il est possible de quantifier vectoriellement un ensemble de données. C'est effectivement le cas, tout en précisant que ce sont les neurones des cartes de Kohonen qui jouent le rôle des centroïdes. Par conséquent, une carte auto-organisée permet de résumer une distribution de points, après convergence, sur base de la position de ses neurones au sein de cette distribution.

Soulignons à ce propos que les illustrations qui figurent dans cette section 2.3.2 ont été obtenues au moyen d'une ficelle de Kohonen comportant 5 neurones. Les liens entre neurones n'avaient pas été indiqués, pour alléger la représentation graphique.

2.3.5.2 Le respect de la topologie

Dans le cas de l'algorithme de Kohonen, nous avons parlé d'une relation de voisinage entre les neurones. Cette relation est à la base de la propriété de respect de la topologie. Pour rappel, le voisinage est une notion associée à la position physique des neurones y_i au niveau de la carte.

En termes de quantification vectorielle, deux neurones sont voisins lorsque leurs zones de Voronoï respectives sont adjacentes :



Les conventions utilisées sont les suivantes :

- les données sont représentées par des points rouges,
- les neurones sont représentés par des ronds bleus,
- les liens symbolisant la relation de voisinage physique au niveau de la carte sont représentés par des segments de droite bleus reliant deux neurones.

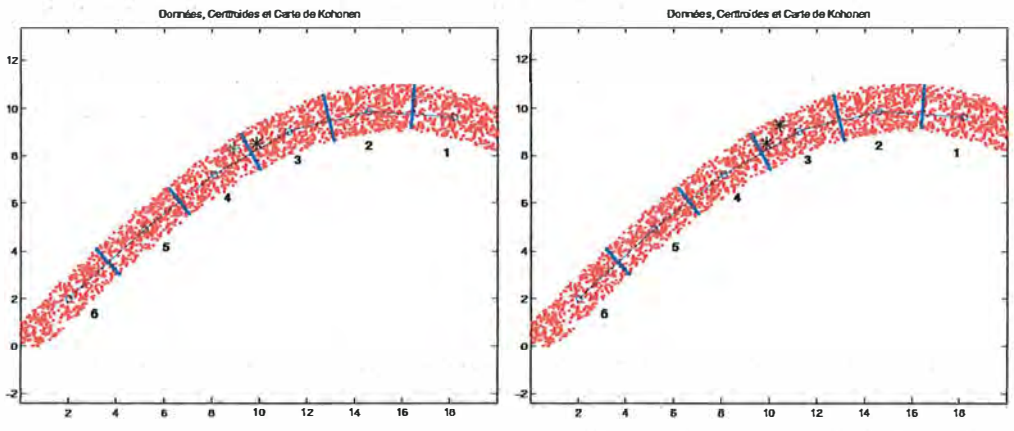
Les segments bleus plus épais sont les frontières des zones de Voronoï.

Insistons sur le fait que cette notion de centroïdes voisins porte sur *la position physique* des centroïdes les uns par rapport aux autres.

La propriété de respect de la topologie peut s'énoncer comme suit : si on considère deux points proches dans l'espace de départ, alors ces deux points sont :

- soit projetés sur un même centroïde,
- soit projetés sur deux centroïdes voisins.

Reprenons l'exemple précédent pour illustrer cette propriété :



Le premier schéma, à gauche, nous montre deux points respectivement noir (zone de Voronoï numéro 3) et vert (zone de Voronoï numéro 4), qui sont projetés sur deux centroïdes distincts, respectivement le centroïde numéroté 3 et celui numéroté 4. Ces deux centroïdes sont bien voisins l'un de l'autre; ce qui peut être constaté au niveau de la position des centroïdes sur la carte, ou au niveau des zones de Voronoï, effectivement adjacentes. Dans le second cas, à droite, les deux points noirs de la $3^{ème}$ zone de Voronoï sont projetés sur le même centroïde, numéroté 3.

En toute généralité, s'il y a plus de points, plus de centroïdes, ou encore si nous utilisons des points dans un espace de dimension supérieure à deux, cette propriété est toujours vraie.

2.3.6 Les paramètres du modèle

Nous avons dit qu'il était préférable d'utiliser des paramètres fonction du temps. Quelle est effectivement l'importance des paramètres du modèle ? C'est ce que nous allons maintenant mettre en évidence, en considérant le taux d'apprentissage, la distance de voisinage, le nombre de neurones, et la forme de la carte.

2.3.6.1 Le taux d'apprentissage

Le taux d'apprentissage α est le paramètre présent dans la première équation, et qui sert à adapter les positions des neurones au sein du réseau. Ce paramètre influence l'amplitude de cette adaptation, et permet au réseau de modifier son comportement au cours de l'apprentissage.

Notons que, puisque ce paramètre multiplie la différence entre la donnée x et le poids w_i du neurone y_i, plus α est grand, plus l'impact sur la variation de ce poids w_i sera grande. L'importance des mises à jour des poids est donc directement liée à la valeur de ce paramètre.

Généralement, la valeur de α est comprise entre 0 et 1 ; et dans la majorité des cas, elle ne dépasse pas 0.5.

2.3.6.2 La distance de voisinage

La distance de voisinage est le paramètre *distmax* présent dans les deux équations de mise à jour des poids w_i. Cette distance recouvre le concept d'influence mutuelle entre deux neurones dans un certain voisinage. Intuitivement, plus la distance de voisinage est grande, plus il y aura de neurones affectés par la présentation d'une donnée à la carte. La mise à jour des poids se fait dans une certaine région autour du neurone vainqueur, la limite de cette région étant ce paramètre *distmax*.

Concrètement, la valeur de *distmax* est fonction du nombre de neurones présents sur la carte. De plus, pour obtenir une bonne convergence, il est important de terminer l'exécution de l'algorithme avec une distance de voisinage nulle, ce qui veut dire qu'en fin d'apprentissage, seul le neurone vainqueur est modifié.

2.3.6.3 Le nombre de neurones

Le nombre de neurones est un des aspects critiques de la méthode. En effet, l'algorithme de Kohonen donne de meilleurs résultats lorsque le nombre de neurones est élevé. Ce fait est facilement compréhensible : à partir du moment où l'algorithme réduit par quantification vectorielle l'information contenue dans tous les points à un certain nombre de neurones, plus il y a de neurones, moins cette information sera réduite et meilleures seront les réponses apportées par le réseau. On peut même considérer le cas limite où il y a autant de centroïdes que de points. Si chacun des centroïdes représente toute l'information d'un point de l'ensemble de données, il n'y a à ce moment aucune perte d'information, les résultats étant les meilleurs possibles.

Malheureusement, ce fait n'est pas tout à fait vrai dans la réalité, car le phénomène de sur-apprentissage, ou « over-fitting » dans la littérature, apparaît quand le nombre de

paramètres devient trop important par rapport au nombre de données, les paramètres étant ici les poids des centroïdes. Intuitivement, ce sur-apprentissage peut être compris comme une mise à jour continuelle des poids, le réseau ayant du mal à déterminer quel centroïde est vainqueur pour les différents points. Dans notre cas, lors de la présentation d'une donnée, un certain neurone sera vainqueur ; puis au cours de la présentation suivante du même point, un autre neurone sera également vainqueur. La mise à jour revient donc à associer le point une fois à un tel neurone, une fois à un tel autre neurone. Dans ce cas, le comportement du réseau est relativement mauvais : il lui faut beaucoup plus de temps pour réaliser la quantification vectorielle des données entre les différents neurones, vu qu'il a des difficultés pour déterminer le vainqueur définitif de certains points (en espérant qu'il puisse toujours décider).

Par ailleurs, l'élément le plus pénalisant, d'un point de vue temps de calcul, est le nombre de neurones, puisque plus il y a de neurones, plus il y a de possibilités à tester pour déterminer le vainqueur, et ce pour chaque donnée. Par conséquent, il y a aussi un plus grand nombre d'instructions machines à réaliser.

Un grand nombre de neurones n'est donc pas avantageux en pratique. Un certain équilibre doit être trouvé entre la perte d'information, due à la quantification vectorielle, et la perte de performances, due au phénomène de sur-apprentissage.

2.3.6.4 La forme du réseau

La forme du réseau de Kohonen représente en fait la façon dont les neurones sont disposés physiquement les uns par rapport aux autres. Généralement, on considère que les réseaux de Kohonen sont des ensembles de neurones disposés dans un espace de dimension 1 ou 2, (très) rarement plus.

Si le réseau est de dimension 1, alors il est appelé ficelle de Kohonen. En effet, la position des neurones au sein de l'espace des données ainsi que les liens entre ces neurones forment une figure géométrique semblable à une ficelle.

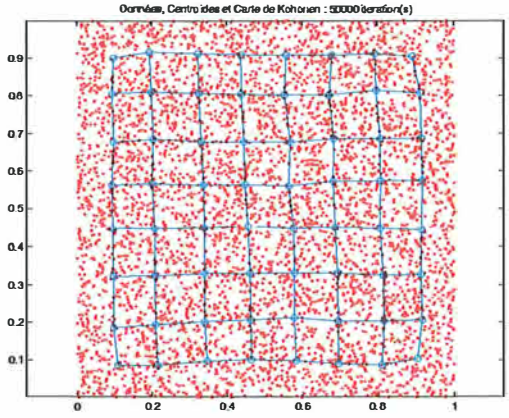
Si le réseau est de dimension 2, il est appelé carte de Kohonen, vu que les neurones se positionnent selon une structure rectangulaire, voire carrée. Cette structure est elle-même constituée de plusieurs petits rectangles ou carrés de plus petite taille, avec un neurone à chaque extrémité, les côtés des rectangles, ou des carrés, étant les liens entre les neurones.

2.3.7 Comportement du modèle

Pour rappel, l'annexe B.2 présente le comportement du modèle lors de l'application de cet algorithme, dans le cas des paramètres constants, puis variables. L'annexe B.3 nous renseigne sur l'importance relative de chacun des paramètres sur le résultat final de l'algorithme. L'exemple utilisé est le même que celui de l'annexe B.2, où on fait varier un à un les paramètres. Quelques « astuces » importantes à retenir en pratique sont énoncées.

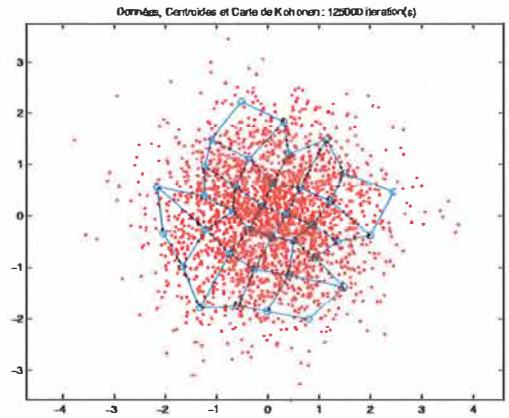
Le lecteur désireux de se faire une intuition sur le comportement des cartes de Kohonen lors de l'application de l'algorithme est donc invité à consulter cette annexe A.2. Nous nous limiterons ici à présenter quelques cas de convergence, observés dans différentes distributions « intéressantes » de points.

Premier exemple : convergence d'une carte carrée dans une distribution carrée de points générés aléatoirement selon une loi uniforme, avec 64 neurones ; un taux d'apprentissage variant de 0,02 à 0,001 selon une loi exponentielle décroissante ; une distance de voisinage variant de 3 à 0, selon une loi linéaire décroissante ; et 10 présentations des points (50 000 itérations) :



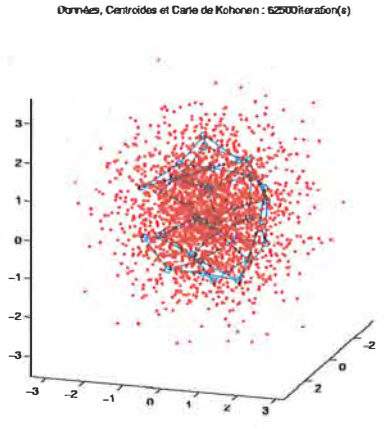
Pour rappel, les lois de décroissances sont décrites en annexe B.2.3.

Autre exemple : convergence d'une carte de Kohonen dans une distribution gaussienne en deux dimensions :



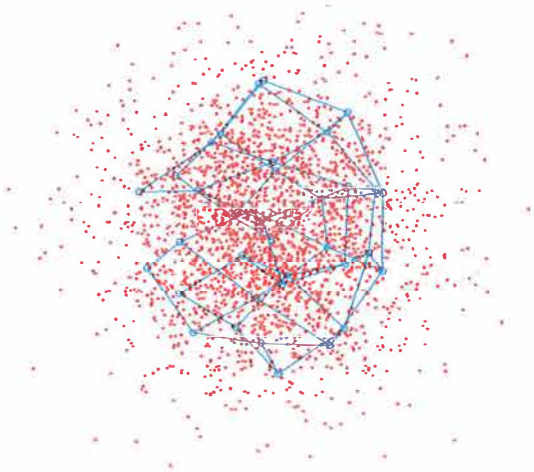
Ce résultat est obtenu avec une carte carrée de 36 neurones, un taux d'apprentissage variant exponentiellement de 0,03 à 0,001, une distance de voisinage linéairement décroissante de 2 à 0, et 50 présentations des 2 500 points contenus dans le fichier d'apprentissage.

Dans le cas d'une distribution gaussienne, mais en trois dimensions cette fois :

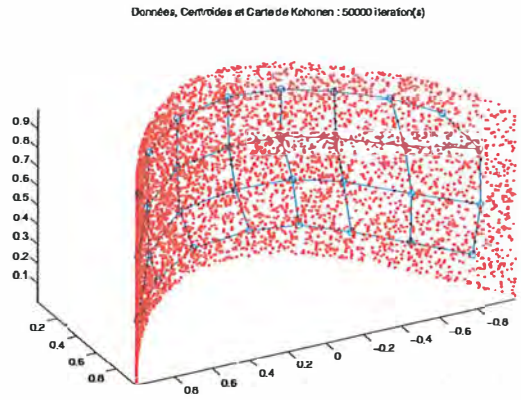


Le résultat est ici plus difficilement observable.

En effectuant un zoom sur le centre de la distribution, il est possible d'observer la bonne convergence de la carte :

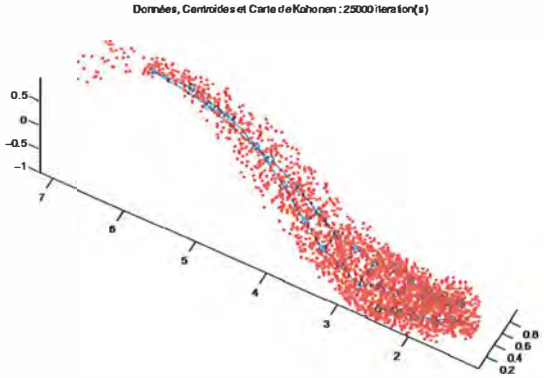


Ici aussi, une carte carrée de 36 neurones a été utilisée, avec un taux variant exponentiellement entre 0,02 et 0,001, un voisinage variant linéairement de 2 à 0, et 25 présentations des 2 500 points d'apprentissage. Dans le cas d'un paraboloïde :



Cette convergence a été obtenue en utilisant une carte rectangulaire de 32 neurones (8x4), un taux décroissant selon une loi exponentielle de 0,02 à 0,001, un voisinage décroissant selon une loi linéaire entre 2 et 0, et 10 présentations des 5 000 points disponibles.

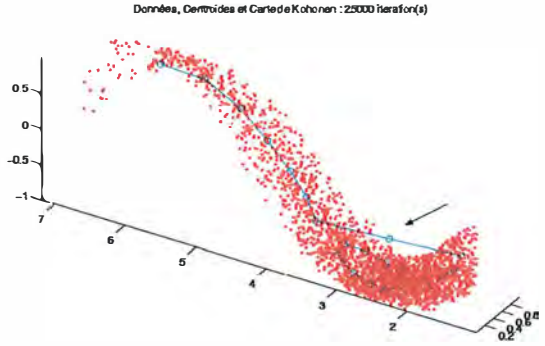
Dernier exemple : une distribution de points générés selon une loi uniforme combinée à l'application croisée d'une loi exponentielle et un cosinus :



Les valeurs des paramètres sont ici : carte rectangulaire de 34 neurones (17x2), un taux décroissant exponentiellement de 0,02 à 0,001, une distance de voisinage décroissant linéairement entre 2 et 0, 25 000 itérations (10 présentations des 2 500 points d'apprentissage).

Notons que la carte est volontairement « longue » (17x2). Le but était de pouvoir mettre en évidence le comportement local de la carte : il y a plus de centroïdes dans la partie de la distribution où il y a le plus de points. Par ailleurs, en prenant volontairement une carte longue, on pouvait espérer voir un repli de la carte sur elle-même, vu la forme de la distribution. C'est effectivement le cas, la carte se repliant même en deux parties pratiquement parallèles entre elles.

Vu la forme de la distribution, on peut penser à utiliser une ficelle à la place d'une carte :



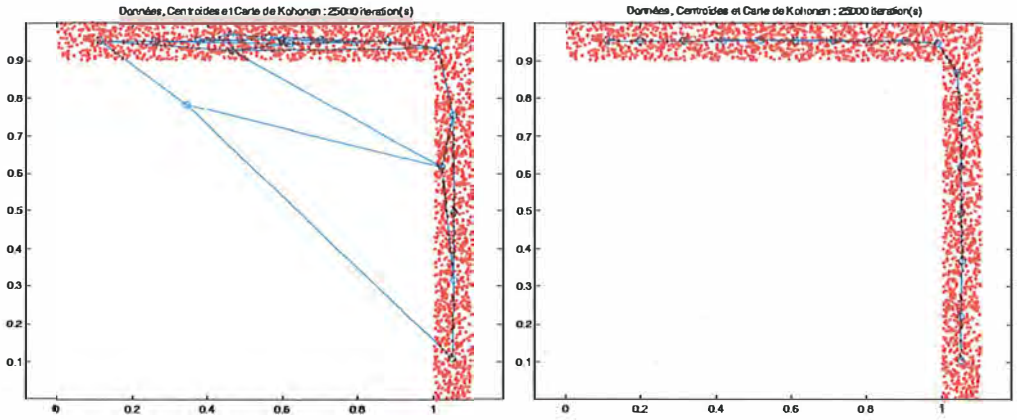
Les paramètres sont : 23 neurones sur une ficelle ; avec un taux variant exponentiellement de 0,015 à 0,001 ; une distance de voisinage décroissant linéairement de 2 à 0 ; 25 000 itérations. Le comportement est globalement le même, la carte se repliant à nouveau sur elle-même dans la partie inférieure de la distribution.

Cependant, on peut observer un phénomène nouveau : un centroïde est 'perdu', en dehors de la distribution. Ce phénomène est désigné ci-dessus par la flèche, qui pointe vers ce centroïde. Dans la littérature, ce cas particulier est désigné sous le terme de centroïde mort.

2.3.8 Le cas des centroïdes morts

Le dernier exemple ci-dessus nous a permis d'introduire la notion de centroïde mort. Ce problème survient en fonction de la distribution des données et de la carte qui est choisie.

Dans le cas d'une distribution de 2 500 points, comportant un angle droit :

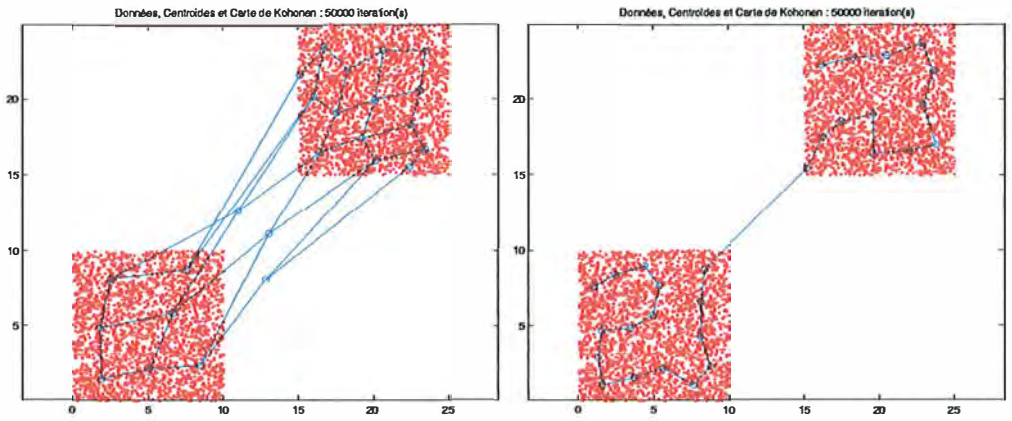


Les paramètres sont respectivement :

- à gauche : 18 neurones sur une carte rectangulaire 6x3, un taux d'apprentissage décroissant exponentiellement de 0,02 à 0,01, une distance de voisinage décroissant linéairement de 2 à 0,
- à droite : 17 neurones sur une ficelle, un taux d'apprentissage décroissant exponentiellement de 0,02 à 0,01, une distance de voisinage décroissant linéairement de 3 à 0.

Dans les deux cas, les points ont été présentés 10 fois, soit un total de 25 000 itérations.

Autre exemple : une distribution comportant deux régions disjointes :



Mentionnons les valeurs des paramètres :

- à gauche : 25 neurones sur une carte carrée, taux d'apprentissage variant exponentiellement de 0,02 à 0,01, distance de voisinage variant linéairement de 2 à 0, 10 présentations des points,
- à droite : 29 neurones sur une ficelle, taux variant exponentiellement de 0,02 à 0,01 voisinage variant linéairement de 2 à 0, 10 présentations des points.

Dans les deux cas, 25 000 itérations ont été effectuées.

Dans les deux exemples de gauche ci-dessus, la quantification vectorielle est correcte, les neurones étant bien répartis à l'intérieur de la distribution. Malgré cette bonne quantification vectorielle, la convergence de la carte n'est pas la meilleure qui soit : on observe des « neurones morts ». Ces neurones morts sont perdus pour la carte : ils ne seront jamais vainqueurs pour aucune des données présentées au réseau.

Une première conclusion, trop rapide, serait de dire que les ficelles permettent d'éviter le phénomène des neurones perdus. C'est effectivement le cas dans les deux exemples ci-dessus, mais il ne faut pas oublier la dernière distribution de la section précédente, où une carte rectangulaire ne posait pas de problème alors que la ficelle comportait un neurone mort.

Retenons plutôt qu'il faut faire attention à chacun des paramètres pour éviter ce problème qui, bien que relativement rare, peut effectivement être rencontré en pratique. Une bonne étude des données, avant d'appliquer l'algorithme de Kohonen, peut par ailleurs aider à choisir des valeurs de paramètres judicieuses.

2.3.9 Les données creuses

Supposons qu'on décide d'appliquer la méthode de Kohonen à une base de données relatives à un problème *concret*. Les points de la base de données constituent des mesures du problème qui est en cours d'étude. Or qui dit mesure dit risque d'erreur. Plus grave encore, la mesure peut ne pas avoir été effectuée, pour de multiples raisons : préposé à la mesure absent à ce moment là, appareil de mesure en panne, ...

Dans la réalité, ce cas est très courant. Il est même d'autant plus courant qu'en général, quand on effectue une mesure, on réalise en fait un ensemble de n mesures qui sont prises en même temps et constituent une seule et même mesure, à n composantes. Malheureusement, plus il y a de mesures à faire, plus le risque d'en rater une, ou plus, est élevé.

Par la suite, nous emploierons le terme de donnée creuse pour désigner une mesure dont une de ses composantes, au moins, est non disponible. Si toutes ses composantes sont non disponibles, la donnée n'en est pas une. Il n'est donc pas pertinent de la conserver dans l'ensemble des points.

Au niveau de l'algorithme de Kohonen, quel peut être l'impact de ce nouveau problème ? Trois moments de l'algorithme peuvent poser problème :

- L'initialisation : si on initialise les centroïdes avec des points de l'ensemble choisis de façon aléatoire, que faire si un de ces centroïdes est une donnée creuse ?
- Le choix du centroïde vainqueur : ce choix s'effectue sur base d'un calcul de distance entre deux points d'un espace à n dimensions. Comment calculer la distance quand une des composantes est manquante ?
- La mise à jour de la position du centroïde : comment modifier la composante creuse d'un centroïde ? Quelle modification faut-il apporter à la composante d'un centroïde pour qu'elle se rapproche d'une donnée creuse, ce rapprochement étant calculé sur base de la composante manquante de la donnée ?

L'algorithme de Kohonen peut être généralisé pour tenir compte de ces cas particuliers. C'est ce que nous avons dû faire dans le cadre de ce mémoire. En raisonnant sur la manière de manipuler les données creuses, nous avons pu résoudre les trois problèmes décrits ci-dessus. Justifions intuitivement les modifications apportées à l'algorithme de Kohonen pour lui permettre d'être appliqué à un ensemble de données comportant des données creuses :

- initialisation : on incorpore dans la procédure d'initialisation un traitement particulier pour le cas de données creuses. On peut dès lors être certain qu'à la fin de cette initialisation, aucun centroïde n'est une donnée creuse. Le traitement particulier que nous avons retenu est l'initialisation de la composante manquante par la moyenne de cette composante, calculée sur les autres données. Ce choix peut être remis en cause si une composante est toujours manquante, pour toutes les données disponibles. Nous considérons ce cas comme improbable, ou en tout cas sans intérêt. En effet, pourquoi analyser une composante qui n'a jamais fait l'attention d'une seule mesure ? Y a-t-il réellement un intérêt à inclure dans le fichier des colonnes entièrement vides ? Devant l'évidence de la réponse, cette possibilité a été écartée.
- choix du centroïde vainqueur : on peut toujours considérer que le centroïde le plus proche est celui situé à la plus petite distance de la donnée, même si elle est creuse. Un changement doit simplement être apporté dans la méthode utilisée pour calculer la distance. On peut, par exemple, ne prendre en compte, lors de ce calcul, que les composantes non manquantes. La distance utilisée est dès lors une variante de la distance euclidienne, où seules certaines composantes sont utilisées. Plus formellement, la distance $dist(a, b)$ entre deux points $a = (a_1, a_2, \ldots, a_m)$ et $b = (b_1, b_2, \ldots, b_m)$ est maintenant calculée sur base de la relation :

$$dist(a, b) = \sqrt{\sum_{\substack{i=1 \\ a_i \neq \perp \\ b_i \neq \perp}}^{m} (a_i - b_i)^2}$$

où $\perp$ est une valeur indiquant que la composante correspondante de la donnée est

manquante, et m est la dimension des données. Le centroïde vainqueur est donc bien celui qui se trouve le moins loin, mais sur base de quelques composantes seulement.
- modification de la position du centroïde : si le centroïde ne comporte aucune composante manquante, ce dont nous pouvons être assurés étant donné la nouvelle version de l'initialisation, le premier aspect de ce troisième problème est résolu. De plus, si on décide de ne modifier que les composantes du centroïde correspondant aux composantes non manquantes de la donnée, le problème est entièrement résolu !

Insistons sur le fait que l'algorithme de Kohonen tel que nous l'avons modifié résiste au problème des données creuses. Ce fait constitue un avantage non négligeable quand on sait qu'il s'agit là d'un problème relativement fréquent dans la réalité.

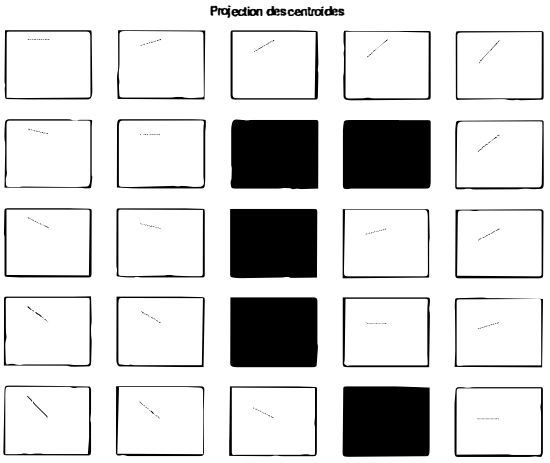
2.3.10 Résumé

Rassemblons en quelques lignes les principaux concepts des cartes de Kohonen.

Les *cartes de Kohonen* sont une variante des *réseaux de neurones artificiels*, dont l'*apprentissage* est *non supervisé*, ce qui conduit à une *auto-organisation* de la *carte*. Généralement, la carte est de *dimension 1 ou 2*. La *règle d'apprentissage* est basée sur le « *Competitive Learning* », avec recherche du *neurone vainqueur* parmi tous les neurones de la carte, dont le *nombre* est un paramètre de la méthode. Il y a ensuite une *mise à jour* des *positions des neurones* dans un certain *voisinage*, également un des paramètres de la méthode. Cette *mise à jour* est fonction du *taux d'apprentissage*, autre paramètre de la méthode. De l'application de l'algorithme de Kohonen résulte une *quantification vectorielle* des données par les neurones de la carte. Les données pouvant être associées à un neurone proche, on obtient une *partition* des données en *zones de Voronoï*, dont les *centroïdes* respectifs sont les neurones de la carte. Après *convergence*, cette carte *respecte la topologie* des données. Il est possible de rencontrer des *centroïdes morts*. Enfin, la méthode résiste au cas des *données creuses*.

En pratique, les cartes de Kohonen servent souvent, vu leur propriété de quantification vectorielle, comme outil d'analyse pour des applications où, par exemple, on cherche à faire de l'extraction de motifs, ou encore lorsqu'on souhaite connaître certaines propriétés statistiques de données. L'avantage des cartes de Kohonen dans ce genre d'applications, lors de l'utilisation de données de grandes dimensions, c'est la possibilité de visualiser ces données en deux dimensions, en utilisant une certaine convention de représentation.

Par exemple, dans le cas de la seconde simulation de l'annexe B.2.2, avec paramètres variables, une carte carrée de 25 neurones avait été utilisée :



Chacun des 25 graphes ci-dessus représente l'information contenue dans un des 25 centroïdes de la carte, après convergence. Intuitivement, cette information est constituée du poids du centroïde dans la distribution des points. Autrement dit, l'image ci-dessus est la représentation graphique des poids w_i des centroïdes y_i, $1 \leq i \leq 25$. Par exemple, dans le coin supérieur gauche, on retrouve le premier centroïde dont le poids, après convergence est (0.8936, 0.8978). Ces deux valeurs sont les extrémités du segment du graphe se trouvant dans le coin supérieur gauche. En dimensions supérieures, nous obtenons donc une courbe passant par toutes les composantes du vecteur poids associé à chaque centroïde, et non plus un segment de droite. Cette convention de représentation nous permet d'observer la « forme » des centroïdes, qui sera par la suite appelée profil du centroïde.

On peut également représenter les profils des points, de la même façon, en les associant à la zone de Voronoï à laquelle ils appartiennent. Cette projection des points n'est pas ici présentée, pour ne pas surcharger l'image avec les valeurs des 2 500 points.

Notons enfin que les cartes auto-organisées peuvent être utilisées pour d'autres applications maintenant classiques dans le domaine des réseaux de neurones artificiels : compression d'image, séparation de sources, reconnaissance, etc.

2.3.11 Concrètement ...

Dans le cadre de ce mémoire, l'algorithme de Kohonen a été codé en C++. Il a ensuite été généralisé pour pouvoir être utilisé indifféremment avec ou sans données creuses. Différentes méthodes d'initialisation ont été envisagées, pour ne retenir finalement que le choix de certaines données à l'intérieur de l'ensemble d'apprentissage. Cette méthode classique a toutefois dû être particularisée pour être compatible avec le reste de l'algorithme qui peut traiter le cas des données creuses.

Pourquoi avoir codé cet algorithme quand on sait qu'il en existe plusieurs versions disponibles sur Internet ? Plusieurs raisons expliquent ce choix :

- L'ensemble du code écrit dans le cadre de ce mémoire est en C++. Ne connaissant pas ce langage, il nous a paru intéressant de commencer par une première étape, relativement courte, de programmation, pour s'habituer au langage et à son compilateur.
- Le code développé se base sur les librairies d'opérations sur les matrices de Monsieur John A. Lee. Le codage de l'algorithme de Kohonen a donc permis de comprendre le fonctionnement de ces librairies.
- A partir du moment où cet algorithme devait être abondamment utilisé par la suite et, surtout, intégré dans une autre méthode, il nous a semblé important de comprendre comment il fonctionne, comment on peut jouer sur ses paramètres et comment il réagit aux points qui lui sont présentés.

Rappelons pas que le but principal de ce mémoire est de décrire l'évolution de données fonction du temps, les cartes de Kohonen constituant un outil de cette prédiction. Une bonne compréhension de cet outil est donc un atout pour la suite de ce travail.

Chapitre 3

Présentation de la méthode de prédiction de données temporelles

3.1 Introduction

Ce chapitre introduit la méthode développée dans l'article [6], base de l'algorithme qui a été développé et implémenté dans le cadre de ce mémoire. Pour des raisons de cohérence et de clarté, les notations utilisées seront aussi souvent que possible les mêmes que celles de l'article.

Seront ici présentées les différentes étapes de la méthode. Les termes et notions utilisés dans ces étapes seront définis au fur et à mesure des explications données dans ce chapitre. Toutes ces notions et les principales étapes de la méthode seront illustrées par un exemple général. Les paramètres de la méthode seront ensuite étudiés. Une seconde version de la méthode, ainsi que les modifications qui lui ont été apportées, seront enfin décrites.

Notons que le choix de la méthode a été influencé par l'utilisation qui allait en être faite. Sachant que le fichier de Gaz de France était constitué de données non scalaires, cette méthode a été préférée à d'autres parce qu'elle est spécifiquement adaptée pour ce genre de données.

3.2 Présentation générale

La méthode de prédiction de données temporelles que nous proposons est basée sur une double application de l'algorithme de Kohonen qui a été présenté dans la seconde partie du chapitre précédent.

Principalement deux étapes distinctes peuvent être mises en évidence dans cette méthode de prédiction.

La première est appelée la caractérisation, et consiste en une classification des données du problème par application de l'algorithme de Kohonen. Cette classification revient à résumer l'information décrivant l'évolution passée d'un processus évoluant au cours du temps.

La seconde est la prédiction proprement dite. Cette prédiction ne peut être effectuée que lorsque la partie caractérisation est terminée. Intuitivement, l'information passée, résumée lors de la caractérisation, est reprise dans un certain ordre pour essayer de décrire une évolution future relativement probable du processus temporel étudié.

3.3 La méthode de prédiction des données temporelles

3.3.1 Note sur le calcul du régresseur

Dans le chapitre 1, la section 1.4 présentait d'un point de vue théorique assez général le problème de la prédiction de données temporelles. Cette section nous a permis d'introduire le concept de modèle d'un processus fonction du temps. Ce modèle nous permet d'estimer la valeur qui serait observée en un temps t sur base de la relation :

$$\hat{x}_t = g(x_1, x_2, x_3, \ldots, x_n, t)$$

où $\hat{x}_t$ constitue une estimation de la valeur x_t réellement donnée par le processus que $g(.)$ modélise. A ce moment de l'introduction, nous avions fait remarquer que tout l'art consistait à trouver, parmi toutes les données $x_1, x_2, x_3, \ldots, x_n$, celles qui ont une réelle influence sur la modélisation. Autrement dit, la difficulté est de pouvoir déterminer quelles données, parmi celles qui sont disponibles, nous aident à construire un modèle proche de la réalité.

En reprenant ces notions, nous pouvons définir le terme de régresseur. Un régresseur est l'ensemble des données passées qui nous permettent de construire le modèle du processus étudié.

Illustrons de suite ce concept, pour clarifier les idées.

Considérons un problème de prédiction de consommation électrique. A première vue, on peut estimer que la consommation de demain ne sera fonction que de celle d'aujourd'hui. Dès lors, le régresseur ne comprendra que la dernière valeur mesurée : x_n. Mais la réalité n'est pas aussi simple. Si nous sommes dimanche, jour de repos hebdomadaire, il est certain que le redémarrage de l'activité d'une grande partie de la population, le lendemain, aura un impact sur la modélisation de la consommation électrique de demain. Par conséquent, il n'est pas inutile de faire rentrer dans le régresseur la valeur de lundi dernier, soit x_{n-6}. De même, on peut estimer que la consommation électrique sera semblable à la consommation de l'année dernière, à la même date. On peut dès lors tenir compte des effets annuels, au même titre que x_{n-6} nous permet de prendre en compte les phénomènes hebdomadaires. Le régresseur devra alors comprendre la valeur x_{n-364}. On peut dès lors continuer à intégrer toute une série de valeurs au régresseur, pour prendre en considération les variations saisonnières, ... Le but est d'affiner la prédiction qui est effectuée grâce au modèle, ce dernier étant basé sur le régresseur que nous sommes en train de construire.

Bien entendu, on peut toujours supposer que chaque valeur passée a son importance dans le modèle en cours d'élaboration. On peut dès lors inclure au régresseur l'ensemble des données passées, obtenant ainsi le régresseur $x_1, x_2, x_3, \ldots, x_n$.

Le problème de ce régresseur est qu'il est extrêmement complexe. Par conséquent, bien qu'il soit très proche de la dynamique réelle du processus, il ne donnera pas forcément de bons résultats, vu la difficulté de sa mise en oeuvre et les contradictions qu'il peut y avoir entre les données passées. Il y a donc lieu de réfléchir sur la façon dont on peut construire un régresseur de taille (et donc de complexité) acceptable mais malgré tout correct. Cet optimum est toutefois dur à obtenir : aucune méthode générale ne permet de définir le régresseur optimal de façon efficace et certaine. La seule méthode reste donc, malheureusement, un test exhaustif qui permet d'envisager toutes les possibilités les unes après les autres ...

Remarquons que le calcul d'un régresseur optimal dépasse le cadre de ce travail. Cette note a pour seul but d'attirer l'attention sur le concept de régresseur, qui reviendra régulièrement dans le reste de ce chapitre et de ce travail.

3.3.2 Partie caractérisation

3.3.2.1 Application du régresseur aux données

La partie initiale de l'algorithme consiste en un chargement des données, qui sont aussi notées D, l'ensemble des données du problème. Ces données sont supposées scalaires dans l'article, ce que nous supposons également, pour simplifier l'explication. Par ailleurs, on suppose que ces données sont classées dans l'ordre chronologique. Nous choisissons de qualifier de telles données comme étant sous forme « brute », du point de vue de la méthode.

Une fois ce chargement des données effectué, l'étape suivante consiste à appliquer le régresseur qui a été choisi. Le régresseur est généralement appelé « lagging order » dans la littérature, de même que dans l'article, et sera ici aussi noté λ. Le fait d'appliquer le régresseur aux données nous fait passer d'une forme « brute » des données à une forme « non brute ». Ces termes « brute » et « non brute » seront souvent utilisés par la suite.

Dans le cas de données scalaires, l'application du régresseur aux données revient à créer des données de dimension λ, dont les λ composantes sont des données scalaires, considérées suivant leur ordre dans le temps, sur un intervalle de temps de longueur λ.

Plus concrètement, considérons un ensemble D de données scalaires fonction du temps, sous forme « brute ». Notons ces données $x(t)$ où t représente un instant donné dans le temps. D peut donc être vu comme une matrice comportant une seule colonne et n lignes :

$$D = \begin{pmatrix} x(1) \\ x(2) \\ \vdots \\ x(n) \end{pmatrix}$$

où nous considérons que D contient n données « brutes ».

Après application du régresseur, ici de taille 4, par exemple, nous obtenons un autre ensemble, que nous notons P, et qui contient la forme « non brute » des données, elles aussi de taille 4 :

$$P = \begin{pmatrix} x(1) & x(2) & x(3) & x(4) \\ x(2) & x(3) & x(4) & x(5) \\ x(3) & x(4) & x(5) & x(6) \\ \vdots & \vdots & \vdots & \vdots \\ x(n-4) & x(n-3) & x(n-2) & x(n-1) \\ x(n-3) & x(n-2) & x(n-1) & x(n) \end{pmatrix}$$

En toute généralité, nous obtenons donc, à partir de l'ensemble des données de départ D, l'ensemble P, de dimension $[n-\lambda+1,\ \lambda]$

3.3.2.2 Première application de l'algorithme de Kohonen

L'ensemble P obtenu lors du calcul du régresseur est à partir de ce point appelé espace de base. Les points de cet espace P sont notés x_t, et définis par :

$$x_t = \{x(t-\lambda+1),\ \ldots,\ x(t)\}.$$

La méthode utilise alors une première fois l'algorithme de Kohonen pour réaliser une quantification vectorielle des points x_t de l'espace de base.

Il résulte de l'application de cet algorithme un ensemble de centroïdes $\bar{x}_i$, dont le nombre est défini a priori, et un ensemble de régions de l'espace R_i, contenant chacune un centroïde $\bar{x}_i$ et une série de points dont $\bar{x}_i$ est le centroïde le plus proche :

$$R_i = \{x_t \mid dist(x_t, \bar{x}_i) \leq dist(x_t, \bar{x}_j) \wedge 1 \leq j \leq nb1\}; \ 1 \leq i \leq nb1$$

où

- $nb1$ représente le nombre total de centroïdes utilisés pour cette première quantification de l'espace de base,
- $dist(.,.)$ est la distance utilisée lors de la quantification vectorielle par l'algorithme de Kohonen.

La quantification vectorielle par l'algorithme de Kohonen réduit donc l'information contenue dans l'espace de base en un certain nombre de points $\bar{x}_i$, respectivement représentatifs des différentes zones de Voronoï R_i de l'espace de base.

3.3.2.3 Deuxième application de l'algorithme de Kohonen

Sur base des données contenues dans P, rangées selon l'ordre chronologique d'un point de vue temporel et de forme « non brute », on crée l'espace de déformation contenant les déformations y_t des points de base x_t, suivant la formule :

$$y_t = x_{t+1} - x_t$$

On peut alors rassembler toutes les déformations y_t dans un seul ensemble, noté P'. Comme nous effectuons des différences entre deux points, P' est évidemment de dimension $[n - \lambda, \lambda]$.

L'étape suivante est de calculer des clusters P_i qui représentent chacun l'ensemble des déformations y_t associées aux points x_t de l'espace de base repris dans la région de Voronoï de centroïde $\bar{x}_i$. Autrement dit :

$$P_i = \{y_t \mid y_t = x_{t+1} - x_t \wedge \bar{x}_i = min_j dist(x_t, \bar{x}_j)\}; \ 1 \leq i \leq nb1.$$

L'algorithme de Kohonen est alors à nouveau appliqué, sur chacun des clusters P_i. Pour chacun de ces P_i, nous obtenons alors un ensemble de centroïdes $\bar{y}_j$ et de régions R_j qui sont définies de façon similaire aux régions R_i de l'espace de base :

$$R_j = \{y_t \mid dist(y_t, \bar{y}_j) \leq dist(y_t, \bar{y}_k) \wedge 1 \leq k \leq nb2\}; \ 1 \leq j \leq nb2$$

où $nb2$ est le nombre total de centroïdes dans chaque cluster P_i.

La dernière étape est un calcul des fréquences d'apparition des centroïdes dans l'espace de déformation, notées :

$$P(\bar{y}_j \mid \bar{x}_i).$$

Ce calcul s'effectue par comptage du nombre de points qui se trouvent dans chacune des régions R_j d'un ensemble P_i de l'espace de déformation, sachant que les points de base des déformations de P_i se trouvent dans une certaine région R_i de l'espace de base.

3.3.3 Partie prédiction

La seconde partie de la méthode consiste en une utilisation de la caractérisation effectuée dans la partie précédente afin de prédire les données suivantes d'un point de vue temporel. Cette prédiction se base sur l'hypothèse que l'évolution future du processus est comparable à son évolution passée. Autrement dit, la relation qui caractérise le problème étudié est supposée être toujours la même, que ce soit avant ou après la prédiction.

Soit le point x_t dont on cherche à connaître la valeur suivante x_{t+1}. La caractérisation étant finie, on connaît les positions des centroïdes dans l'espace de base. On peut dès lors rechercher le centroïde le plus proche, en utilisant la mesure de distance $dist(.,\ .)$ entre deux points :

$$\bar{x}_k \ = \ min_i \ dist(x_t,\ \bar{x}_i);\ 1 \leq i \leq nb1.$$

Dès que ce centroïde le plus proche $\bar{x}_k$ est connu, on connaît également la zone de Voronoï R_k qui lui est associée. A cette zone de Voronoï est associée un cluster P_k dans l'espace de déformation. On choisit alors aléatoirement, en fonction des fréquences observées, un centroïde $\bar{y}_j$ dans ce cluster P_k de l'espace de déformation.

La prédiction est alors calculée en 'retournant' la formule qui nous a permis de calculer les déformations. Partant de l'équation :

$$y_t \ = \ x_{t+1} - x_t$$

on obtient immédiatement :

$$x_{t+1} \ = \ x_t + y_t.$$

En utilisant l'information obtenue lors de la recherche du centroïde $\bar{x}_k$ le plus proche du point x_t et du choix d'un des centroïdes $\bar{y}_j$ du cluster P_k de l'espace de déformation, on obtient concrètement la prédiction par :

$$x_{t+1} \ = \ x_t + \bar{y}_j$$

où :

- x_t est l'élément dont on cherche à connaître le suivant,
- $\bar{y}_j$ est le centroïde choisi dans le cluster P_k de l'espace de déformation, associé à la région R_k de l'espace de base dont le centroïde $\bar{x}_k$ est le centroïde le plus proche de x_t parmi tous les centroïdes de l'espace de base,
- x_{t+1} est l'élément suivant prédit par cette méthode.

Cette manière de faire est la *prédiction à un pas*, où l'horizon de prédiction est la valeur suivante.

Par la suite, il est possible de connaître x_{t+2}, x_{t+3}, ... en faisant boucler la méthode jusqu'à la prédiction x_{t+k} voulue, k étant l'horizon de prédiction. Cette autre manière de faire est la *prédiction à long terme.*

Cette répétition de la prédiction nous donne une évolution possible à long terme en itérant les prédictions pas à pas. Bien entendu, cette évolution à long terme n'est qu'une évolution possible parmi toutes celles, très nombreuses, qui sont potentiellement réalisables. En recommençant les simulations de l'évolution jusqu'à l'horizon de prédiction k, on obtient une série d'évolutions futures possibles. La moyenne de ces évolutions est alors calculée. Cette moyenne représente l'évolution la plus probable, statistiquement.

La technique retenue pour effectuer le calcul de cette moyenne des prédictions, que ce soit à un pas ou à long terme, sera l'objet du chapitre suivant.

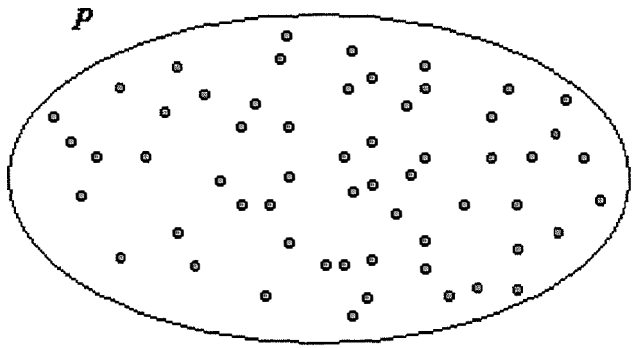
3.4 Illustration de la méthode

Dans cette partie, nous allons présenter un exemple général, dont l'intérêt principal est la représentation graphique qu'il propose. Cette représentation permet de mieux discerner les concepts d'espace de base et de déformation.

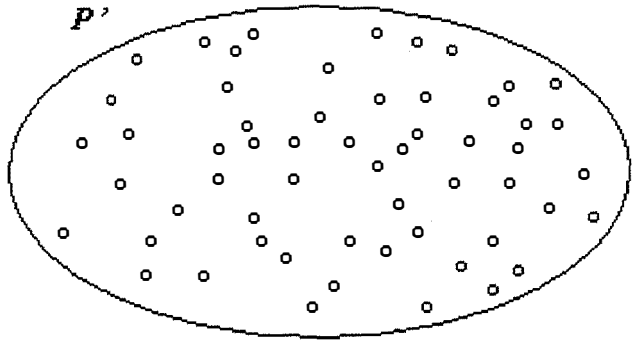
Cette représentation graphique présente deux colonnes : à gauche, il s'agit de l'espace de base, où les points sont représentés en vert. A droite, il s'agit de l'espace de déformation, où les points sont en orange. Les opérations effectuées dans chacun des espaces sont brièvement commentées et numérotées chronologiquement. Enfin, lors de la sélection de certains éléments, ceux-ci sont soit mis en rouge, dans le cas de points, soit agrandis, dans le cas d'une région ou d'un cluster de l'un ou l'autre des deux espaces.

Précisons que cette représentation est simplifiée. En effet, par construction, les déformations des points ne sont pas distribuées de la même façon que les points de base. Par conséquent, les frontières des clusters de l'espace de déformation ne sont pas les mêmes que celles des zones de Voronoï, créées dans l'espace de base lors de la quantification vectorielle. Cependant, pour simplifier l'illustration, on choisit arbitrairement de définir les frontières de ces clusters comme étant identiques aux frontières des zones de Voronoï.

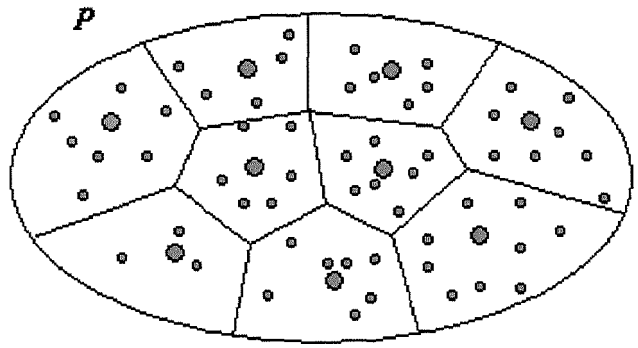
1. Application du régresseur Considérons un ensemble D de 60 données scalaires et un régresseur de taille $\lambda = 3$. L'ensemble P, de taille $[n - \lambda + 1,\ \lambda]$, comporte 58 éléments :



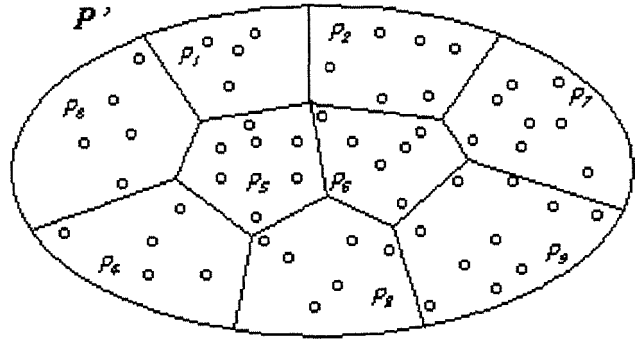
2. Calcul des déformations Parallèlement, on construit P', l'ensemble des déformations, de taille $[n-\lambda,\ \lambda]$, à savoir 57 données de dimension trois, dans ce cas-ci :



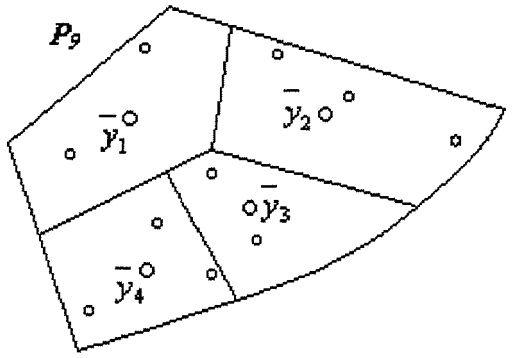
3. Première quantification L'algorithme de Kohonen est appliqué une première fois sur cet ensemble de base. Dans le cas d'une carte comportant 9 centroïdes $\bar{x}_i$, $1 \leq i \leq 9$, on obtient une quantification vectorielle des données selon ces 9 centroïdes, en 9 zones de Voronoï :



4. Calcul des clusters Puisqu'il y a 9 centroïdes, on crée 9 clusters, contenant les déformations y_t :



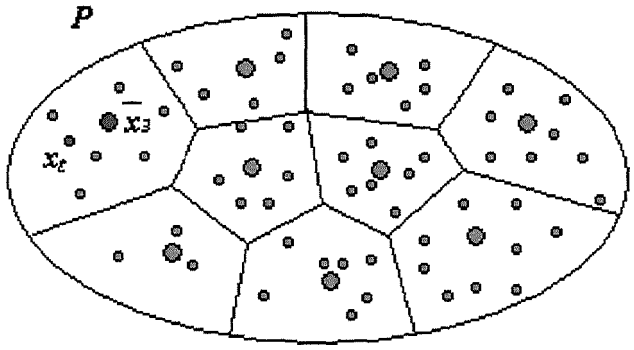
5. Deuxième quantification On applique alors une seconde fois l'algorithme de Kohonen, pour chacun des 9 clusters P_i. En considérant qu'il y a à chaque fois 4 centroïdes, chacun des 9 clusters est quantifié vectoriellement par 4 centroïdes, associés aux 4 zones de Voronoï. Par exemple, pour la région P_9 :



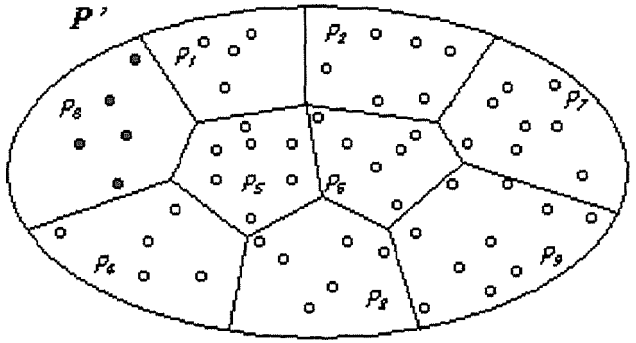
6. Calcul des fréquences Enfin, on peut effectuer le comptage des éléments de ce cluster de l'espace de déformation. P_9 étant divisé en 4 zones de Voronoï, les fréquences respectives de ces zones sont notées :

$$\begin{array}{llll} P(\bar{y}_1|\bar{x}_9) & = 2 & P(\bar{y}_3|\bar{x}_9) & = 2 \\ P(\bar{y}_2|\bar{x}_9) & = 3 & P(\bar{y}_4|\bar{x}_9) & = 3 \end{array}$$

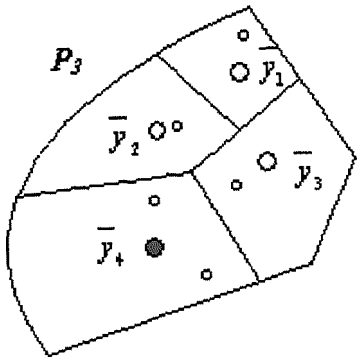
7. Choix du centroïde vainqueur En ce qui concerne la prédiction, supposons qu'on souhaite connaître l'élément suivant x_t. Supposons que, par exemple, le centroïde le plus proche de l'élément x_t est $\bar{x}_3$:



8. Cluster correspondant Puisque $\bar{x}_3$ est le centroïde le plus proche de x_t, on observe le cluster P_3 de l'espace de déformation associé à la zone de Voronoï R_3 de l'espace de base, où se trouve x_t :



9. Choix de la déformation Il reste à choisir un des centroïdes de ce cluster de l'espace de déformation, afin d'obtenir la prédiction du point x_t de l'espace de base. Un tirage aléatoire pondéré par les fréquences observées donne, par exemple, le centroïde $\bar{y}_4$:



10. Prédiction La valeur prédite est donnée par :

$$x_{t+1} = x_t + \bar{y}_4$$

3.5 Les paramètres de la méthode

Après avoir présenté et illustré la méthode, observons les moyens dont nous disposons pour exercer une influence sur son déroulement, et faire ainsi varier les résultats.

3.5.1 Les paramètres de l'algorithme de Kohonen

Bien entendu, puisqu'il est ici question d'appliquer l'algorithme de Kohonen, les paramètres de cet algorithme font partie des paramètres de la méthode.

En pratique, dans le modèle général de Kohonen, présenté dans la seconde partie du chapitre précédent, il est possible de faire varier le nombre de neurones, la forme de la carte, la valeur du taux d'apprentissage et la distance de voisinage, dès lors que nous avons fixé une fois pour toutes le mode d'initialisation des neurones. Il est également possible d'utiliser diverses fonctions décroissantes au cours du temps pour faire varier le taux d'apprentissage ainsi que la distance de voisinage. Etant donné le nombre de fois que l'algorithme est appliqué, principalement dans l'espace de déformation, nous choisissons de fixer certains paramètres, comme indiqué au point B.3.5 de l'annexe B. Par conséquent, seul le nombre de neurones et la forme de la carte doivent encore être fixés.

3.5.2 Les paramètres spécifiques de la double quantification

Le premier élément qui intervient dans la méthode est l'application du régresseur aux données. La taille de ce régresseur est donc un des paramètres de la méthode. N'oublions pas que le choix de la taille du régresseur est un problème délicat, et pas toujours optimal.

Le choix du nombre de centroïdes est également un paramètre à part entière de cette méthode. En effet, il faut choisir le nombre de centroïdes aussi bien dans l'espace de base que dans l'espace de déformation. Si le choix du nombre de neurones pour une carte de Kohonen, devant être effectué *a priori*, est déjà relativement délicat, ce paramètre est ici aussi assez difficile à déterminer : rien n'impose d'utiliser le même nombre de centroïdes dans l'espace de base et dans l'espace de déformation. De plus, le nombre de centroïdes dans chaque cluster de l'espace de déformation peut être différent, d'une région à l'autre. Il y a là une grande variété de combinaisons possibles.

Un autre paramètre est le choix du type de tirage des centroïdes dans l'espace de déformation, lors de la prédiction. On pourrait envisager d'inclure d'autres types de tirage aléatoire, fonctionnant sur un autre critère que les fréquences d'apparition. Par exemple, il est possible de prendre le centroïde qui a la plus haute fréquence d'apparition ; ou de prendre à tour de rôle chacun des centroïdes ; ou encore de ne choisir que parmi les 2, les 3, ..., les n plus fréquents, les autres étant négligeables car peu fréquents ; ...

Enfin, il faut considérer l'horizon de prédiction. Savoir si la prédiction ne porte que sur la valeur suivante ou si on essaie de connaître le comportement de la série à long terme est un autre choix, préalable à l'application de la méthode. L'horizon de prédiction est donc un paramètre à part entière. Remarquons à ce propos que la prédiction de la valeur suivante est aussi appelée prédiction à 1 pas, tandis que la prédiction à long terme est appelée prédiction à n pas, où n est l'horizon de prédiction.

Notons qu'il est possible d'envisager différentes techniques pour déterminer la moyenne des valeurs prédites, mais que, dans le cadre de ce travail, une seule façon de faire sera envisagée. Le chapitre suivant sera l'occasion d'expliquer la procédure au moyen de laquelle on calcule la moyenne des prédictions à long terme.

3.6 Variante de la méthode

3.6.1 Note préalable

La méthode présentée à la section 3.3 est la « version DICE » d'une méthode originellement écrite par M. Cottrell, E. de Bodt, P. Grégoire [3], [2] que nous allons présenter dans cette section.

Signalons que même si la version originale est celle qui va suivre, nous supposerons par la suite que la méthode « version DICE » est la méthode de base ; la méthode « version Cottrell » étant la variante. Ce petit écart avec la réalité tant scientifique qu'historique est motivé par l'ordre dans lequel ces deux versions ont été appréhendées puis implémentées. En effet, lors du stage, il n'a jamais été question que de la « version DICE », jusqu'au jour où une discussion avec Marie Cottrell nous a permis de nous rendre compte de la différence bien réelle entre ces deux versions de la méthode.

3.6.2 La méthode, seconde version

Cette variante sera rapidement présentée, les deux versions étant fort proches l'une de l'autre. L'accent sera toutefois mis sur leurs différences. Les notations utilisées recouvrent les mêmes concepts que précédemment.

3.6.2.1 Partie caractérisation

Le début de l'algorithme suit le même schéma dans cette version que dans la précédente. Au début, un régresseur est appliqué aux données, ce qui nous fait passer des données « brutes », d'un point de vue temporel, de l'ensemble de départ D aux données « non brutes » x_i. Ces dernières sont regroupées dans l'espace de base P.

Cet espace P est ensuite quantifié par une première application de l'algorithme de Kohonen, ce qui nous permet d'obtenir une série de centroïdes $\bar{x}_i$ et de créer les zones de Voronoï R_i correspondantes.

La première, et principale, différence entre les deux méthodes se situe ici, avant la deuxième application de l'algorithme de Kohonen. Au lieu de calculer les clusters, on quantifie l'ensemble de l'espace des déformations. Autrement dit, on travaille plutôt avec *un seul* ensemble contenant *toutes* les déformations y_i, et non plus sur chacun des clusters rassemblant certaines des déformations. Ces déformations sont bien entendu calculées de la même façon que précédemment.

Insistons sur le fait qu'il n'y a ici qu'un seul ensemble, contenant toutes les déformations. Cet ensemble reprend donc tout l'espace de déformation et est quantifié par l'algorithme de Kohonen qui est appliqué une seule fois.

Une fois la seconde quantification vectorielle effectuée, nous sommes en présence des centroïdes $\bar{y}_j$ situés dans l'espace de déformation, et des zones de Voronoï R_j qui leur sont associées.

Il est alors possible de calculer les fréquences d'apparition des centroïdes dans l'espace de déformation conditionnellement à la position du point de base associé à la déformation. Ces fréquences sont ici encore notées $P(\bar{y}_j \mid \bar{x}_i)$ et sont calculées par comptage, comme dans la première version.

C'est toutefois ici que se situe la deuxième différence, conséquence de la première. En effet, ce comptage n'est pas exactement le même ici, puisque les déformations sont situées dans un seul et même espace. Cette différence constitue un avantage de la deuxième version

de la méthode sur la première, ce que nous détaillerons dès la fin de la présentation de cette seconde version.

3.6.2.2 Partie prédiction

Les étapes de la partie prédiction sont les mêmes que dans la version précédente.

Lors de la présentation d'un point x_t dont on souhaite prédire le (ou les) suivant(s), on commence par chercher le centroïde $\bar{x}_i$ vainqueur dans l'espace de base.

On choisit alors un centroïde $\bar{y}_j$ de l'espace de déformation au moyen d'un tirage aléatoire, en fonction des fréquences d'apparition des centroïdes de l'espace de déformation, en sachant que le centroïde vainqueur est $\bar{x}_i$. Ce tirage s'effectue parmi *tous* les centroïdes $\bar{y}_j$ qui ont permis de quantifier l'ensemble de l'espace des déformations. Insistons une fois encore sur cette différence majeure, au risque de nous répéter : il n'y a pas de clusters.

De même que précédemment, la déformation choisie $\bar{y}_j$ est alors ajoutée au point à prédire x_t, ce qui nous donne la prédiction sur base de la même relation :

$$x_{t+1} = x_t + \bar{y}_j.$$

On peut ensuite itérer cette seconde partie pour obtenir l'horizon de prédiction souhaité. Il reste alors à répéter cette dernière partie de l'algorithme, puis à prendre la moyenne des résultats obtenus, comme dans l'autre version.

3.6.3 Avantages de la méthode seconde version

La seconde version possède l'avantage sur la « version DICE » de permettre une étude *a priori* des données. En effet, puisque la deuxième quantification porte sur l'ensemble des déformations, il est possible d'observer l'appartenance des points et des déformations aux différentes zones de Voronoï de l'espace de base et de l'espace de déformation.

Plus précisément, vu la méthode de comptage des fréquences, il est possible d'étudier si l'appartenance d'un point à une certaine zone de Voronoï, dans l'espace de base, implique l'appartenance de la déformation qui lui est associée à une zone de Voronoï bien précise de l'espace de déformation. Cette étude peut être réalisée grâce à un test χ^2 appliqué à une matrice contenant toutes les fréquences $P(\bar{y}_j \mid \bar{x}_i); 1 \leq i \leq nb1; 1 \leq j \leq nb2$, où $nb1$ et $nb2$ sont les nombres de neurones dans chacun des deux espaces, à déterminer. Cette matrice constitue une table de contingence sur laquelle on teste l'hypothèse nulle selon laquelle l'appartenance des déformations à une certaine zone de l'espace de déformation est indépendante de la zone de l'espace de base où se situe le point de base associé à la déformation. La question est alors de savoir comment déterminer $nb1$ et $nb2$.

Le choix de la valeur de $nb1$ se base sur l'observation du rapport des variances inter- et intra-classes, où une classe est en fait une zone de Voronoï. La variance intra-classe v_{intra} est définie par zone de Voronoï, comme la somme des carrés des distances entre les points de la zone de Voronoï et le centroïde correspondant, ou encore par :

$$v_{intra} = \sum_{i=1}^{n_k} (\bar{x}_k - x_i)^2$$

où :

- $\bar{x}_k$ est le centroïde de la zone de Voronoï d'indice $k, 1 \leq k \leq nb1$, $nb1$ étant le nombre total de centroïdes, choisi *a priori*,

- n_k est le nombre de points situés dans la k$^{\text{ième}}$ zone de Voronoï, et tel que $\sum_{k=1}^{nb1} n_k = N$, N étant le nombre total de points contenus dans le fichier,
- x_i, $1 \leq i \leq k$ sont les n_k points de la k$^{\text{ième}}$ zone de Voronoï.

La variance inter-classe v_{inter} est définie sur l'ensemble des points, en considérant un seul centroïde X. Ce dernier est alors le centre de la distribution, ou barycentre, et est déterminé au moyen d'une quantification vectorielle avec un unique centroïde. On calcule alors la somme des carrés des distances entre les points et ce barycentre X :

$$v_{inter} = \sum_{i=1}^{N} (X - x_i)^2.$$

Le critère que nous utilisons est le rapport r_{var} entre ces deux valeurs, calculé comme :

$$r_{var} = \frac{v_{intra}}{v_{inter} - v_{intra}}.$$

On peut alors faire varier le nombre de centroïdes $nb1$, dans l'espace de base. Disposant de ces différentes valeurs et de v_{inter}, on peut alors dessiner un graphe de la valeur du rapport r_{var} en fonction du nombre de neurones $nb1$. Ce graphe nous permet de choisir le nombre de classes à considérer dans l'espace de base, en choisissant le nombre de neurones où la tangente au graphe devient pratiquement horizontale. Puisqu'au-delà de ce point la pente de la tangente est pratiquement nulle, cela signifie que la diminution de l'erreur obtenue lorsque le nombre de neurones continue à augmenter est marginale.

Dans le cas de $nb2$, la valeur est choisie telle qu'elle est du même ordre de grandeur que $nb1$.

Notons que ce test χ^2 n'est pas possible dans la première méthode, dans la mesure où les clusters constituent une partition des déformations. Il n'est alors possible d'observer chacun des centroïdes $\bar{y}_j$ que dans le cluster auquel ils appartiennent, mais pas au niveau de l'espace des déformations dans son entièreté. Or, cette information ne nous est pas de grande utilité pour étudier les liens entre les points et les déformations, à partir du moment où on sait par construction à quel centroïde $\bar{x}_i$ de l'espace de base sont associés les centroïdes $\bar{y}_j$ de l'espace de déformation.

3.7 Concrêtement ...

La présentation des deux versions de la méthode, jusqu'ici, ne tient pas compte de bon nombre de cas qui pourraient survenir lors de l'analyse d'un problème réel. Ces cas particuliers ont dû être envisagés dans le cadre de ce mémoire.

D'un point de vue pratique, la première étape fut l'implémentation de la méthode présentée à la section 3.3. Cette version a ensuite été généralisée pour pouvoir résoudre certains problèmes.

3.7.1 Les données non scalaires

Le premier problème rencontré, c'est que la « version DICE » de l'article ne décrit que le cas de données scalaires, ou données de dimension 1. Que faire si les données temporelles sont non scalaires, autrement dit lorsqu'elles ont plusieurs composantes ? Si le problème étudié consiste à prédire une série de valeurs, et non pas une seule valeur, il faut envisager

l'application d'un régresseur sur des données de dimension supérieure à 1. De même, le reste de l'algorithme doit pouvoir être appliqué à des données de dimension supérieure à 1.

Nous avons donc modifié la méthode pour rendre possible ce cas d'utilisation. Il est assez facile de généraliser le cas des données scalaires. Il suffit de considérer une donnée non scalaire, par exemple de dimension dix, comme étant une seule donnée à dix composantes et non comme dix données distinctes à une seule composante. Par conséquent, il est possible d'appliquer un régresseur sur des données à plusieurs composantes si on les considère comme un seul bloc. On obtient de nouveau une matrice P définie comme précédemment par :

$$P = \begin{pmatrix} x(1) & \dots & x(\lambda) \\ x(2) & \dots & x(\lambda+1) \\ \vdots & \ddots & \vdots \\ x(n-\lambda) & \dots & x(n) \end{pmatrix}$$

où chaque élément $x(t); 1 \leq t \leq n$ est de la forme :

$$x(t) = (x_1(t),\ x_2(t),\ x_3(t),\ \dots,\ x_m(t)),$$

m étant la dimension des données. Cette matrice P est maintenant de dimension $[n - \lambda + 1,\ \lambda * m]$.

Le reste de l'algorithme peut lui aussi être facilement généralisé, et ce notamment parce que l'algorithme de Kohonen permet de manipuler des données de dimension quelconque. Par ailleurs, lors de la partie prédiction, on peut prédire chacune des dimensions de la donnée séparément, ce qui revient à effectuer une prédiction multi-dimension.

La méthode a donc été modifiée pour permettre cette manipulation des données non scalaires.

3.7.2 Les données creuses

A partir du moment où on considère un processus basé sur des mesures réelles, le cas particulier des données creuses, ou incomplètes, peut surgir. Nous avons déjà abordé ce problème bien réel dans le chapitre précédent. Souvenons-nous d'ailleurs que l'algorithme de Kohonen « généralisé » peut être appliqué à un ensemble comportant des données creuses.

Idéalement, si une (ou plusieurs) composante(s) de la donnée à prédire x_t est (sont) manquante(s), il faudrait pouvoir quand même prédire une valeur pour les autres composantes, non manquantes.

Résoudre cet autre cas particulier est également possible. Mais il nous faut pour cela raisonner sur l'utilité de la prédiction d'une donnée creuse. Le fait d'avoir au moins une composante manquante ne doit pas être pénalisant au point d'interdire toute prédiction. Cependant, vu la formule utilisée dans la « version DICE » de la méthode, si une composante de la donnée x_t est inconnue, quelle que soit la composante de la déformation $\bar{y}_j$ qui lui est ajoutée, qu'elle soit manquante ou non, il est impossible de savoir ce que cela peut donner.

Par exemple, considérons :

$$x_i = (x_{i1},\ x_{i2},\ \dots,\ x_{ik},\ \dots,\ x_{im})$$

et

$$\bar{y}_j = (\bar{y}_{j1},\ \bar{y}_{j2},\ \dots,\ \bar{y}_{jk},\ \dots,\ \bar{y}_{jm})$$

où m est la dimension des données et x_{ik} est une composante creuse de la donnée x_t. Lors de la prédiction, on additionne x_t et $\bar{y}_t$, et donc on effectue l'opération intermédiaire $x_{jk} + \bar{y}_{jk}$. Puisque la valeur de x_{jk} est indéterminée avant ce calcul intermédiaire, le résultat de l'addition est lui même indéterminé, quelle que soit la valeur de $\bar{y}_{jk}$.

Intuitivement, nous avons choisi de considérer les composantes manquantes comme une valeur absorbante, au même titre que l'infini, en mathématiques, est une valeur absorbante. Par conséquent, ajouter une déformation à une donnée creuse nous donne encore une donnée creuse.

La prédiction d'une donnée creuse est donc possible par cette méthode de double quantification, quelle que soit la version utilisée, dès que la méthode est modifiée pour que les composantes manquantes soient traitées en tant qu'absorbantes. L'idée est d'observer une à une les m composantes d'une donnée de dimension m. Parmi ces composantes, celles qui sont manquantes ne seront pas prédites, alors que les autres pourront faire l'objet d'une prédiction. Notons que l'idée de considérer les composantes manquantes comme une valeur absorbante, telle que l'infini, est celle qui a été utilisée au niveau de l'implémentation.

En conclusion, il est possible de manipuler des données creuses, mais il faut savoir que la prédiction d'une donnée creuse est elle-même creuse.

3.7.3 Les centroïdes morts

Comme nous utilisons l'algorithme de Kohonen, qu'en est-il du problème des centroïdes morts évoqué lors de la présentation des cartes auto-organisatrices ? Ce problème semble devoir être résolu au niveau de l'algorithme de Kohonen, mais il est en fait beaucoup plus général.

Idéalement, on souhaite pouvoir prédire, même si un point dont on souhaite connaître le suivant venait à appartenir à une région de l'espace de base pour laquelle la méthode n'avait jusqu'alors associé aucun point. Dans ce genre de régions, les centroïdes, tant dans l'espace de base que dans l'espace de déformation, ont des valeurs indéterminées, puisqu'ils sont initialisés à l'intérieur des points de la zone qu'ils quantifient. Mais comme ce centroïde est mort, sa zone de Voronoï est vide.

Ici encore, une solution au problème est envisageable. Essayons de comprendre rapidement ce que signifie le fait qu'une nouvelle donnée soit associée, lors de la prédiction, à un centroïde mort.

Comme nous l'avons expliqué dans le paragraphe 2.3.8 du chapitre précédent, un centroïde mort est un centroïde dont la zone de Voronoï est vide. Autrement dit, après convergence, aucun point n'est plus proche de ce centroïde que des autres centroïdes. Encore en d'autres termes, cela signifie que ce centroïde se trouve en dehors de la distribution des points que la carte de Kohonen quantifie. Par conséquent, un nouveau point dont on souhaite connaître le suivant et qui se retrouve associé à un centroïde mort, est en fait un point situé en dehors de la distribution jusqu'ici connue et quantifiée lors de la partie caractérisation.

Comment prédire la valeur suivante de ce point hors distribution ? Vu l'hypothèse selon laquelle le processus étudié évolue dans le futur selon une dynamique proche de son évolution passée, ce point atypique devrait être considéré comme une « erreur ». Que ce soit une erreur de mesure ou un comportement anormal du processus. Par conséquent, il ne faut pas associer le point au centroïde mort, mais au premier centroïde qui non seulement est le plus proche du point à prédire mais est également un centroïde non mort, ce qui a pour effet de ramener le point à l'intérieur de la distribution.

On décide donc d'ignorer complètement les centroïdes morts et de projeter les points hors distribution sur le centroïde non mort le plus proche. Cette modification nous permet dès lors de rendre la méthode beaucoup plus sûre face à ce phénomène qui, bien que rare, a effectivement été rencontré à ce niveau de la méthode.

3.7.4 Autres améliorations

Bien d'autres modifications ont été apportées à la méthode. Toutefois, ces derniers changements dans l'implémentation de la méthode sont plus souvent techniques que conceptuels, comme c'était le cas dans les trois derniers points ci-dessus.

Présentons toutefois brièvement les plus importantes de ces modifications techniques qui permettent de manipuler encore plus facilement la méthode.

Le premier point, même si chronologiquement ce fut un des derniers, a été l'intégration des deux versions de la méthode en un algorithme unique. Le but était de pouvoir lancer indifféremment une des deux versions à partir du même menu de départ. Il a donc fallu modifier la logique d'exécution pour pouvoir « fusionner » ces deux versions, en tenant compte de leurs spécificités.

Il nous a également semblé intéressant, dès lors qu'il est possible d'appliquer la méthode sur des données non scalaires, de pouvoir prédire chaque composante indépendamment des autres. L'algorithme a donc encore été changé pour pouvoir considérer seule à seule chacune des composantes.

Dans le même ordre d'idée, si on considère que chaque composante peut être indépendante des autres, on peut choisir de ne quantifier vectoriellement les données que sur base de certaines composantes, ce qui a également été implémenté.

Enfin, au fur et à mesure des premières simulations et des premiers résultats, une méthode de pré-traitement des données a été imaginée dans le but d'améliorer les résultats. Cette idée consiste à pondérer les données par les coefficients d'un modèle linéaire de ces mêmes données. Bien que ce pré-traitement ait été implémenté et testé, il n'a jamais été utilisé au cours des différentes utilisations de la méthode.

Chapitre 4

Sélection de modèle et critères d'erreur

4.1 Introduction

Le chapitre précédent nous a permis de présenter la méthode à la base du travail réalisé dans le cadre de ce mémoire. Dans cette présentation, il a été expliqué en détails la manière de caractériser les données d'un problème pour pouvoir prédire son comportement futur, que ce soit à court terme (prédiction à un pas) ou à long terme (simulation de l'évolution à long terme).

Bien entendu, l'évolution prédite par la méthode n'est jamais qu'une des nombreuses tendances potentiellement réalisables. Intuitivement, il existe une certaine probabilité que l'évolution future soit effectivement celle prédite. Cependant, puisque nous n'utilisons jamais qu'un modèle du processus évoluant au cours du temps, il existe une certaine erreur entre le comportement futur véritable et la tendance prédite ; un modèle n'étant jamais qu'une approximation de la réalité. Autrement dit, la probabilité que la véritable évolution soit celle qui ait été prédite n'est pas égale à 1.

Comment définir cette erreur qui survient lors de la prédiction ? Comment la mesurer ? Comment sélectionner un modèle des données, dans notre cas par la méthode de double quantification vectorielle par cartes de Kohonen, qui atteint un minimum de cette erreur ? Comment utiliser la base de données à notre disposition pour rechercher ce modèle ?

Telles sont quelques-unes des questions auxquelles nous allons maintenant répondre.

4.2 L'erreur de prédiction

Avant toute autre chose, expliquons le concept clé de ce chapitre, qui sera régulièrement utilisé dans le reste de ce mémoire.

Nous définissons l'erreur de prédiction ε_x comme étant le carré de la différence entre la vraie valeur résultant de l'évolution naturelle du processus étudié et la valeur prédite par la méthode :

$$\varepsilon_x \;=\; (x - \hat{x})^2,$$

où :

- x est la valeur suivante dans la série temporelle
- $\hat{x}$ est la prédiction, notée comme l'estimation de la vraie valeur.

Notons que l'utilisation de la notation $\hat{x}$ n'est pas un hasard. Ce que la méthode permet d'obtenir, c'est effectivement une estimation de x.

Par ailleurs, pour des raisons évidentes de manipulations de signe, le terme d'erreur est défini à la deuxième puissance. Ce choix nous permet de simplifier le raisonnement en ne devant pas tenir compte de la sur- ou sous-estimation de la valeur suivante, ce qui se serait traduit par une erreur positive ou négative, respectivement.

Dans le reste de ce travail, nous utiliserons indifféremment les termes d'erreur de prédiction ou, plus simplement, d'erreur ; sauf mention explicite.

4.3 La sélection de modèle

4.3.1 L'importance des paramètres dans la modélisation

Lors de l'introduction, au premier chapitre, nous avons expliqué le problème de la modélisation d'un processus fonction du temps, problème qui a été repris et développé dans le chapitre 3, sous le concept de régresseur. Il était alors question de savoir quelles sont les données passées de la série temporelle qui peuvent nous aider à concevoir un modèle pour nous aider, par exemple, à déterminer la valeur suivante.

En réalité, lorsqu'on utilise des réseaux de neurones, le modèle développé n'est pas uniquement fonction des données passées. Chaque type de réseau de neurones peut nous aider à obtenir une modélisation plus ou moins avantageuse du processus étudié, en fonction de leurs avantages respectifs. Une fois que la famille de réseaux de neurones est sélectionnée, chacun des paramètres du réseau peut lui aussi influencer la qualité de la modélisation.

Dans notre cas, le choix du type de réseau pour la modélisation s'est porté sur les cartes de Kohonen. Ce choix était dicté par la méthode utilisée, mais rien ne nous empêchait d'utiliser d'autres algorithmes : RBF, MLP, ou autres méthodes. Certaines de ces autres méthodes (AR, ARMA, ...) ont d'ailleurs été développées par le SAMOS pour modéliser la série Gaz de France qui a été étudiée dans le cadre de ce mémoire.

Dès lors que le choix du type de réseau a été effectué, nous pouvons compléter le modèle en déterminant les valeurs des paramètres. Pour rappel, les paramètres de la méthode de double quantification, quelle que soit sa version, sont, comme indiqué dans le chapitre précédent :

- les paramètres de l'algorithme de Kohonen : nombre de neurones, distance de voisinage, taux d'apprentissage, forme de la carte. Comme précédemment, le type d'initialisation des centroïdes et les fonctions de décroissance (distance de voisinage et taux d'apprentissage) sont fixés une fois pour toutes.
- les paramètres propres de la méthode : la taille du régresseur, le nombre de centroïdes tant dans l'espace de base que dans l'espace de déformation, le type de tirage utilisé dans l'espace de déformation, l'horizon de prédiction.

Observons de plus près chacun de ces paramètres, en gardant à l'esprit que ceux-ci doivent nous aider à définir un modèle des données aussi performant que possible, au sens de l'erreur de prédiction.

Comme expliqué dans l'annexe B, section B.3.5, certains paramètres de l'algorithme de Kohonen peuvent être fixés, à l'exception du nombre de neurones et de la forme de la carte. Ce choix de fixer les paramètres a été motivé par le nombre de fois où l'algorithme est appliqué lors de la double quantification, notamment dans la « version DICE », où une quantification avec n centroïdes est réalisée sur chaque cluster de l'espace de déformation.

Qu'en est-il des autres paramètres de la méthode de double quantification ? Ici encore, le nombre de neurones dans chaque espace est sans doute l'aspect le plus important. En effet, supposons qu'on « connaisse » relativement bien la série temporelle. Cette connaissance de la série peut se traduire notamment par le régresseur, déjà déterminé et appliqué aux données. Plus simplement, on peut aussi choisir de ne pas appliquer de régresseur. L'horizon de prédiction n'est rien d'autre qu'une itération de la prédiction à un pas. Qui plus est, cette prédiction fait partie de la seconde partie de la méthode, qui succède à la caractérisation où on développe un modèle des données. Le type de tirage des centroïdes dans l'espace de déformation, également dans la deuxième partie de la méthode, est quant à lui une manipulation technique qui permet de jouer sur la qualité de la prédiction, mais n'influence pas la modélisation de la série.

Par conséquent, il ne nous faut plus définir que le nombre de neurones dans chacun des deux espaces. Ce sont ces deux nombres qui vont constituer la caractéristique principale d'une des modélisations possibles. Notre but est, par conséquent, de pouvoir déterminer ces nombres.

Malheureusement, la méthode que nous utilisons ne nous dit pas combien de centroïdes il nous faut utiliser. Elle n'indique pas non plus comment choisir ce nombre en fonction de la complexité de la série. Pire, on ne sait même pas si les nombres de neurones respectivement présents dans chacun des deux espaces sont liés entre eux ou pas !

4.3.2 Le choix d'un modèle

S'agit-il là d'une limite à l'utilisation de la méthode ? Oui et non. Oui, parce que tout ce que nous pouvons faire, c'est tester les différentes possibilités les unes après les autres, alors qu'il aurait été plus intéressant de pouvoir déterminer à l'avance un nombre approximatif de centroïdes dans chaque espace, simplement pour être plus performant d'un point de vue temps de calcul. Non, dans la mesure où rien ne nous empêche, justement, de considérer ces différentes possibilités les unes après les autres. De plus, ce test ne doit pas nécessairement être exhaustif, dans un premier temps en tout cas. Enfin, il existe malgré tout une borne supérieure, visible, lorsque le réseau rencontre le problème de l'overfiting, ou sur-apprentissage.

Ce phénomène de sur-apprentissage est un problème typique en réseaux de neurones artificiels que nous avons abordé dans le chapitre 2, lors de la présentation générale des réseaux de neurones artificiels. Brièvement, nous pouvons rappeler que ce problème survient lorsque la complexité du modèle est telle que le nombre de points connus devient insuffisant pour réaliser l'apprentissage en un nombre limité de présentations de la base de données. Le réseau a alors tendance à ne pas pouvoir caractériser correctement l'ensemble de points qui lui sont présentés.

Retenons que, pour modéliser une série temporelle, la méthode que nous proposons d'employer nous oblige à tester les différentes possibilités. Ces différentes modélisations sont bornées supérieurement par le phénomène de sur-apprentissage. Dès lors que nous décidons d'examiner les différents modèles, comment déterminer qu'un de ces modèles est meilleur qu'un autre ? Bien entendu, nous allons nous servir de l'erreur de prédiction pour comparer les différents modèles.

4.4 La cross-validation

Le problème pour ainsi dire récurrent, en apprentissage par des techniques artificielles, c'est le nombre de données. En pratique, ce nombre de données est généralement assez réduit ; ou en tout cas il est rarement suffisant pour modéliser l'entièreté d'un problème donné.

En utilisant une propriété intéressante des réseaux de neurones artificiels, on peut toutefois dépasser cette contrainte. En effet, lors de l'apprentissage, l'ordre dans lequel les données sont présentées au réseau de neurones est sans importance.

4.4.1 Notions de cross-validation

La cross-validation est une technique particulière régulièrement utilisée dans différentes disciplines, notamment les réseaux de neurones artificiels, mais également en statistique. Dans notre cas, cette technique nous permet d'une part de réaliser un apprentissage d'un réseau de neurones, ce qui revient à modéliser un processus, et d'autre part de tester la qualité du modèle développé ; le tout avec un fichier unique contenant un nombre fini de données.

4.4.1.1 Idée intuitive

Le but à la base de la cross-validation est de pouvoir tester les capacités d'un réseau de neurones artificiels après apprentissage.

Supposons que, lors de l'étude d'un certain problème, nous disposions de 1000 données d'apprentissage. Ces 1000 données sont utilisées pour modéliser le problème par un réseau de neurones artificiels. En outre, on voudrait pouvoir tester ce réseau sur des données « nouvelles ». Ces autres données doivent toutefois être cohérentes avec le problème étudié. Par « cohérentes », nous voulons signifier que les nouvelles données doivent être issues du même processus, dont nous ne connaissons que 1000 valeurs mesurées par le passé.

Le plus simple, c'est évidemment de séparer les données de départ en deux ensembles. D'un côté, nous constituons une base de données avec des éléments qui ne servent que lors de l'apprentissage. De l'autre, on conserve des données qui ne seront utilisées qu'après l'apprentissage, et seront dès lors nouvelles. On peut de la sorte tester la qualité de l'apprentissage en observant les capacités de généralisation des « connaissances » sur un nouvel ensemble de données. La cross-validation utilise cette idée simple avec une petite nuance que nous préciserons ultérieurement.

4.4.1.2 Définition et construction des ensembles d'apprentissage et de validation

Le terme d'ensemble d'apprentissage désigne l'ensemble des données qui sont utilisées pour modéliser un problème par un réseau de neurones artificiels. Les données de cet ensemble sont présentées successivement au réseau de neurones, un certain nombre de fois.

L'ensemble de validation est constitué des données qui servent lors du test des capacités du réseau, après apprentissage. Ces données sont présentées une seule fois au réseau. La qualité de l'apprentissage peut être mesurée en comparant les réponses calculées par le réseau et les réponses réelles correspondant aux points de l'ensemble de validation. Ces comparaisons sont reprises dans un terme d'erreur lié au modèle développé.

Dans notre cas, après apprentissage, on teste le modèle sur l'ensemble de validation, en mesurant l'erreur de prédiction réalisée sur cet ensemble de points. Nous définissons donc un critère d'erreur plus général, le critère d'erreur quadratique, reprenant la somme des erreurs de prédiction réalisée sur chacune des données de l'ensemble de validation :

$$E = \frac{\sum_{x \in E_V} (x - \hat{x})^2}{N_{E_V}} = \frac{\sum_{x \in E_V} \varepsilon_x}{N_{E_V}}$$

où N_{E_V} est la notation qui désigne le nombre d'éléments dans l'ensemble de validation E_V. Ce critère nous permet de mesurer la qualité de l'apprentissage d'un modèle pour l'ensemble de validation correspondant.

Par la suite, nous utiliserons les notations E_A et E_V pour désigner les ensembles d'apprentissage et de validation, respectivement.

Si l'idée de la cross-validation est extrêmement simple, son utilisation pratique est relativement délicate. Il faut pouvoir déterminer la part des données qui seront utilisées lors de chacune des deux étapes. Autrement dit, si on coupe le fichier de départ en deux parties distinctes, où faut-il couper ? Nous sommes ici face à un dilemme.

En effet, si une majorité des données est utilisée lors de la première étape, on favorise l'apprentissage aux dépens de la validation. Vu qu'un réseau de neurones apprend « par l'exemple », en lui procurant un plus grand nombre de ces exemples, on lui permet de pouvoir plus facilement généraliser ses « connaissances », plus nombreuses, aux nouvelles données qui lui sont présentées. Par ailleurs, comme le nombre de données est limité, on dispose de moins de données pour tester les capacités du réseau. On sur-estime donc la qualité de l'apprentissage.

A l'opposé, si on garde beaucoup de données pour le test du réseau, on dispose de peu de ces données pour l'apprentissage. Le réseau de neurones réalisera donc plus d'erreurs lors de la validation, dès lors qu'il sera confronté à plus d'exemples nouveaux, la probabilité de faire une erreur augmentant avec ce nombre de nouveaux exemples. On sous-estime cette fois la qualité de l'apprentissage.

Reprenons l'exemple ci-dessus, avec les 1000 données. Si nous décidons de prendre 100 données pour l'apprentissage et 900 données pour la validation, il est certain que les résultats seront mauvais. Le réseau est dans ce cas-ci confronté à un nombre trop important d'exemples nouveaux qui sont pour lui source de mauvaise réponse.

Dans l'autre sens, si on utilise 900 données pour l'apprentissage et 100 pour la validation, la partie test sera défavorisée. On peut estimer que le réseau ne se trompera pas souvent, mais le test ne sera pas complètement représentatif : les données de l'ensemble de validation ne constituent qu'une partie réduite des nouveaux exemples qui peuvent survenir.

Des proportions de 10% et 90% viennent d'être mentionnées. De telles répartitions des données entre les deux ensembles sont extrêmes. En pratique, la grande majorité des expérimentations sont menées avec des ensembles d'apprentissage contenant de 60 à 85% des données, rarement plus et très rarement moins. Les 15 à 40% des données restantes servent bien évidemment lors de la validation du modèle.

4.5 Cross-validation et simulation Monte-Carlo

La technique, simple, de cross-validation présentée ci-dessus nous permet, grâce à une manipulation préalable des données, de non seulement développer un modèle mais encore

de le tester. Il faut toutefois constater qu'à elle seule, la cross-validation telle que présentée jusqu'ici ne nous aide que partiellement à déterminer le modèle réalisant l'erreur de prédiction minimale. En effet, s'il est possible de tester un modèle, il n'est pas possible de le comparer avec d'autres. Pour sélectionner le meilleur modèle possible, on se sert d'une autre technique, également bien connue, notamment en statistique.

Supposons que nous ayons développé une série de modèles par la méthode de double quantification. Par exemple, nous avons testé les modèles avec ficelle, comportant de 1 à 20 neurones tant dans l'espace de base que dans l'espace de déformation, soit en tout 400 modèles différents. Notons que la forme de la carte dans chaque espace importe peu pour l'explication qui va suivre. Après recherche du minimum de l'erreur, on constate par exemple que celui-ci est rencontré pour le modèle comportant 8 neurones dans l'espace de base et 5 dans l'espace de déformation.

Rappelons maintenant la propriété selon laquelle l'ordre dans lequel les données sont présentées au réseau de neurones n'a pas d'importance, dans le cadre de l'apprentissage. Il est légitime de se demander si le modèle réalisant le minimum de l'erreur de prédiction pour l'ensemble de validation qui vient d'être employé, à savoir 8 et 5 neurones, est également le meilleur si on venait à redistribuer les données entre les ensembles d'apprentissage et de validation. Cette redistribution peut mener à une convergence différente de la carte et peut demander de tester des exemples qui faisaient partie des « connaissances » de la carte, auparavant. Cette redistribution des données est la nuance supplémentaire de la technique de cross-validation qui devait être précisée. L'utilité de cette redistribution est de pouvoir tester plusieurs fois un même modèle sur des ensembles d'apprentissage et de validation différents mais tous issus du même ensemble de données. On résume ensuite les différents résultats obtenus lors des différentes validations par le calcul de la moyenne, selon la procédure Monte-Carlo.

La procédure Monte-Carlo est une technique souvent utilisée dans l'étude de phénomènes statistiques. Le principe général de ce mode de simulation consiste à répéter un certain nombre de fois une expérience, puis de calculer la moyenne des résultats obtenus. Seuls ces résultats moyens sont pris en compte. Cette technique a pour but d'éviter tout biais pouvant survenir lors d'une observation unique d'un phénomène. En effet, s'il est possible de constater une certaine tendance parmi un ensemble d'évènements, c'est que, statistiquement, cette tendance fait partie du processus. Il est statistiquement incorrect de tirer ce genre de conclusion sur base d'une seule observation.

Dans notre cas, la procédure Monte-Carlo est indispensable, puisque les neurones sont initialisés avec les valeurs de certaines données choisies au hasard. Un autre apprentissage, qui plus est après redistribution des données, ne donnera pas forcément une même convergence de la carte. Pour reprendre l'exemple ci-dessus, si on constate, en répétant plusieurs fois l'apprentissage, que l'erreur de prédiction est en moyenne inférieure pour le modèle avec 8 et 5 neurones à celle de tous les autres modèles, on peut effectivement tirer la conclusion que ce modèle est celui qui est statistiquement le plus adapté à la série qui est étudiée.

La procédure Monte-Carlo est dès lors utilisée pour répéter un certain nombre de fois l'analyse des différents modèles. La cross-validation est intégrée dans cette procédure de simulation puisqu'elle permet une découpe du fichier de départ en deux fichiers distincts, avec redistribution des données entre les deux fichiers qui seront utilisés par tous les modèles testés lors de cette itération de la procédure Monte-Carlo. Pour effectivement redistribuer les données, il suffit d'appliquer une permutation du contenu du fichier de départ avant de scinder le fichier entre les parties apprentissage et validation.

Par la suite, une itération de la procédure Monte-Carlo sera appelée une itération de

cross-validation. On peut expliquer cet abus de langage par le fait que lors d'une itération de la procédure Monte-Carlo, on applique une cross-validation qui permet de couper le fichier en ensembles d'apprentissage et de validation. Ces deux ensembles sont les mêmes pour tous les modèles développés lors de cette itération de la procédure Monte-Carlo, la redistribution des données ayant lieu lors de l'itération suivante.

4.6 Concrètement ...

4.6.1 Définition des critères d'erreur

4.6.1.1 Première définition

Nous avons défini l'erreur de prédiction entre une valeur calculée par la méthode et la vraie valeur suivante dans la série. Cette définition nous a permis de définir le critère d'erreur quadratique pour tester les différents modèles. Ce critère est lié à l'ensemble de validation.

Le programme développé dans le cadre de ce mémoire utilise effectivement ce premier critère d'erreur quadratique, mais il est également défini par d'autres critères, tous liés au premier.

Nous définissons le critère d'erreur normalisée par l'erreur moyenne, noté E_N, par :

$$E_N = \frac{\frac{\sum_{x \in E_V} (x - \hat{x})^2}{N_{E_V}}}{\frac{\sum_{x \in F} (x - \bar{x})^2}{N_F}}$$

où :

- x est la vraie valeur dans la série temporelle,
- $\hat{x}$ est la prédiction,
- $\bar{x}$ est la moyenne des données du fichier de départ F,
- E_V est l'ensemble de validation,
- $F = E_V \cup E_A$ est le fichier de départ contenant tous les points de la base de données,
- N_{E_V} représente le nombre de données de E_V,
- N_F représente le nombre de données de F.

Notons que cette appellation d'erreur quadratique normalisée par l'erreur moyenne est un abus de langage. On considère en réalité l'erreur quadratique normalisée par l'erreur obtenue lorsque la prédiction est toujours la valeur moyenne. Pour des raisons de simplicité, cet abus de langage sera pourtant utilisé par la suite.

En modifiant le dénominateur, on peut également définir le critère de l'erreur quadratique normalisé par l'erreur des différences E_D par :

$$E_D = \frac{\frac{\sum_{x \in E_V} (x - \hat{x})^2}{N_{E_V}}}{\frac{\sum_{x \in F} (x_{t+1} - x_t)^2}{N_F}}$$

où :
- x_t est une valeur de la série temporelle,
- x_{t+1} est la valeur suivante de x_t,

les autres notations ayant toujours la même signification.

Un troisième et dernier critère d'erreur auquel nous avons également pensé, c'est le critère d'erreur quadratique normalisé par l'erreur d'un modèle linéaire. Pour rappel, un modèle linéaire, en réseaux de neurones artificiels, est un réseau de type adaline, dont la matrice des poids w peut être rapidement calculée par la méthode de la matrice pseudo-inverse :

$$w = (X^T X)^{-1} X^T Y$$

où
- X est la matrice des entrées du réseau adaline,
- X^T est la transposée de la matrice X, au sens mathématique habituel,
- $(.)^{-1}$ est l'opération inverse sur les matrices, au sens mathématique habituel,
- Y est la matrice des sorties attendues pour la matrice des entrées correspondantes X.

Une fois que la matrice des poids w est connue, on peut facilement calculer les sorties $\hat{Y}$ calculées par le modèle linéaire par le produit :

$$\hat{Y} = X\, w.$$

Nous pouvons maintenant définir le dernier critère d'erreur quadratique normalisée par l'erreur du modèle linéaire E_L :

$$E_L = \frac{\dfrac{\sum_{x \in E_V} (x - \hat{x})^2}{N_{E_V}}}{\dfrac{\sum_{y \in Y;\ \hat{y} \in \hat{Y}} (y - \hat{y})^2}{N_F}}$$

où :
- y est la sortie attendue,
- $\hat{y}$ est la sortie calculée par le réseau de type adaline.
- Y est la matrice des sorties attendues pour toutes les entrées X,
- $\hat{Y}$ est la matrice des sorties calculées pour toutes les entrées X
- X et Y ont le même nombre d'éléments que F.

4.6.1.2 Critique

Les critères définis ci-dessus ne tiennent pas compte d'une chose importante : l'erreur réalisée par un réseau de neurones est fonction de l'apprentissage qu'il a pu réaliser. Dès lors, normaliser par un terme calculé sur l'ensemble F de toutes les données n'est pas correct. La valeur au numérateur est calculée sur un ensemble de points plus petit que celle se retrouvant au dénominateur.

Les critères d'erreur définis ci-dessus sous-estiment donc toujours l'erreur réelle. Celle-ci est en effet toujours divisée par un terme trop grand, où il est tenu compte de plus de données et donc d'une erreur potentiellement plus grande.

4.6.1.3 Deuxième définition

Une correction s'impose vu la sous-estimation de l'erreur réelle. Il est préférable de calculer celle-ci sur base d'un même ensemble de données, pour pouvoir comparer des choses comparables. Or, puisque nous utilisons la cross-validation, l'erreur de prédiction est mesurée sur l'ensemble de validation E_V. En modifiant les définitions ci-dessus, nous sommes donc en mesure de définir des critères qui ne sous-estiment plus l'erreur de prédiction.

En reprenant les notations E_A et E_V qui désignent respectivement les ensembles d'apprentissage et de validation, on peut définir le nouveau critère d'erreur quadratique normalisée par l'erreur moyenne E_N selon la relation :

$$E_N = \frac{\dfrac{\sum_{x \in E_V} (x - \hat{x})^2}{N_{EV}}}{\dfrac{\sum_{x \in E_V} (x - \bar{x})^2}{N_{EV}}}$$

ou plus simplement :

$$E_N = \frac{\sum_{x \in E_V} (x - \hat{x})^2}{\sum_{x \in E_V} (x - \bar{x})^2}.$$

De même, le critère d'erreur quadratique normalisée par l'erreur des différences E_D devient :

$$E_D = \frac{\sum_{x \in E_V} (x - \hat{x})^2}{\sum_{x \in E_V} (x_{t+1} - x_t)^2}.$$

Enfin, le critère d'erreur quadratique normalisée par l'erreur d'un modèle linéaire E_L s'écrit :

$$E_L = \frac{\sum_{x \in E_V} (x - \hat{x})^2}{\sum_{y \in Y_V;\ \hat{y} \in \hat{Y}} (y - \hat{y})^2}.$$

où Y_V, X_V et $\hat{Y}$ ont le même nombre d'éléments que E_V.

Dans ce dernier cas, le calcul du modèle linéaire de type adaline a dû être modifié pour tenir compte de la distinction maintenant effectuée entre les ensembles d'apprentissage et de validation. Par conséquent, si

- X_A est la matrice des entrées de l'ensemble d'apprentissage,
- X_A^T est, ici encore, la transposée de la matrice X_A,
- Y_A est la matrice des sorties pour l'ensemble des entrées correspondantes,

on peut calculer la matrice des poids w sur base de la relation :

$$w = (X_A^T X_A)^{-1} X_A^T Y_A$$

Connaissant cette nouvelle matrice des poids, on calcule alors l'erreur obtenue pour les entrées de l'ensemble de validation X_V selon :

$$\hat{Y} = X_V w$$

Le terme $\hat{Y}$ final est celui utilisé dans la définition du critère E_L.

4.6.1.4 Interprétation

A quoi peuvent servir ces critères ?

Bien sûr, la première utilité d'un critère d'erreur est de pouvoir évaluer un modèle. Cette évaluation nous permet de comparer les modèles entre eux afin de sélectionner celui réalisant le minimum d'erreur, dans notre cas d'erreur de prédiction, pour le problème considéré.

Mais, dans ce cas, le premier critère d'erreur quadratique était largement suffisant. Pourquoi avoir défini d'autres critères, qui plus est avec une restriction sur le terme au dénominateur pour pouvoir normaliser le critère de base ?

L'intérêt des trois critères supplémentaires est l'interprétation que l'on peut faire de la valeur numérique obtenue après normalisation. Examinons cette interprétation pour chacun des trois cas :

- Critère E_N : vu sa définition, lorsque la valeur calculée après normalisation est proche de 1, cela signifie que le modèle ne fait pas mieux que de prédire la moyenne des données.
- Critère E_D : de par sa définition, lorsque E_D tend vers 1, cela veut dire que le modèle ne fait pas mieux que donner la dernière valeur connue comme prédiction.
- Critère E_L : ce critère nous permet de voir dans quelle mesure le modèle calculé au moyen de la double quantification fait mieux ou moins bien qu'un modèle linéaire calculé sur le même ensemble de données. Lorsque le rapport est supérieur à 1, cela signifie que le modèle linéaire est plus performant, lorsqu'il est inférieur à 1, le modèle par double quantification est meilleur que le modèle linéaire.

4.6.2 Concrètement ...

Au niveau informatique, pour cette partie du travail, il nous a fallu modifier le code de la méthode pour pouvoir simuler un grand nombre de fois les mêmes expériences. Cette boucle sur le nombre d'itérations de cross-validation a donc été ajoutée à l'extérieur de la partie de programmation qui implémente la méthode de double quantification vectorielle.

Cette dernière a également été modifiée pour pouvoir examiner les différents modèles. Deux boucles sur le nombre de centroïdes dans chacun des deux espaces ont ainsi été ajoutées, pour permettre cet examen.

Il a également fallu inclure la manipulation des données effectuée lors de la cross-validation : découpe du fichier en deux ensembles, possibilité de permuter les données, ...

Ensuite, les critères d'erreur ont été inclus dans le programme. Les premiers critères définis ont d'abord été utilisés. Les résultats obtenus lors des premières simulations étaient étonnamment bons. Après réflexion sur la qualité, plutôt inattendue, de ces résultats, nous sommes arrivés à la conclusion que l'erreur était en fait sous-estimée, parce que pondérée par un facteur trop important. C'est pourquoi nous avons deux définitions des critères, la deuxième ayant finalement été retenue, implémentée et utilisée. Ce sont ces critères qui vont nous permettre d'évaluer les performances des modèles dans le reste de ce travail.

Enfin, nous avons implémenté une procédure qui nous permettait de prendre la moyenne des simulations et de résumer l'information obtenue en prenant les modèles les meilleurs parmi tous ceux testés.

Chapitre 5

Application de la méthode : la série Santa Fe A

5.1 Introduction

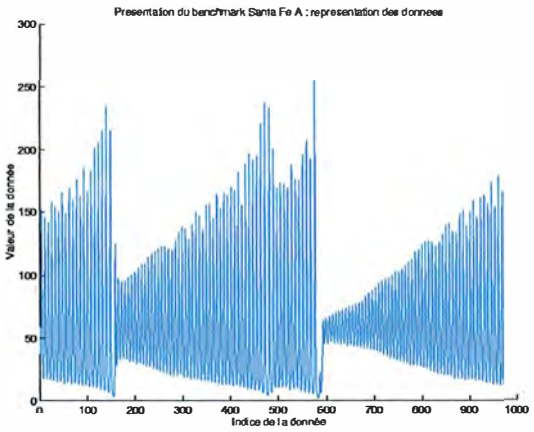
Dans le chapitre 3, nous avons expliqué la méthode de double quantification, dans ses deux versions. Dans le chapitre 4, nous avons expliqué la technique classique de cross-validation, la procédure générale Monte-Carlo et décrit des critères pour la mesure de l'erreur de prédiction.

Avant de nous intéresser au fichier de Gaz de France, but principal de ce travail, il nous a semblé intéressant de prendre le temps de tester l'implémentation de la méthode. Pour ce faire, nous avons choisi d'utiliser un benchmark classique, en prédiction de séries temporelles, à savoir la série Santa Fe A. Notre but est de valider la partie programmation en testant toutes les possibilités de la méthode : étude des données et prédiction, que ce soit à un pas ou à long terme.

5.2 Présentation de la série temporelle

La série Santa Fe A, la série laser, est une série bien connue [9] dans le cadre de la prédiction de données temporelles. Les fichiers contenant les données peuvent être obtenus à l'adresse Internet [14] ou [15]. La première référence donne également une description de la série. Deux fichiers sont disponibles. Traditionnellement, le premier fichier, comptant 1 000 données, est utilisé comme fichier d'apprentissage, alors que le second fichier, comportant 10 000 données, constitue la série complète.

Voici une représentation graphique de ces 1 000 premières données :



Notons que ces données sont scalaires. Par ailleurs, il est possible, à première vue, de noter une double tendance sur base de ce graphe : un premier comportement relativement périodique et un autre beaucoup plus chaotique. Lié à ce comportement chaotique, on observe deux fortes cassures dans l'allure générale du graphe.

5.3 Présentation générale des simulations

5.3.1 Le fichier utilisé

Vu le comportement de la série, il est clair que se baser sur la dernière valeur connue pour prédire la suivante ne sera pas suffisant. Par conséquent, nous allons appliquer un régresseur à la série temporelle.

Profitant de l'expérience de M. Amaury Lendasse et de sa bonne connaissance de la série Santa Fe A, nous connaissons le régresseur optimal. Celui-ci est de taille 6 et est composé des données en $t-6$, $t-5$, $t-3$, $t-2$, $t-1$ et t. Dans notre cas, le fichier reçu comporte 971 données de dimension 6. Il n'y a pas de données creuses.

5.3.2 Les simulations

Dans un premier temps, nous allons rechercher un modèle optimal des données. Ce modèle sera ensuite utilisé pour prédire la série à un certain terme dans le temps.

Cette méthodologie, très générale, se base sur les possibilités qu'offre la méthode de double quantification. En effet, il est possible de sélectionner un modèle puisque l'algorithme développé comporte une boucle de type Monte-Carlo, couplée avec la technique de cross-validation (découpe du fichier en ensembles d'apprentissage et de validation). Pour trouver un modèle optimal, en moyenne, nous mesurons l'erreur de prédiction à un pas.

Dans un deuxième temps, un nouvel apprentissage sera effectué sur le modèle retenu, avec la totalité des données. Sur base de cet apprentissage, on itère la prédiction à un pas pour obtenir l'évolution à long terme de la série temporelle. Ici encore, la procédure Monte-Carlo est utilisée pour répéter les simulations à long terme et obtenir une prédiction moyenne, qui sera l'évolution statistiquement la plus probable de la série temporelle.

5.4 Simulations et résultats

Etant donné que nous avons développé deux versions de la méthode, toutes les simulations seront faites en double, afin de comparer les performances respectives de chaque version et d'observer les différences de comportement, s'il y en a.

Pour toutes les simulations qui vont suivre, 100 itérations de cross-validation ont été effectuées, sauf mention explicite. La proportion du fichier utilisée pour l'apprentissage est de 66%, le tiers restant étant bien évidemment utilisé lors de la validation. Les 4 critères d'erreur définis ont été observés.

Le calcul du régresseur n'a pas été fait au niveau de la méthode : le fichier fourni contenait déjà des données sur lesquelles le régresseur avait été appliqué.

5.4.1 Etude des données

La méthode développée dans le cadre de ce mémoire offre la possibilité de réaliser une étude des données. Cette étude nous permet d'accepter ou de rejeter, via un test χ^2 sur une table de contingence, l'hypothèse nulle selon laquelle il y a indépendance entre l'appartenance d'un point de base à une zone de Voronoï de l'espace de base et l'appartenance de sa déformation à une zone de Voronoï en déformation. Autrement dit, on essaie de savoir s'il existe ou non un lien entre la position d'un point de base et la déformation qui est susceptible de lui être ajoutée. Encore en d'autres termes, on veut tester si le fait d'appartenir à une certaine zone de Voronoï dans l'espace de base implique qu'un certain centroïde en déformation sera plus particulièrement choisi ou pas. Par la suite, les zones de Voronoï seront aussi appelées « classes », selon la terminologie utilisée en statistiques.

5.4.1.1 Rappel théorique : le test χ^2

Ce rappel est basé sur [12], [4].

Brièvement, un test χ^2 sur une table de contingence offre permet de tester l'indépendance d'éléments classés selon deux critères. Dans ce cadre, on note généralement p_{ij} la probabilité qu'un élément appartienne à la fois à la classe i et à la classe j.

Considèrons :

- $p_{i.} = \sum_{j=1}^{s} p_{ij}$ où s est le nombre de classes selon le second critère, indicées j, et
- $p_{.j} = \sum_{i=1}^{r} p_{ij}$ où r est le nombre de classes selon le premier critère, indicées i,

alors on a indépendance lorsque $p_{ij} = p_{i.} * p_{.j}$.

Au sein de l'échantillon, on observe n_{ij} individus appartenant aux classes i et j. On peut alors définir les notations $n_{i.}$ et $n_{.j}$:

- $n_{i.} = \sum_{j=1}^{s} n_{ij}$,
- $n_{.j} = \sum_{i=1}^{r} n_{ij}$.

Le nombre déterminé par la formule :

$$\sum_{i=1}^{r} \sum_{j=1}^{s} \frac{(n_{ij} - np_{i.}p_{.j})^2}{np_{i.}p_{.j}}$$

est alors une valeur prise par une variable χ^2 à $rs-1$ degré de liberté, sous l'hypothèse d'indépendance.

Or les probabilités $p_{i.}$ et $p_{.j}$ sont inconnues. Elles sont donc approximées par $p_{i.} = n_{i.}/n$ et $p_{.j} = n_{.j}/n$, où n est le nombre d'individus dans l'échantillon. En remplaçant les probabilités $p_{i.}$ et $p_{.j}$ dans la formule ci-dessus et en développant les calculs, on obtient un moyen de calculer la distance nD_{χ^2} :

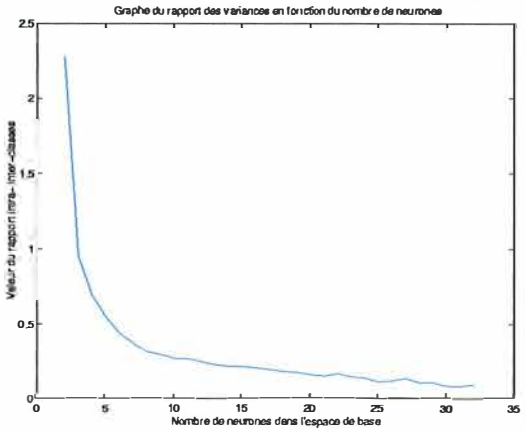
$$nD_{\chi^2} = n(\sum_{i=1}^{r}\sum_{j=1}^{s}\frac{n_{ij}^2}{n_{i.}n_{.j}} - 1)$$

On estime ici $r+s-2$ paramètres, et donc la valeur obtenue par le calcul de nD_{χ^2} est la valeur prise par une variable χ^2 à $rs-1-r-s+2$, soit $(r-1)(s-1)$ degrés de liberté.

En observant les tables du χ^2, lorsque la valeur de nD_{χ^2} est inférieure à la valeur reprise dans la table et correspondante à $(r-1)(s-1)$ degrés de liberté, l'hypothèse d'indépendance est acceptée, dans le cas contraire elle est rejetée.

5.4.1.2 Application

Pour étudier les données, nous appliquons la méthode « version Cottrell », en utilisant des ficelles de Kohonen, pouvant compter de 1 à 32 neurones, et ce dans chacun des deux espaces. Le graphe du rapport des variances inter- et intra-classes nous permet de choisir le nombre de centroïdes à considérer dans les deux espaces :



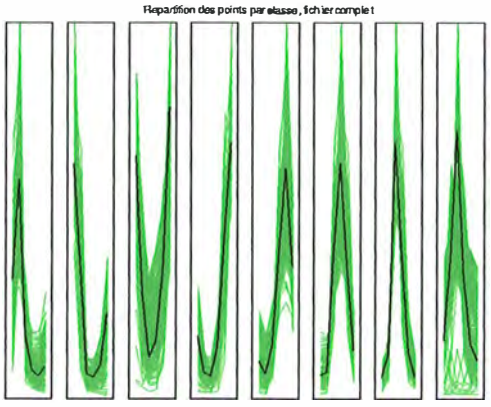
Nous choisissons ici de considérer 8 centroïdes dans l'espace de base, étant donné qu'au-delà de ce nombre, la variation du rapport des variances, conséquente à l'ajout de nouveaux neurones le long de la ficelle, est à ce moment marginale. Nous avons donc 8 zones de Voronoï, et donc 8 classes. Autrement dit, on choisit ici $r = 8$. Dans l'espace de déformation, on choisi de considérer 10 neurones, soit $s = 10$.

Les fréquences d'apparition, calculées sur base d'un comptage et précédemment notées $P(\bar{y}_j \dim \bar{x}_i)$, sont reprises dans un tableau, qui constitue une table de contingence puisque $P(\bar{y}_j \dim \bar{x}_i) = n_{ij}$. Voici les valeurs obtenues :

Classe Base / Déformation	1	2	3	4	5	6	7	8
1	0.000	0.883	0.086	0.000	0.000	0.000	0.000	0.010
2	0.000	0.074	0.728	0.087	0.000	0.000	0.000	0.005
3	0.797	0.043	0.006	0.000	0.000	0.000	0.000	0.010
4	0.196	0.000	0.000	0.000	0.000	0.000	0.314	0.518
5	0.000	0.000	0.000	0.000	0.000	0.000	0.357	0.124
6	0.000	0.000	0.019	0.087	0.642	0.000	0.000	0.000
7	0.007	0.000	0.062	0.000	0.058	0.557	0.329	0.332
8	0.000	0.000	0.000	0.000	0.008	0.443	0.000	0.000
9	0.000	0.000	0.000	0.000	0.292	0.000	0.000	0.000
10	0.000	0.000	0.099	0.826	0.000	0.000	0.000	0.000

Le calcul de la distance nD_{χ^2} nous donne 4158,1. Or, dans les tables du χ^2, on observe que pour $(r-1)(s-1)$ degrés de liberté, soit 63 dans notre cas, la valeur de rejet pour une probabilité à 95% est de 82,5. Nous sommes très nettement au-dessus, ce qui nous permet de rejeter l'hypothèse selon laquelle la distribution des déformations serait indépendante de la classe à laquelle la donnée de base appartient. Autrement dit, il existe un lien entre les déformations et les données de base auxquelles elles sont associées.

Nous pouvons en outre observer les zones de Voronoï de l'espace de base, afin de tester d'une part la qualité de la convergence et d'avoir d'autre part une idée du profil des centroïdes et des données, ainsi que de la répartition de ces dernières entre les différentes zones de Voronoï :



On constate que la convergence dans l'espace de base est de bonne qualité, les centroïdes se trouvant à chaque fois au centre des données reprises dans leur zone de Voronoï respective. On observe de plus une transition fluide entre les profils des différents centroïdes.

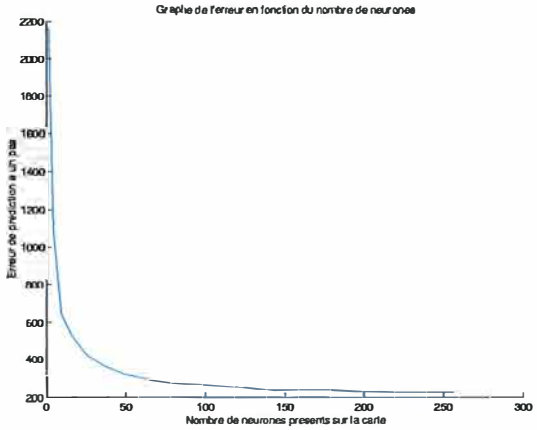
5.4.2 Le cas des cartes carrées

La méthode de double quantification laisse une série de degrés de liberté que l'utilisateur peut fixer. Parmi ces paramètres, il y a bien sûr le nombre de neurones présents sur la carte de Kohonen, mais également la forme de la carte. Les premières simulations modélisent les données par des cartes carrées, tant dans l'espace de base que dans l'espace de déformation.

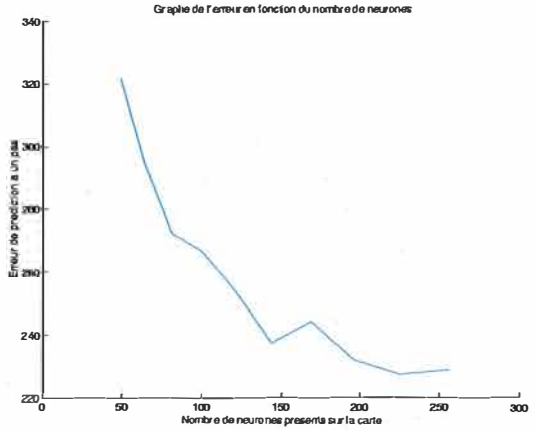
Comme nous l'avons expliqué dans le chapitre précédent, la méthode ne nous renseigne pas sur le nombre de neurones dans chaque espace. Il nous faut donc passer en revue les différentes possibilités.

5.4.2.1 La méthode « version DICE »

Une première simulation a été lancée, avec un nombre de neurones variant de 1 à 16 sur le côté de la carte, dans l'espace de base, soit de 1 à 256 neurones. Dans l'espace de déformation, chaque cluster comportait lui aussi des cartes carrées avec de 1 à 16 neurones sur le côté. Cette simulation nous permet donc d'examiner 256 modèles différents. Observons le graphe de l'erreur quadratique lors de la prédiction à un pas, en fonction du nombre de neurones présents sur la carte située dans l'espace de base :



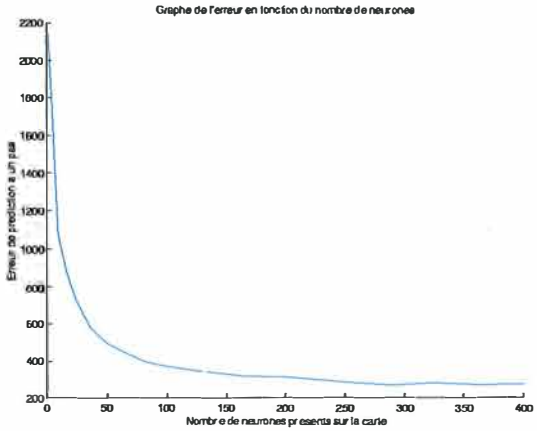
Comme on pouvait s'y attendre, l'erreur quadratique diminue au fur et à mesure que le nombre de neurones augmente. Si on agrandit maintenant la courbe, en la tronquant de ses premiers éléments :



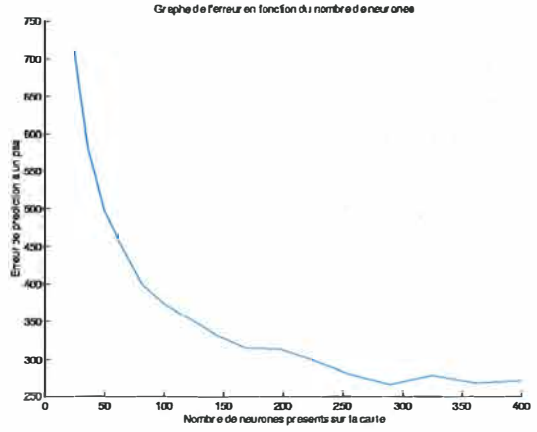
On peut ici constater qu'un minimum est bien présent, pour une carte carrée avec 225 neurones, soit une carte 15x15. Cependant, vu la forme générale du graphe, on peut supposer que l'erreur quadratique va continuer à diminuer lorsque le nombre de neurones augmentera encore. Il est donc intéressant de pousser ici les simulations plus loin.

5.4.2.2 La méthode « version Cottrell »

Une autre simulation a été lancé en utilisant la seconde version de la méthode de double quantification. Les paramètres sont les mêmes, excepté le nombre de neurones, variant ici de 1 à 400 dans chacun des deux espaces (cartes carrées 1x1 jusque 20x20).



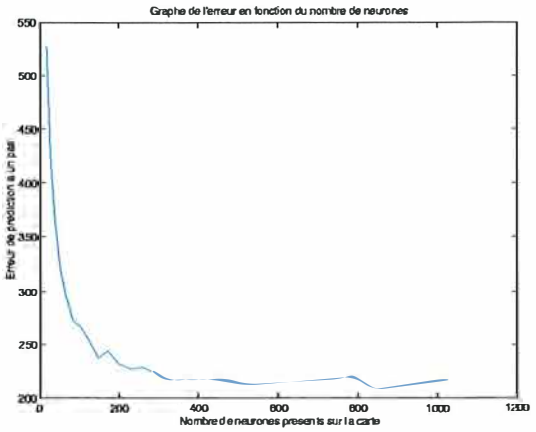
L'erreur quadratique décroît ici aussi en fonction du nombre de neurones. Le minimum est nettement visible :



On peut noter que le minimum, rencontré avec un modèle comportant 289 neurones, soit une carte 17x17, réalise une erreur quadratique de prédiction à un pas supérieure à celle du modèle développé avec la méthode « version DICE », puisque nous avons ici 267, contre 227, soit 17,5% d'erreur en plus.

5.4.2.3 Simulation supplémentaire

Comme nous avons pu le supposer lors de la première simulation avec la « version DICE » de la méthode, il se peut que l'erreur quadratique baisse encore. Une autre simulation a donc été effectuée, avec de 17 à 32 neurones sur le côté de la carte carrée, dans l'espace de base ; le côté des cartes présentes dans les clusters de l'espace de déformation variant ici encore de 1 à 16.



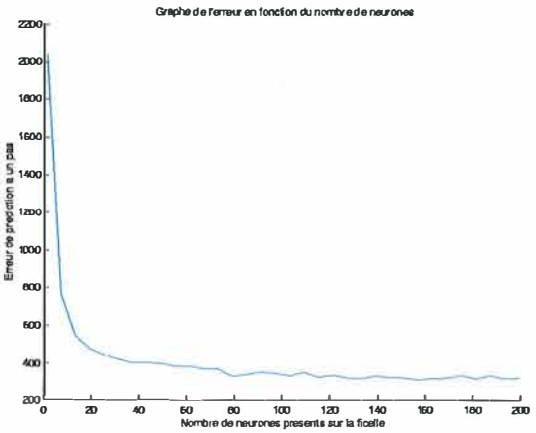
Un minimum est présent lorsque la carte carrée possède 841 neurones (29x29). Le nombre de neurones dans chaque cluster de l'espace de déformation, correspondant à ce minimum, est contenu dans le fichier résumant les 100 itérations de cross-validation. Ce nombre est ici de 1 neurone. Ce nombre peut paraître surprenant, à première vue. Les 841 centroïdes du modèle retenu sont associés à 841 clusters. Or, chaque cluster ne peut comporter qu'un nombre réduit de déformations. En effet, les 66% des données disponibles utilisées lors de l'apprentissage ne donnent que 641 déformations, à répartir entre tous les clusters. Il est donc intuitivement évident qu'un nombre réduit de centroïdes permet de résumer le ou les quelques point(s) présent(s) dans chaque cluster, lorsqu'il y en a.

5.4.3 Le cas des cartes en ficelle

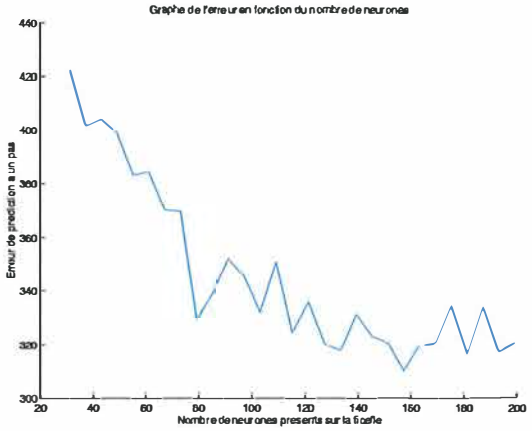
Nous allons maintenant observer l'erreur de prédiction à un pas lorsque des ficelles de Kohonen sont utilisées, tant dans l'espace de base que dans l'espace de déformation. Pour des raisons pratiques de temps de calcul, les simulations ont ici été limitées à 50 itérations de cross-validation.

5.4.3.1 La méthode « version DICE »

Les modèles testés au cours de cette simulation comportent de 1 à 200 neurones le long de la ficelle située dans l'espace de base, et des ficelles de 1 à 200 neurones, dans chaque cluster de l'espace de déformation. Pour des raisons de temps de calcul, les simulations s'allongeant considérablement, nous ne testons qu'un modèle sur 6, passant de 1 à 7, 13, 19, ..., 199 neurones dans chacun des deux espaces, soit un total de 1156 modèles différents.



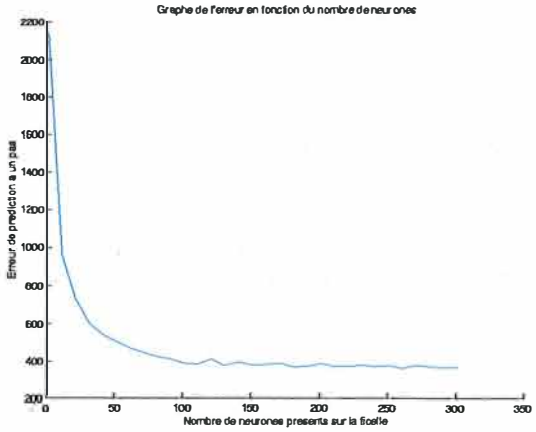
Ici encore, l'erreur quadratique diminue lorsque le nombre de neurones augmente. En observant le graphe de plus près, on obtient :



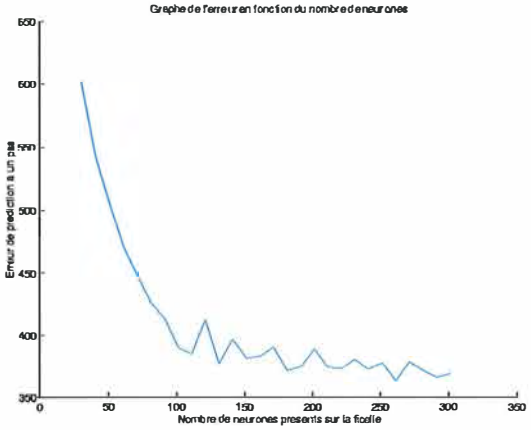
On constate la présence d'un minimum relativement bien marqué lorsque la ficelle comporte 157 neurones. Au-delà de ce nombre, la valeur de l'erreur quadratique a tendance à augmenter. Les simulations ne seront pas poussées plus loin, étant donnée cette tendance à la hausse. Par contre, il peut être intéressant de cerner de plus près le modèle optimal. En passant d'un modèle à l'autre par incrément de 6, il est fort possible que nous soyons passés à côté du véritable minimum. Le besoin d'une autre simulation se fait ici aussi ressentir.

5.4.3.2 La méthode « version Cottrell »

Présentons maintenant les résultats obtenus avec l'autre version de la méthode. Le nombre de neurones présents sur la ficelle varie de 1 à 301, en augmentant de 10 unités d'un modèle à l'autre :



Le minimum est obtenu avec un modèle de 261 neurones :

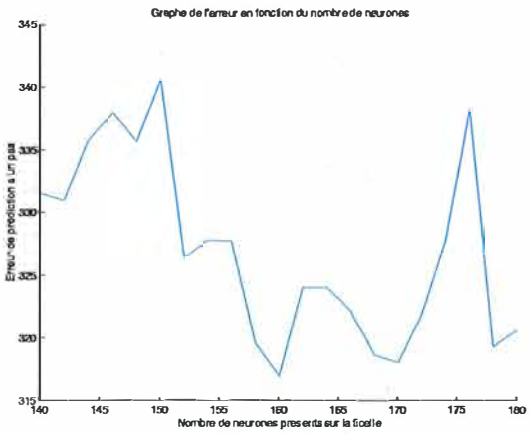


Ce modèle réalise alors une erreur quadratique de 363.

Ce qui est surtout intéressant, c'est l'ordre de grandeur de l'erreur quadratique obtenue lors de l'emploi de cette seconde version de la méthode, ici encore supérieure à celle de la simulation précédente.

5.4.3.3 Simulation supplémentaire

La simulation présentée dans la section 5.4.3.1 nous a permis de cerner plus ou moins la zone où doit normalement se situer le minimum. Une autre simulation nous permet de préciser ce résultat, en utilisant de 140 à 180 neurones le long de la ficelle de l'espace de base, par incrément de 2, la ficelle de chaque cluster de l'espace de déformation comportant, quant à elle, de 1 à 40 neurones, par incrément de 2.



Nous observons de nouveau un minimum, cohérent avec l'autre simulation, à trois neurones près (160 ici pour 157 précédemment). Le nombre de neurones dans chaque cluster de l'espace de déformation est ici aussi de 1. Par ailleurs, le comportement au-delà de ce minimum est également cohérent, puisqu'on peut constater la même tendance à la hausse. Le minimum est toutefois légèrement inférieur lors de la simulation précédente : 316 contre 310 lors de la première simulation.

5.4.4 Notes

Nous pouvons résumer ces premières simulations sur le fichier de données Santa Fe A, comme suit :

- la méthode « version DICE » semble être plus performante lors de la prédiction à un pas, quelle que soit la forme de la carte de Kohonen utilisée.
- les cartes carrées permettent de modéliser la série en réalisant une erreur quadratique inférieure au cas des ficelles, quelle que soit la version de la méthode utilisée.

Notons enfin que si l'erreur quadratique observée pour le minimum rencontré dans le cas des ficelles est effectivement supérieure à celle obtenue dans le cas des cartes carrées (316 contre 207), le modèle optimal est toutefois moins complexe, dans ce deuxième cas, puisqu'il comporte cinq fois moins de neurones (160 contre 841).

Pour des raisons pratiques, principalement des contraintes de temps (longueur des simulations effectuées), le cas des cartes de Kohonen rectangulaires n'est pas abordé.

De plus, nous n'avons pas non plus testé les nombreux cas de simulations « croisées », c'est-à-dire les cas où un certain type de cartes est utilisé dans l'espace de base, alors qu'un autre est utilisé en déformation. Par exemple, le cas de cartes carrées en base et de ficelle en déformation.

Enfin, on peut également imaginer le cas où la forme de la carte serait différente dans chacun des clusters de l'espace de déformation. De même que le nombre de neurones qui quantifie chacun de ces clusters, jusqu'ici fixe et identique pour tous les clusters. On peut immédiatement penser au cas de deux clusters différents en déformation, l'un avec une centaine de déformations, l'autre avec une dizaine, mais tous les deux quantifiés par trois neurones, par exemple. Il est clair que la perte d'informations sera relativement importante dans un cas et beaucoup plus réduite dans l'autre.

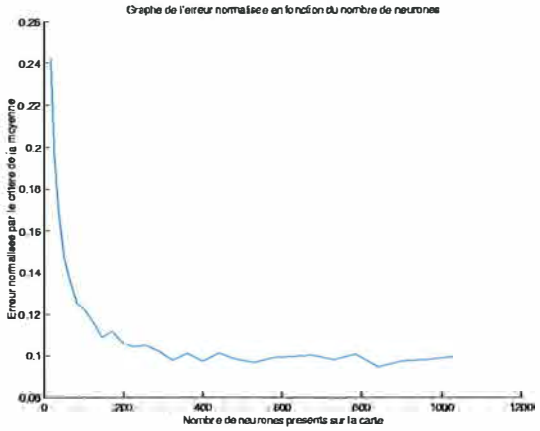
Ces autres possibilités n'ont pas été inclues dans l'algorithme qui devient alors beaucoup plus complexe, et beaucoup plus lent. Le temps de calcul s'allonge en effet considérablement, vu qu'il faudrait demander à l'utilisateur quelle carte utiliser pour chaque cluster (nombre de neurones et forme) et ce pour chaque modèle, lors de chaque itération de cross-validation !

5.4.5 Sélection de modèle et performances

Aux vues des simulations effectuées ci-dessus, le modèle optimal pour le fichier Santa Fe A est celui que nous avons obtenu avec des cartes carrées de 29x29 neurones, dans l'espace de base, et de 1 neurone dans l'espace de déformation. Ce modèle a été obtenu avec la « version DICE » de la méthode de double quantification.

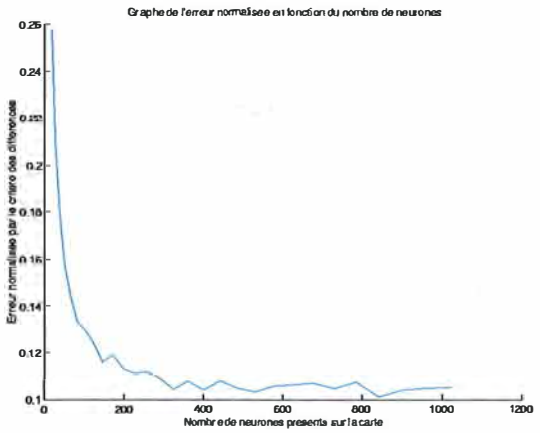
Comment situer ce modèle, dans l'absolu ? Le fichier de données Santa Fe A a fait l'objet d'une compétition, relativement connue dans le monde des réseaux de neurones artificiels. De nombreux chercheurs se sont attelés à développer des modèles pour cette série. Parmi ces modèles, les plus performants (RBF) réalisaient une erreur quadratique de plus ou moins 50 [16], à comparer avec notre erreur de 207. En outre, les modèles linéaires les plus performants qui ont été développés pour cette série réalisent une erreur quadratique de l'ordre de 400 [16]. Autrement dit, dans l'absolu, notre solution n'est pas la meilleure qui soit, mais elle se situe malgré tout dans une bonne moyenne.

Pour mieux évaluer le modèle retenu, il est intéressant d'observer les autres critères d'erreur que nous avons définis. Tout d'abord, nous observons le graphe de l'erreur quadratique normalisée par l'erreur par rapport à la moyenne des données, le critère E_N. Pour des raisons de place, nous ne présentons ici que des graphes tronqués des premières valeurs :



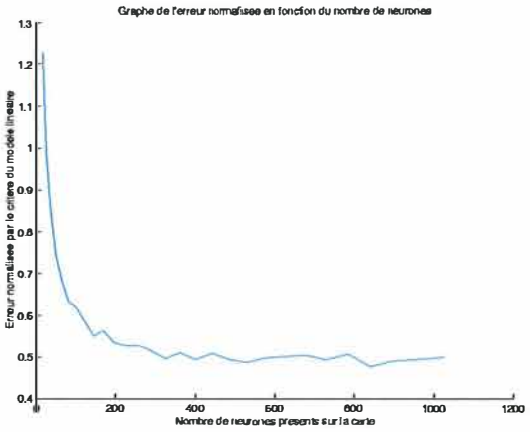
Si on peut effectivement constater un minimum en 841 neurones (carte 29x29), le plus intéressant est l'ordre de grandeur de l'erreur quadratique normalisée selon le critère de la moyenne, qui est de 0,095 avec 841 neurones. Pour rappel, un rapport proche de 1 aurait signifié que notre méthode ne faisait pas mieux que de prédire comme prochaine valeur la moyenne des données passées, ce qui n'est pas le cas ici.

Dans le cas du critère E_D, critère de l'erreur quadratique normalisée par l'erreur des différences :



Ici aussi, un minimum est rencontré pour une carte carrée de 29x29 neurones, mais une fois encore, c'est surtout l'ordre de grandeur qui est important. Nous avons ici 0,104. Or, l'interprétation de ce critère nous indique que, dans le cas d'une valeur proche de 1, le modèle ne fait pas mieux que d'utiliser la dernière valeur connue pour prédire la suivante. Ce qui revient à dire que la valeur suivante du processus fonction du temps, pour demain par exemple, c'est la même que celle d'aujourd'hui. Le modèle développé fait nettement mieux.

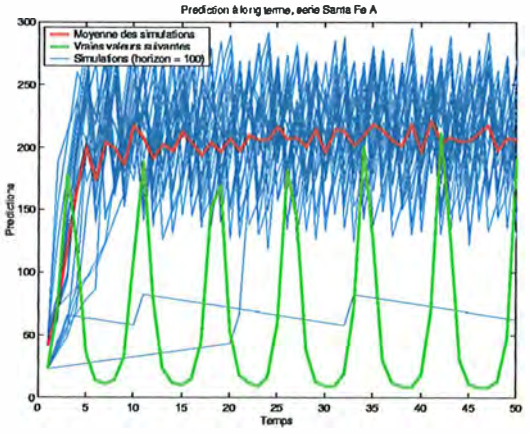
Enfin, observons le comportement dans le cas du dernier critère, E_L, le critère de l'erreur quadratique normalisée par l'erreur d'un modèle linéaire :



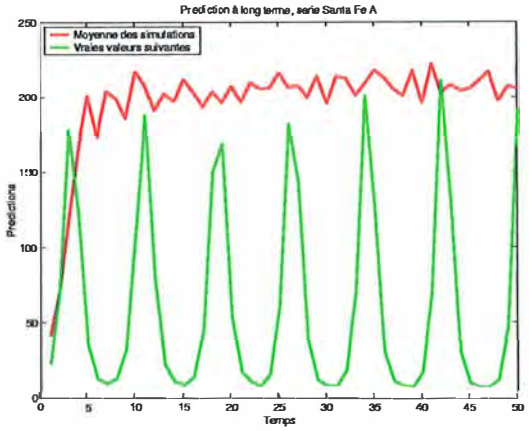
On constate ici encore la présence d'un minimum, toujours au même point (carte carrée 29x29), pour lequel l'erreur est de 0,476. Intuitivement, cela signifie que le modèle développé fait environ deux fois mieux qu'un modèle linéaire. Ceci est assez cohérent avec les résultats observés dans l'absolu, où notre méthode fait une erreur quadratique de 207 alors qu'un modèle linéaire fait environ 400.

5.4.6 Prédiction à long terme

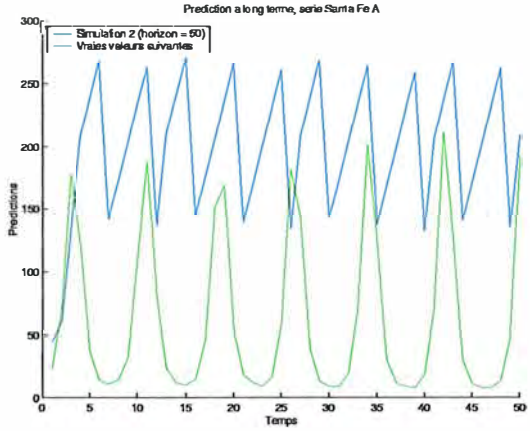
L'étude de l'erreur nous a permis de choisir un modèle optimal. Sur base de ce « meilleur » modèle des données, on s'intéresse maintenant à la simulation à long terme. L'horizon de prédiction, pour cette simulation, est de 50. Les simulations sont répétées 25 fois, puis la moyenne de ces simulations est calculée. Voici le graphe représentant les simulations, la moyenne de ces simulations, tout en les comparant avec les vraies valeurs de la série :



Si on « nettoie » le graphe pour ne garder que la moyenne et les vraies valeurs :



Nous observons sur œ graphe que les 4 premières valeurs de la série sont correctement prédites. Au-delà, seul l'ordre de grandeur des valeurs prédites est correct. Les phénomènes périodiques de la série ne sont pas mis en évidence par la modélisation. Cependant, cette non-périodicité de la série est en fait perdue lors du calcul de la moyenne des simulations. Pour preuve, on peut considérer les simulations une à une. Par exemple, voici la deuxième des 25 simulations qui ont été effectuées :



Bien que la simulation varie beaucoup plus, avec une fréquence de variation plus élevée et une amplitude différente, le caractère périodique de la série est clairement mis en évidence, à long terme.

On constate par ailleurs sur ce dernier graphe que l'évolution à long terme ne suit pas exactement la même dynamique que celle de la série. Une autre explication possible vient de l'erreur produite lors de chaque prédiction à un pas. Plus formellement, supposons que la valeur de départ soit la dernière connue, ce que nous notons $x(t)$. Le modèle des données nous permet de prédire la valeur suivante, que nous notons $\hat{x}(t+1)$. Le modèle des données n'étant qu'une approximation de la véritable fonction qui caractérise l'évolution des données, $\hat{x}(t+1)$ n'est jamais qu'une estimation de la véritable valeur suivante, qu'est $\hat{x}(t+1)$. Qui dit estimation dit erreur, ce qui se traduit ici par une différence entre $\hat{x}(t+1)$ et $x(t+1)$, notée e_1.

Si nous itérons la prédiction, par exmple jusqu'à un certain horizon i, lors de chaque prédiction, on effectue une erreur. Par conséquent, lors du calcul de $\hat{x}(t+2)$, nous commettons une deuxième erreur e_2. De plus, comme le point de départ de cette prédiction, $\hat{x}(t+1)$,

comporte déjà un certain terme d'erreur e_1, la prédiction $\hat{x}(t+2)$ est « doublement fausse », puisque l'erreur commise e_2 doit être ajoutée à celle effectuée sur le point de départ, soit e_1.

De façon générale, lorsqu'on itère la prédiction jusqu'à un horizon de prédiction i, on commet une erreur E_i qui vaut :

$$E_i = \sum_{k=1}^{i} e_i.$$

Par conséquent, plus l'horizon de prédiction est élevé, plus la prédiction obtenue devient incertaine, les erreurs s'accumulant petit à petit.

La réflexion ci-dessus peut nous permettre d'envisager une amélioration de la méthode, qui pourrait dans une certaine mesure répondre au problème de l'erreur croissante à long terme. Cette amélioration consisterait à inclure les n premières prédictions obtenues dans le fichier de données, de recommencer l'apprentissage sur base de ce nouveau fichier de données, puis de prédire les n données suivantes, et de recommencer jusqu'à l'horizon de prédiction final i souhaité.

Ce changement de la méthode auquel nous avons pensé, mais qui n'est pas implémenté dans le cadre de ce travail, pourrait peut-être améliorer la prédiction à long terme, en annulant l'erreur, toutes les n itérations, puisqu'on part d'un nouveau point de départ. De plus, si les dernières valeurs prédites ont tendance à s'écarter de la dynamique générale du processus, l'importance de cet écart sera réduite par la quantification vectorielle, ce qui permettrait de corriger quelque peu le tir.

5.5 Conclusion

Nous avons abordé la série Santa Fe A pour tester l'implémentation de la méthode. Le but était de faire tourner le programme sur un exemple concret, « grandeur nature », avant de nous atteler au fichier fourni par Gaz de France.

L'étude des données nous a permis de mettre en évidence un lien entre les points de l'espace de base et leurs déformations.

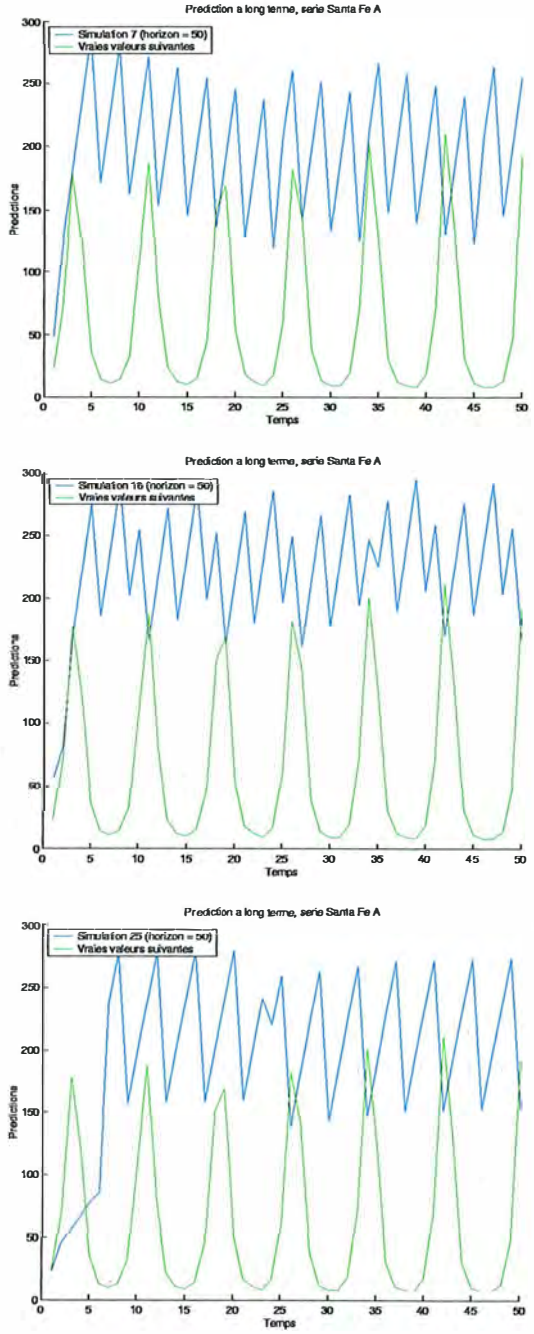
La prédiction à un pas nous a donné des performances relativement bonnes, puisque notre méthode se situe dans la moyenne des modèles neuronaux développés : moins performante que les meilleurs RBF qui ont modélisé la série, mais tout de même meilleure que les modèles linéaires. Ce fait a d'ailleurs été confirmé en étudiant les autres critères d'erreur que nous avions définis.

En ce qui concerne la prédiction à long terme, les résultats sont plus mitigés. Très bons à court terme (pour les 4 premières valeurs), moins bons à long terme. Malgré tout, certaines informations ont été mises en évidence :

- la périodicité de la série est reconnue, à long terme,
- l'amplitude de la série est considérée comme quasi constante sur la période observée,
- il n'y a pas de cassure dans l'évolution du graphe jusqu'à l'horizon de prédiction $t+50$.

Par cassure, nous désignons le changement de comportement brusque de la série, visible sur le graphe de présentation de la série, au début de ce chapitre.

Une fois encore, ces informations sont difficiles à obtenir en observant le graphe de la moyenne des prédictions. On peut tout de même justifier ces affirmations en observant les simulations de façon séparée, comme nous l'avons déjà fait ci-dessus :



Chapitre 6

Application de la méthode : la série Gaz de France

6.1 Présentation générale

Ce chapitre n'est constitué que d'une présentation assez sommaire du travail réalisé dans le cadre de l'étude de faisabilité commandée au centre de recherche en Statistiques Appliquées et Modélisation Stochastique (SAMOS) par Gaz de France.

Pour notre part, la participation à cette étude de faisabilité consistait à appliquer au fichier de données fourni par Gaz de France la méthode de double quantification vectorielle présentée au chapitre 3.

Vu le caractère confidentiel de ces données, les résultats obtenus sont décrits dans un document indépendant de ce mémoire et fourni en annexe. Il ne sera ici présenté que le protocole de simulations que nous avons choisi d'appliquer, ainsi que quelques remarques générales portant sur la méthode et les modifications apportées.

6.2 Application

6.2.1 Les données

Au début du travail concernant le fichier Gaz de France, il a été réalisé une première étude des données disponibles. Cette analyse des données nous a permis de prendre connaissance de certaines informations : données creuses, dépendances entre certaines colonnes, ...

6.2.2 Les différentes séries

L'analyse des coefficients de corrélation nous a conduit à considérer la série temporelle de départ sous plusieurs angles. La méthode de double quantification vectorielle a donc été appliquée sur les données Gaz de France selon ces différentes approches.

6.2.2.1 Série brute

Dans un premier temps, on a choisi de considérer la série temporelle telle qu'elle se trouvait dans le fichier fourni par Gaz de France, alors appelée série brute.

6.2.2.2 Série calée

Suite à l'analyse des coefficients de corrélation, où il a été constaté une influence d'une des colonnes sur les valeurs respectives des autres colonnes, la série calée a été définie. Dans cette deuxième série, les données sont « calées » sur base de la valeur d'une des colonnes.

En pratique, le fait de caler une série revient à soustraire à chaque ligne de la base de données la valeur présente sur une certaine colonne de cette ligne.

Par exemple, soit le fichier F comportant n données temporelles, chacune de la forme :

$$x(t) = (x_1(t),\ x_2(t),\ \ldots,\ x_j(t),\ \ldots,\ x_m(t))$$

où m est la dimension des données.

Si on choisit de caler la série selon la colonne j, on obtient une nouvelle série, dont les données sont maintenant :

$$x_{calée}(t) = (x_1(t) - x_j(t),\ x_2(t) - x_j(t),\ \ldots,\ 0,\ \ldots,\ x_m(t) - x_j(t)).$$

Cette manipulation permet de mettre toutes les données sur un pied d'égalité en supprimant le lien avec la colonne qui exerce une influence sur les autres colonnes. Il est dès lors possible d'étudier les données en faisant abstraction des phénomènes extérieurs spécifiquement liés à une des colonnes.

6.2.2.3 Série des différences de la série brute

Rappelons que la série est étudiée au moyen de la méthode de double quantification vectorielle. Cette méthode caractérise les données au moyen d'un certain nombre de centroïdes dans l'espace de base (les données) et dans l'espace de déformation (les déformatins, calculées comme les différences entre ces données). Il nous a dès lors semblé intéressant de pouvoir étudier la série des différences des données brutes, pour nous retrouver avec une application de la méthode sur des déformations dans l'espace de base. Par conséquent, les éléments dans l'espace de déformation sont en fait les déformations des déformations des données de départ.

Cette troisième série est désignée sous l'appellation de série des différences de la série brute.

6.2.2.4 Série des différences de la série calée

Dans la mesure où il était possible de prendre la série des différences d'une série, nous avons également considéré la série des différences de la série calée.

6.2.2.5 Conséquences

Cette volonté de pouvoir considérer d'autres séries que celle reçue de Gaz de France nous a conduit à apporter de nouvelles modifications au code de l'application, notamment en ajoutant des choix supplémentaires au niveau du pré-traitement des données (calcul de la série des différences) ainsi que du choix du type de série (brute ou calée), sans oublier les manipulations nécessaires pour caler la série, le cas échéant.

Notons que ces modifications nous ont contraints à réexaminer complètement l'implémentation. Dès lors qu'on décide de caler les données, par exemple, il faut pouvoir, au moment de la prédiction, rajouter la valeur qui a permis de caler la donnée de départ qui

doit être prédite. De même, si on travaille avec une série en différence, il faut pouvoir garder trace de la valeur de base de la déformation, qui est maintenant dans l'espace de base, lorsque la déformation de la déformation, dans ce cas-ci, sera choisie lors de la prédiction.

6.2.3 Les simulations

Comme précisé dans la présentation de la méthode au chapitre 3, celle-ci nous offre trois possibilités :

- une analyse des données *a priori*,
- la recherche d'un modèle sur base du minimum de l'erreur de prédiction à un pas et procédure de type Monte-Carlo avec cross-validation,
- la simulation de l'évolution à long terme ou prédiction à n pas, de nouveau avec procédure Monte-Carlo et cross-validation.

6.2.3.1 Analyse des données

L'analyse des données a été réalisée en utilisant les 4 séries temporelles. Cette étape nous a permis d'écarter certaines séries et de concentrer nos efforts sur les autres. La méthode « version Cottrell » fut la seule que nous avons utilisée, dans la mesure où l'autre version ne permet pas cette analyse des données.

Notons que l'étude des données par la méthode nous a permis de constater une convergence relativement mauvaise des ficelles de Kohonen utilisées. Après une fouille complète du code, il a été conclu que les lois de décroissance utilisées n'étaient pas adaptées au fichier Gaz de France. Nous avons donc défini de nouvelles lois, et par conséquent dû revoir le code de l'implémentation de l'algorithme de Kohonen.

Le test χ^2 sur la table de contingence a également été étudié, ainsi que les profils des données et centroïdes, pour chacune des 4 séries temporelles. Une classification des données selon certains critères a également été proposée dans le cas de certaines séries.

6.2.3.2 Sélection de modèle

La recherche du minimum de l'erreur de prédiction a été effectuée sur base des deux versions de la méthode, dans un premier temps, mais la méthode « version DICE » a été écartée pour des raisons de cohérence. Il a en effet été choisi de ne présenter à Gaz de France que les résultats issus d'une même version de la méthode.

6.2.3.3 Evolution à long terme

Enfin, la prédiction à long terme a été appliquée, en n'utilisant de nouveau que la « version Cottrell ».

6.3 Note globale sur les résultats obtenus

Dans l'ensemble, les résultats obtenus sont plutôt satisfaisants.

Mais en répondant aux premières questions de Gaz de France, de nombreuses autres questions ont été soulevées. Malheureusement, il n'a pas été possible de poursuivre l'étude de la série, ni de tenter d'apporter une réponse à ces nouvelles questions. Nous étions en effet limités par le temps : l'étude de faisabilité a débuté en septembre 2001 et a été clôturée en janvier 2002.

Sur base des premiers résultats, Gaz de France a été très intéressé et a demandé certaines précisions. La méthode a été de nouveau appliquée, suivant les demandes de Gaz de France. Les résultats ont été fournis au mois de février, mais cette partie du travail dépasse le cadre de ce mémoire.

Pour plus de précisions, le lecteur est invité à consulter l'annexe confidentielle.

Chapitre 7

Conclusion

Au cours de ce mémoire, nous avons eu l'occasion d'appliquer une méthode connue et publiée dans la littérature à un cas concret, le fichier de données fourni par Gaz de France.

Ce travail n'aurait pas été possible sans une série de modifications apportées à la méthode telle qu'elle figure dans l'article de référence, à la base de ce travail. Ces modifications ont été apportées dans le but de généraliser la méthode de double quantification vectorielle, en y intégrant de nouveaux aspects :

- manipulation de données creuses et non creuses, ce qui a notamment induit une généralisation de l'algorithme de Kohonen,
- généralisation au cas des données de dimension supérieure à 1,
- intégration d'une deuxième version de la méthode,
- implémentation de multiples traitements des données (pré-traitement, différentes séries, ...),
- définition de différents critères d'erreur, pour situer le modèle développé lors de la double quantification

Cette méthode a par ailleurs été testée sur une série temporelle bien connue, principalement dans le but de valider le code de l'implémentation, mais également par curiosité. S'il était intéressant de tester l'aspect programmation, il était tout aussi intéressant d'évaluer les capacités de la méthode, que ce soit en prédiction à un pas ou en simulation de l'évolution à long terme.

Si les résultats pour Gaz de France constituent effectivement une partie importante de ce travail, il nous semble intéressant de souligner qu'en outre des modifications, une certaine réflexion sur la méthode a également été effectuée, notamment en proposant un moyen d'améliorer la prédiction à long terme.

Enfin, disposant de deux versions de la méthode, différentes bien que fort proches, il est tentant d'essayer de dire quelle version est la meilleure. Nous resterons prudents à cet égard, simplement parce que le comportement de la méthode s'est montré différent selon la série abordée, et selon le type de prédiction choisi, à un pas ou à long terme.

Annexe A

Tableau des températures

Pour illustrer les concepts de donnée et série temporelles, nous avons choisi de présenter un processus évoluant au cours du temps parmi les plus connus, à savoir la température extérieure.

Cette température a fait l'objet de 5 jours d'observations, entre le 5 mars et le 9 mars 2002 ; jours au cours desquels 24 mesures ont été effectuées, heure par heure, ce qui constitue bien un ensemble de 120 mesures. Ces valeurs ont été relevées dans la région de Gembloux.

Ci-dessous, nous présentons l'ensemble de ces mesures dans un tableau comportant trois colonnes par jour d'observation : numéro de la donnée, heure de mesure de la donnée, donnée elle-même.

05/03/2002			06/03/2002			07/03/2002			08/03/2002			09/03/2002		
n°	heure	temp.	n°	heure	temp.	n°	heure	temp.	n°	heure	temp.	n°	heure	temp.
1	0	4	25	0	6	49	0	9	73	0	7	97	0	8.2
2	1	3	26	1	5	50	1	9.5	74	1	6	98	1	8.1
3	2	2	27	2	5	51	2	10	75	2	5	99	2	8.2
4	3	1.8	28	3	5	52	3	10	76	3	4.5	100	3	8.5
5	4	1.2	29	4	4.2	53	4	10.2	77	4	4	101	4	8.1
6	5	1.8	30	5	4.2	54	5	10.5	78	5	3.5	102	5	8.1
7	6	1.5	31	6	4.5	55	6	10.8	79	6	3	103	6	8.4
8	7	1.1	32	7	4.8	56	7	10.9	80	7	3.5	104	7	8.5
9	8	2	33	8	5.1	57	8	11	81	8	4.9	105	8	9
10	9	4	34	9	6	58	9	11.5	82	9	8	106	9	9.5
11	10	7	35	10	7	59	10	12	83	10	9.2	107	10	9.9
12	11	9.8	36	11	8	60	11	13	84	11	11	108	11	9.5
13	12	10.5	37	12	9	61	12	13	85	12	12	109	12	10.2
14	13	10.5	38	13	9	62	13	13.8	86	13	13.5	110	13	11
15	14	10	39	14	9	63	14	13	87	14	14	111	14	11
16	15	9.5	40	15	8.8	64	15	12	88	15	15	112	15	10.8
17	16	9	41	16	8.8	65	16	12.8	89	16	14.5	113	16	12
18	17	8.5	42	17	8.8	66	17	11.8	90	17	13	114	17	10.6
19	18	8	43	18	8.8	67	18	10.2	91	18	11	115	18	9
20	19	7.5	44	19	8.8	68	19	10	92	19	10	116	19	8.5
21	20	7.5	45	20	8.8	69	20	9.2	93	20	9.9	117	20	8.2
22	21	7.5	46	21	8.9	70	21	9	94	21	9	118	21	8
23	22	7.3	47	22	9	71	22	8	95	22	8.8	119	22	7.5
24	23	7.2	48	23	9	72	23	8	96	23	8.5	120	23	7

Annexe B

Comportement des cartes de Kohonen

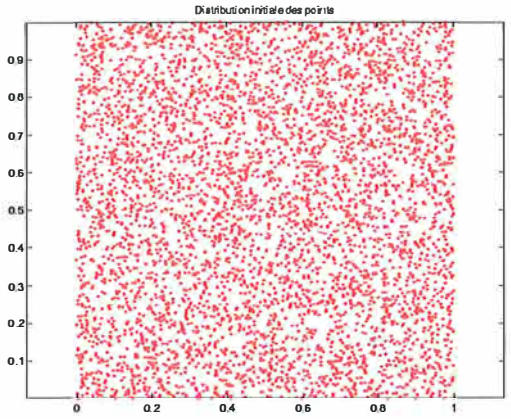
Cette annexe nous permet de présenter de manière graphique les cartes de Kohonen, sur un exemple relativement simple : une distribution uniforme de points, en deux dimensions, générés aléatoirement. Nous allons utiliser cet exemple pour décrire l'évolution de la carte au fur et à mesure de la présentation des points au réseau, et mettre en évidence l'importance des paramètres sur la convergence de la carte.

B.1 Présentation des données

Dans cet exemple, nous allons appliquer l'algorithme de Kohonen à une base de données comprenant des points de $\mathbb{R}^2$, générés aléatoirement selon une loi uniforme, et dont les deux composantes sont comprises entre 0 et 1. Cet ensemble de données comprend 5 000 points, dont voici les dix premiers, pour fixer les idées :

Première composante	Deuxième composante
0.9501293	0.5483870
0.2311385	0.8640048
0.6068426	0.4565521
0.4859825	0.7125836
0.8912990	0.3294646
0.7620968	0.4157588
0.4564677	0.1903881
0.0185036	0.8311772
0.8214072	0.8687236
0.4447034	0.6178251

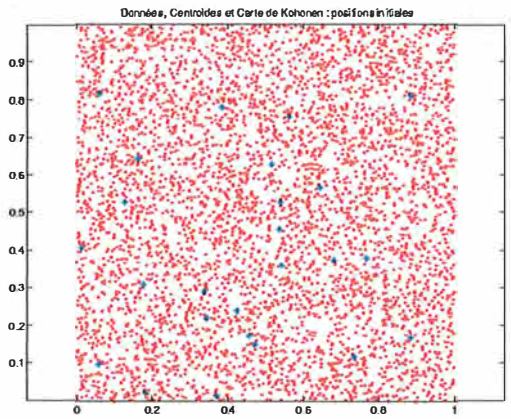
Voici une représentation de ce fichier :



B.2 Importance des paramètres variables

Par la suite, nous utiliserons le terme d'« itération » de l'algorithme pour désigner la présentation d'une donnée au réseau, en entrée, ce qui conduit à déterminer le centroïde vainqueur et à mettre à jour les positions respectives de ce centroïde, et de ses voisins, le cas échéant.

Les premières exécutions de l'algorithme qui vont suivre sont toutes basées sur une même initialisation des neurones, afin de pouvoir comparer des choses comparables. Voici une représentation de cette initialisation :



Les points bleus représentent les centroïdes, avant exécution de l'algorithme. Par la suite, les conventions suivantes seront utilisées :

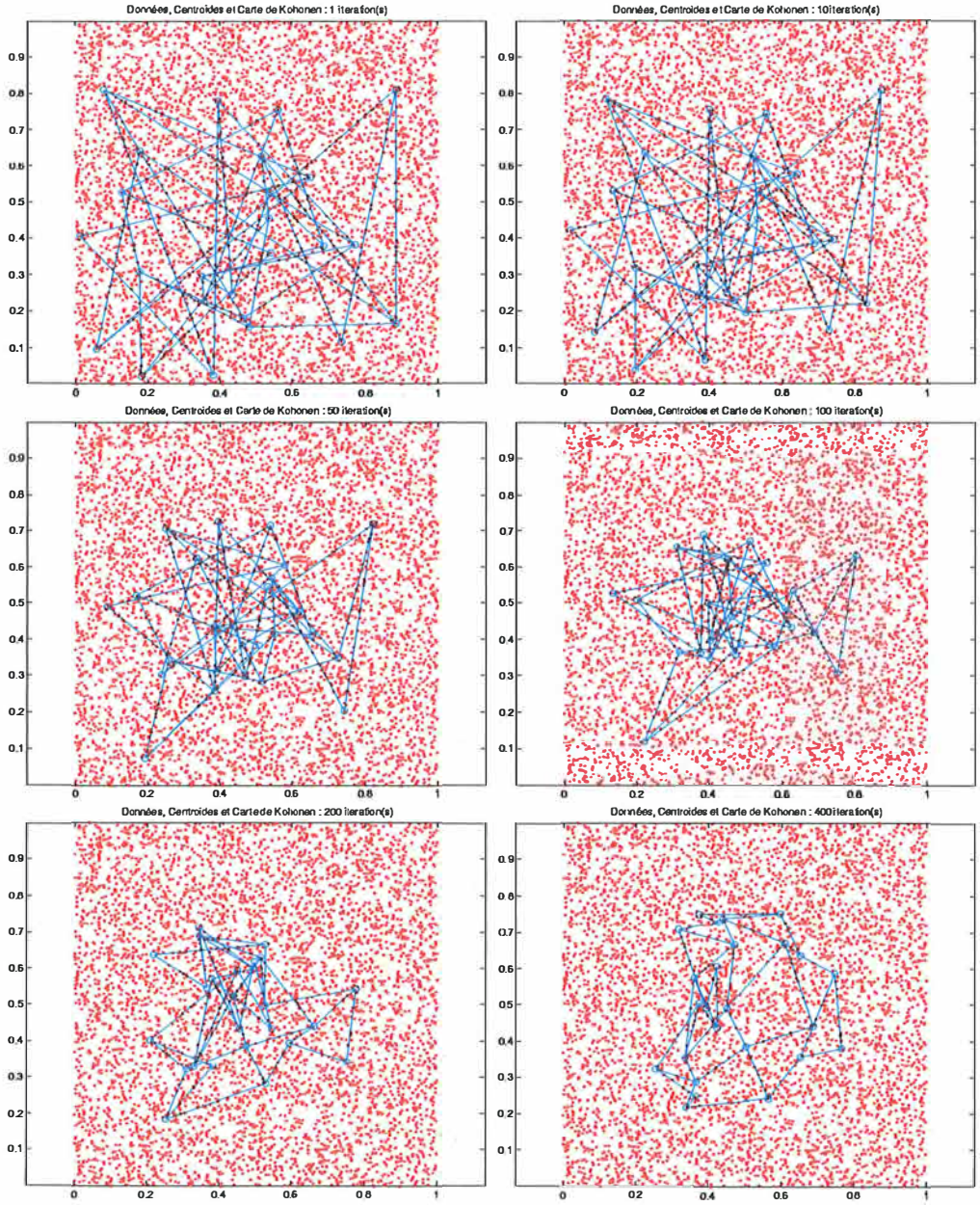
- les données seront représentées par des points rouges,
- les neurones seront représentés par des ronds bleus,
- les liens symbolisant la relation de voisinage physique au niveau de la carte seront représentés par des segments de droite bleus entre les neurones.

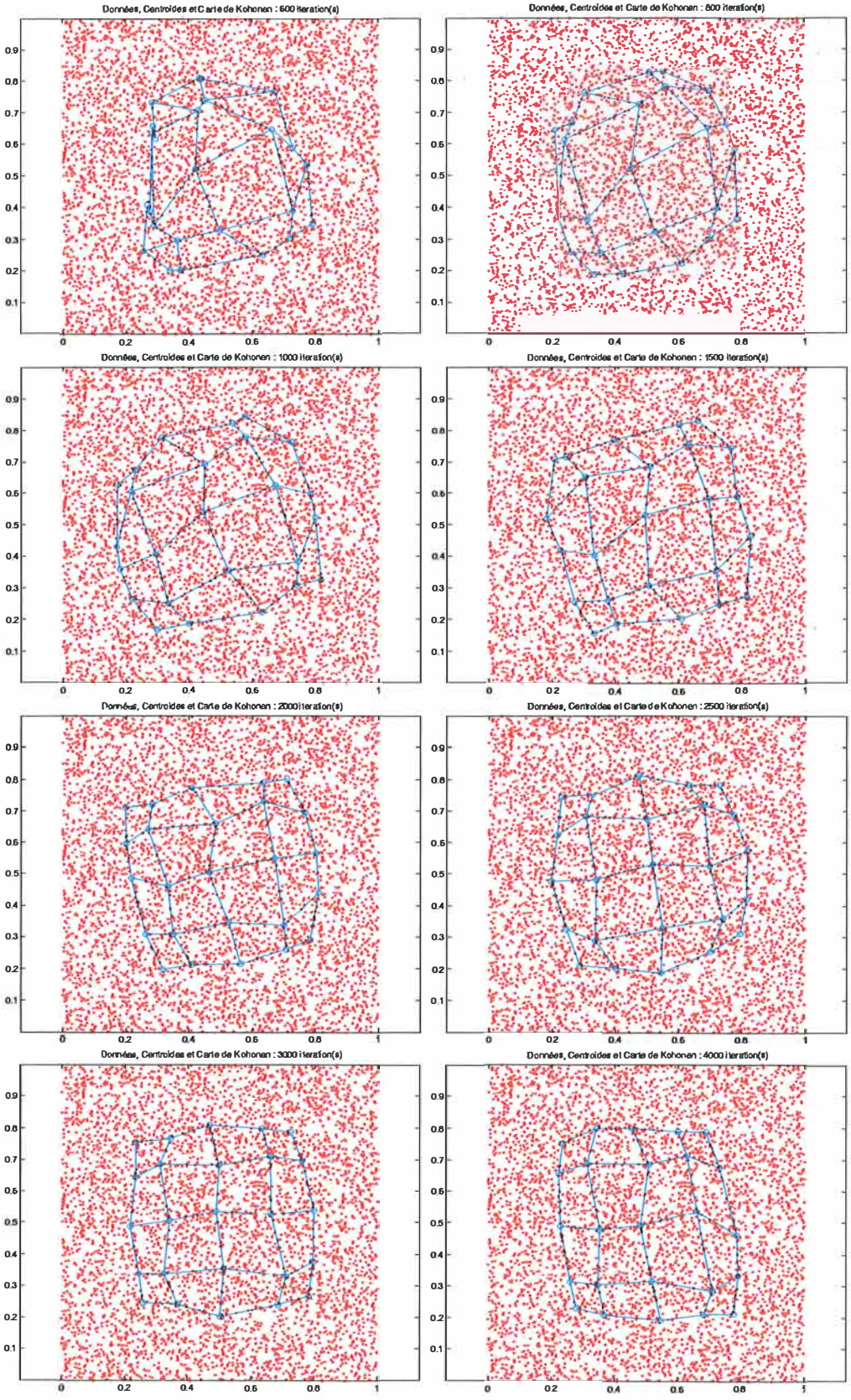
B.2.1 Paramètres constants

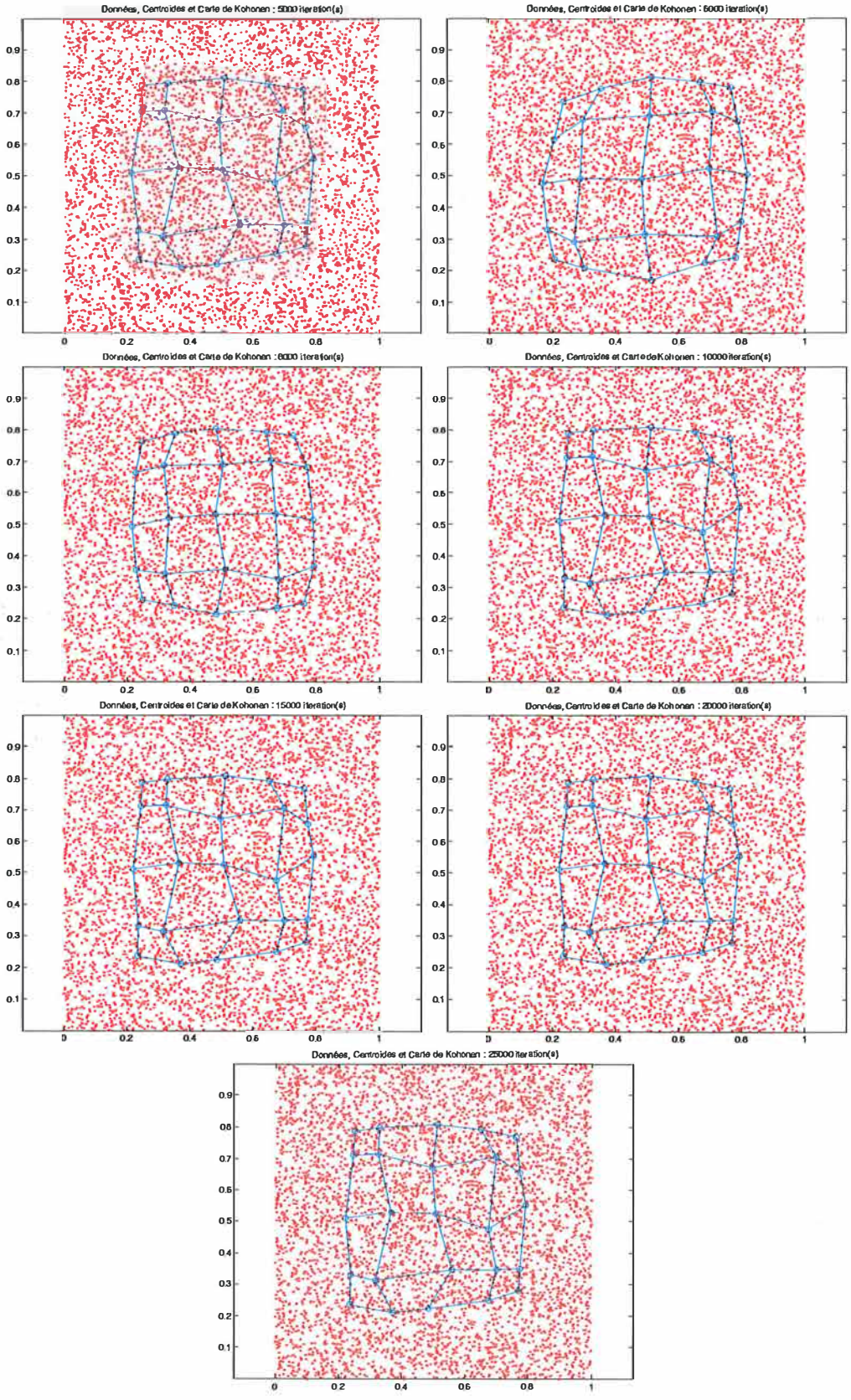
La première simulation montre l'évolution de la carte lors de l'exécution de l'algorithme de Kohonen « première version » sur l'ensemble des points présentés ci-dessus.

Nous considérons ici le cas de l'algorithme à paramètres constants, présenté dans la section 2.3.2 b) du chapitre 2. Une carte carrée de 25 neurones (5x5) est utilisée. La distance de voisinage *distmax* est fixée à la valeur constante de 2, le taux d'apprentissage α étant lui de 0,02.

Les figures ci-dessous représentent l'état du réseau après un certain nombre d'itérations, nombre indiqué dans le titre de l'image. Les images se suivent de gauche à droite et de haut en bas.




Données, Centroïdes et Carte de Kohonen : 600 iteration(s)
Données, Centroïdes et Carte de Kohonen : 800 iteration(s)
Données, Centroïdes et Carte de Kohonen : 1000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 1500 iteration(s)
Données, Centroïdes et Carte de Kohonen : 2000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 2500 iteration(s)
Données, Centroïdes et Carte de Kohonen : 3000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 4000 iteration(s)


Données, Centroides et Carte de Kohonen : 5000 iteration(s)
Données, Centroides et Carte de Kohonen : 6000 iteration(s)
Données, Centroides et Carte de Kohonen : 8000 iteration(s)
Données, Centroides et Carte de Kohonen : 10000 iteration(s)
Données, Centroides et Carte de Kohonen : 15000 iteration(s)
Données, Centroides et Carte de Kohonen : 20000 iteration(s)
Données, Centroides et Carte de Kohonen : 25000 iteration(s)

Plusieurs « étapes » peuvent être observées sur ces quelques figures :

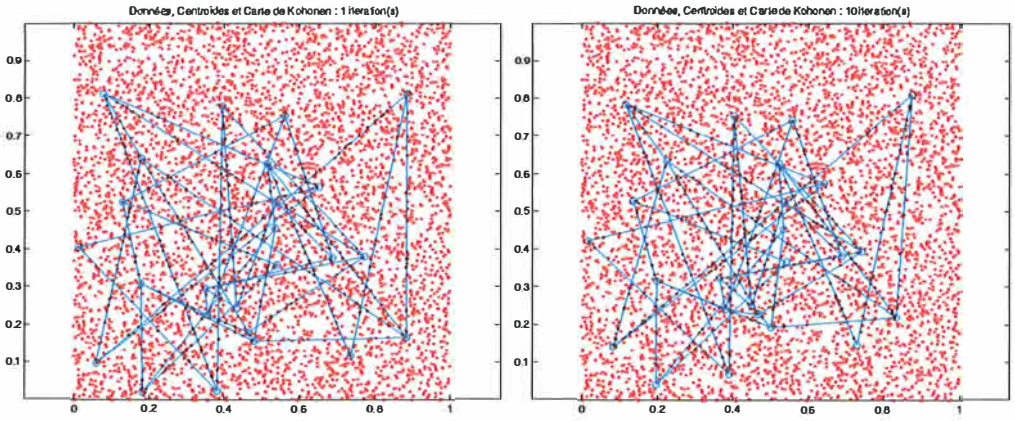
- La carte commence par se replier sur elle-même, tout en se retournant. Les neurones commencent à se placer les uns par rapport aux autres. Dans le même temps, les liens reliant les neurones entre eux ont de moins en moins tendance à se croiser. (représentations des itérations 1 à 200)
- Les itérations suivantes permettent à la carte de se déplier. Les neurones se répartissent petit à petit sur toute la distribution des points, comme s'ils avaient tendance à s'éloigner les uns des autres. (représentations des itérations 400 à 1 000)
- Vu la forme carrée de la distribution, la carte, également carrée, tourne légèrement sur elle-même pour épouser la forme de la distribution, tout en continuant de se déplier. (représentations des itérations 1 000 à 5 000)
- Les itérations suivantes ne permettent pas à la carte de recouvrir au mieux les points. En effet, ces itérations supplémentaires n'empêchent pas la constitution d'une large bande de points où aucun neurone ne se trouve, même si la forme de la carte varie encore un peu. (représentations des itérations 5 000 à 10 000)
- La carte ne change pratiquement plus de forme, quel que soit le nombre d'itérations supplémentaires. (représentations des itérations 10 000 à 25 000)

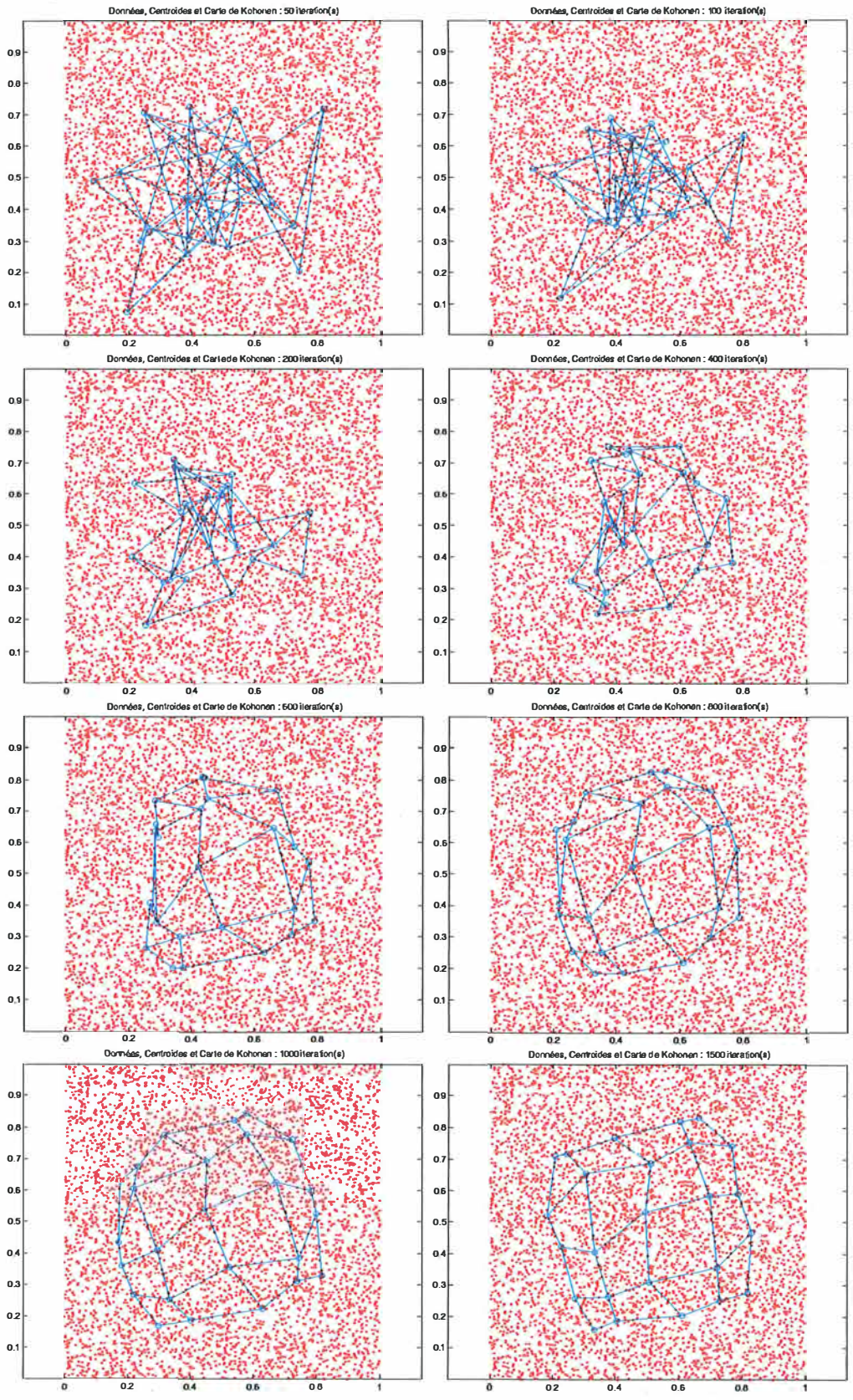
B.2.2 Paramètres variables

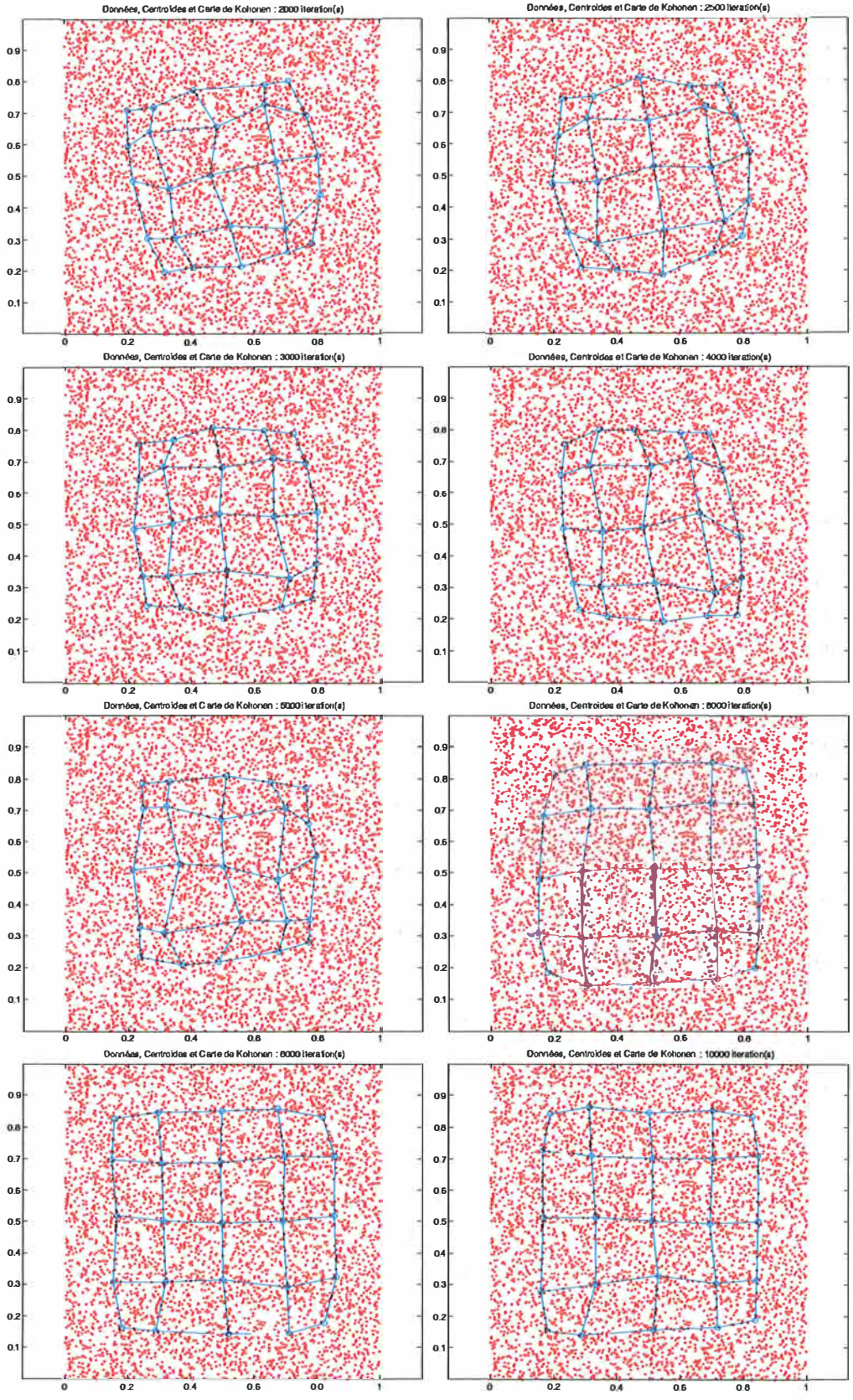
Pour améliorer la convergence obtenue, des paramètres variables sont utilisés. Ces paramètres sont décroissants en fonction du temps et permettent d'obtenir un meilleur recouvrement des données par les neurones de la carte de Kohonen.

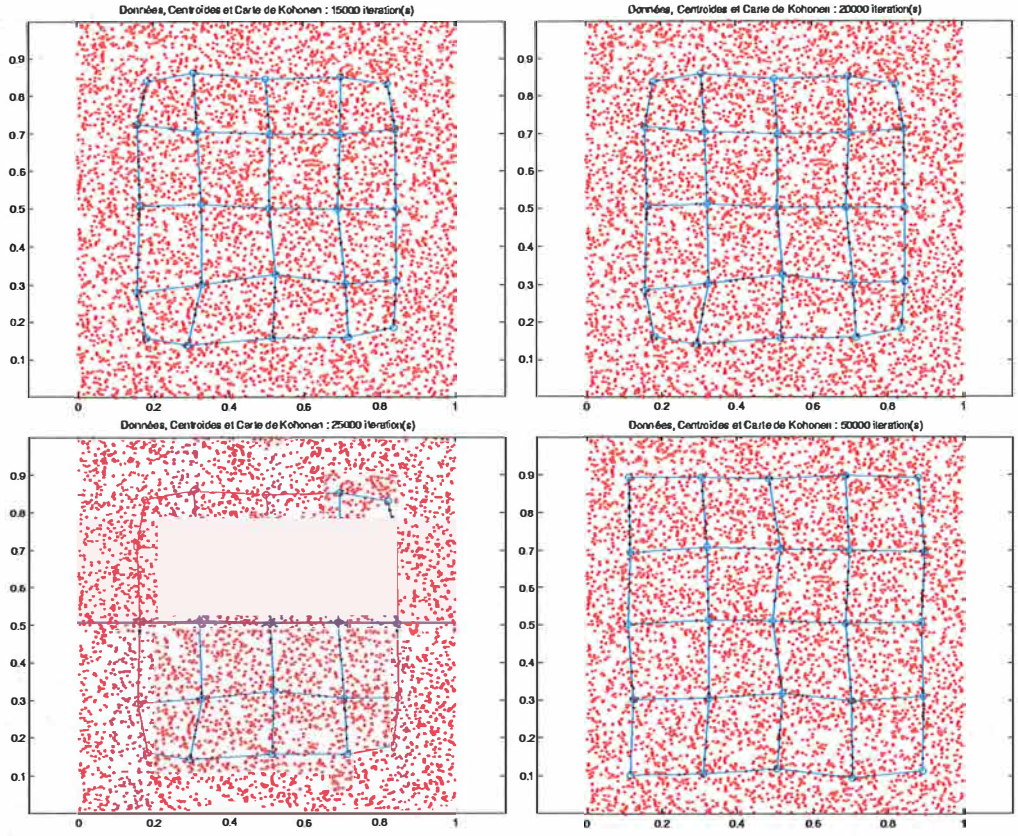
La simulation suivante présente le cas des paramètres variables. La distance de voisinage $distmax(t)$ varie de 2 à 0 voisin(s), en décroissant de façon linéaire, alors que le taux d'apprentissage $\alpha(t)$ diminue de 0,02 à 0,001 suivant une loi exponentielle. Une carte carrée de 25 neurones (5x5) est utilisée ici aussi.

Les figures ci-dessous représentent la forme de la carte de Kohonen lors des mêmes étapes que dans le cas des paramètres constants. Les neurones sont initialisés aux mêmes valeurs que lors du cas précédent.




Données, Centroides et Carte de Kohonen : 50 iteration(s)
Données, Centroides et Carte de Kohonen : 100 iteration(s)
Données, Centroides et Carte de Kohonen : 200 iteration(s)
Données, Centroides et Carte de Kohonen : 400 iteration(s)
Données, Centroides et Carte de Kohonen : 600 iteration(s)
Données, Centroides et Carte de Kohonen : 800 iteration(s)
Données, Centroides et Carte de Kohonen : 1000 iteration(s)
Données, Centroides et Carte de Kohonen : 1500 iteration(s)


Données, Centroïdes et Carte de Kohonen : 2000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 2500 iteration(s)
Données, Centroïdes et Carte de Kohonen : 3000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 4000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 5000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 6000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 8000 iteration(s)
Données, Centroïdes et Carte de Kohonen : 10000 iteration(s)



Lors de l'application de cette seconde version de l'algorithme, les mêmes étapes peuvent être observées :

- La carte se replie sur elle-même et se retourne, alors que les neurones se redéploient les uns par rapport aux autres. (représentations des itérations 1 à 200)
- La carte se déplie, et les neurones se répartissent à l'intérieur de la distribution des points. (représentations des itérations 400 à 1 000)
- La carte tourne dans la distribution tout en continuant de s'étendre. (représentations des itérations 1 000 à 5 000)
- La carte continue de s'étirer, les neurones situés à chacun des quatre coins se déplaçant toujours plus loin, vers le bord de la distribution. (représentations des itérations 5 000 à 25 000)

Il est légitime de se demander jusqu'à quel moment il est utile de poursuivre la présentation de l'ensemble des points à la carte. Précédemment, nous avions choisi de nous arrêter après 5 présentations de tous les points. Cependant, en observant la carte après 10 présentations, on constate ici que la convergence est meilleure, dans la mesure où la bande de points où aucun neurone ne se trouve est plus petite. Par conséquent, nous utiliserons toujours un minimum de 10 présentations de l'ensemble des points, dans toutes les simulations qui vont suivre.

B.2.3 Lois de décroissance

Une autre question intéressante à soulever est le choix de la loi de décroissance utilisée pour faire varier les paramètres « taux d'apprentissage » et « distance de voisinage » de l'algorithme. En effet, de nombreuses fonctions mathématiques sont décroissantes !

Afin de répondre à cette question, observons l'impact de l'utilisation de différentes lois de décroissance sur la qualité de la convergence de la carte. Nous proposons d'examiner trois types de décroissance, et ce pour chacun des deux paramètres variables.

En utilisant les notations :

- x_0 : la valeur initiale du paramètre,
- x_i : la valeur du paramètre lors de la i^{ème} itération,
- x_n : la valeur finale du paramètre,
- i : la i^{ème} itération, en cours,
- n : le nombre total d'itérations,

nous pouvons donner les équations définissant les décroissances :

- linéaire :

$$x_i = x_0 + i * \frac{x_n - x_0}{n} ,$$

- exponentielle :

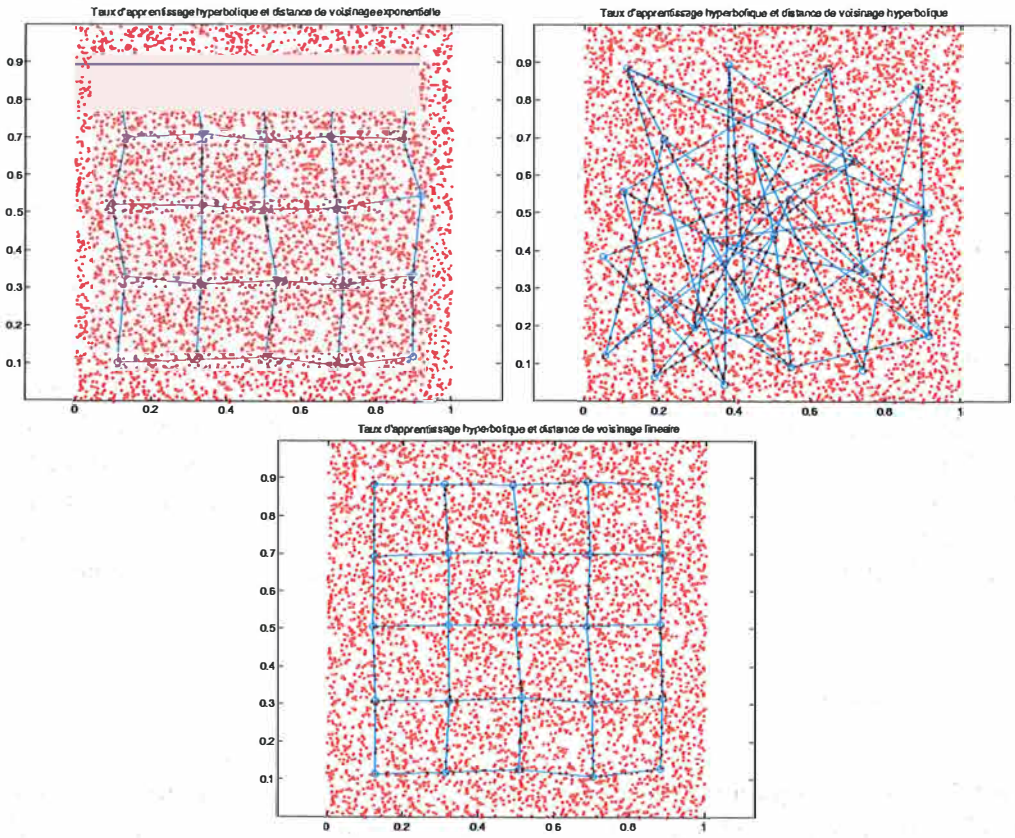
$$x_i = x_0 * (\frac{x_n}{x_0})^{\frac{i}{n}} ,$$

- hyperbolique :

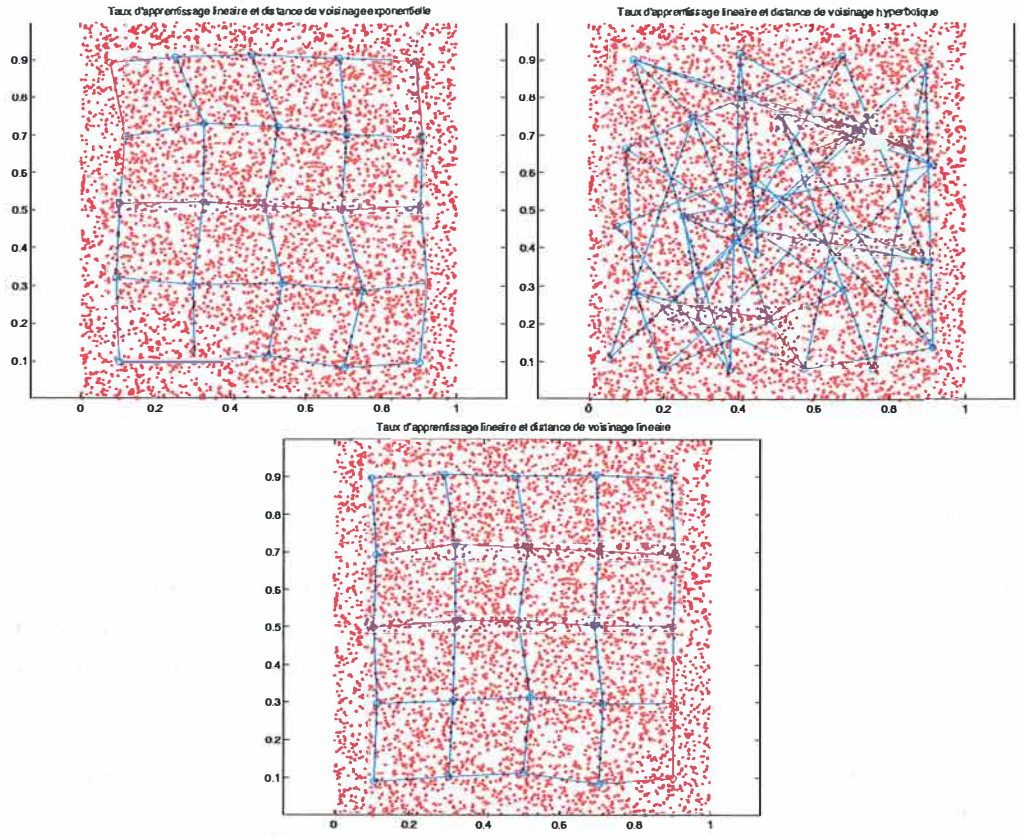
$$x_i = x_0 * x_n * \frac{(\frac{n}{x_0 - x_n})}{i + x_n * (\frac{n}{x_0 - x_n})} .$$

Les figures ci-dessous reprennent les neuf cas possibles. Pour des raisons évidentes de place, les cartes ne sont plus présentées qu'après convergence. Toutes ces simulations utilisent 25 neurones sur une carte carrée (5x5), une distance de voisinage qui varie de 2 à 0, un taux d'apprentissage qui varie 0,02 à 0,001. On présente 10 fois les points au réseau (50 000 itérations). Seul le choix de la fonction décroissante varie d'une simulation à l'autre.

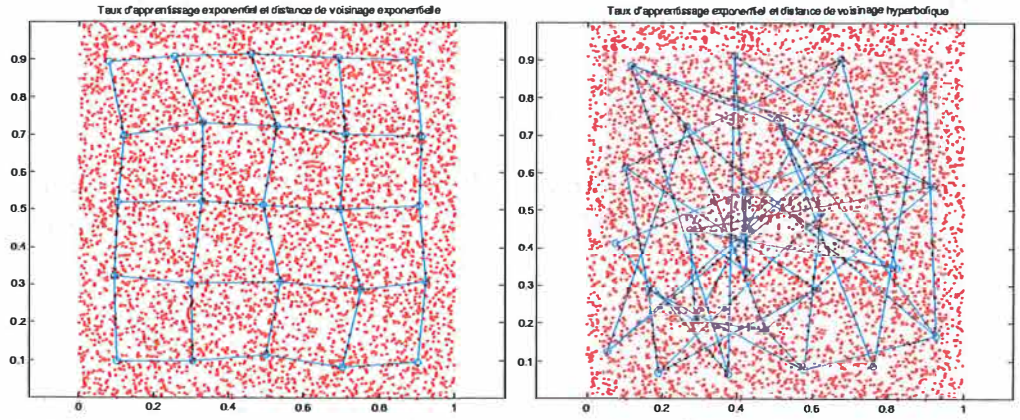
Taux d'apprentissage hyperbolique :

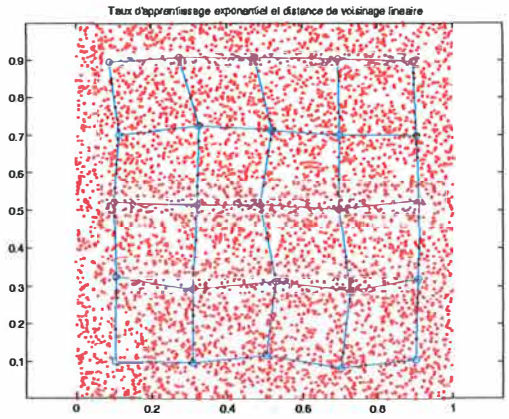


Taux d'apprentissage linéaire :



Taux d'apprentissage exponentiel :





L'utilisation d'une décroissance de type hyperbolique, pour la distance de voisinage, n'est assurément pas une bonne idée ... Si on regarde les figures obtenues, les cartes n'ont pas convergé, même si les neurones se sont plus ou moins correctement répartis dans la distribution. Autrement dit, seul l'aspect quantification vectorielle de l'algorithme a réellement joué. Notons toutefois que les neurones sont un peu plus nombreux dans la partie inférieure de la distribution, ce qui peut se comprendre vu leur position initiale respective.

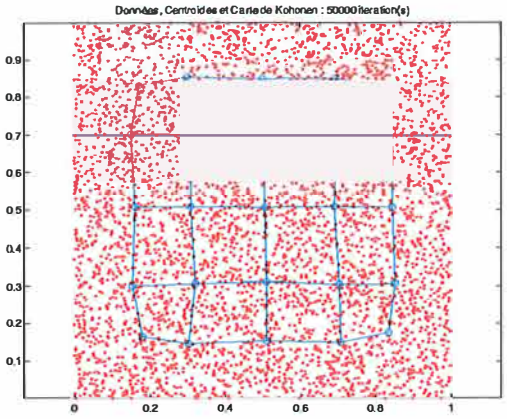
Concernant les autres exécutions, les résultats obtenus sont comparables, à quelques petites différences près : la bande extérieure de points où il n'y a pas de neurones est parfois un peu plus étroite, parfois un peu plus large ; les neurones sont légèrement plus proches de régions où la densité des points est un peu plus élevée ; ... Bref tout un ensemble de petites différences locales font que certaines cartes résument un tout petit mieux que d'autres l'information contenue dans l'ensemble des données utilisé. Malgré ces petites différences, la forme des cartes est pratiquement toujours la même. Les lois de décroissance n'ont donc pas une très grande influence, mais elles permettent d'affiner le résultat d'un point de vue local. L'impact se situe donc au niveau de la convergence finale dont la qualité peut être observée localement.

B.3 Importance relative de chacun des paramètres

Observons rapidement l'impact de chacun des paramètres sur la qualité de la convergence de la carte, en considérant comme référence l'application de l'algorithme dans le cas des paramètres variables.

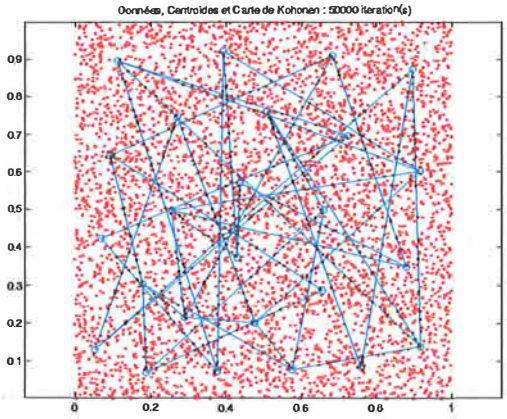
B.3.1 Distance de voisinage

Présentons une première simulation, avec comme paramètres : 25 neurones, un taux d'apprentissage diminuant exponentiellement de 0,02 à 0,001, une distance de voisinage diminuant linéairement de 2 à 1 :

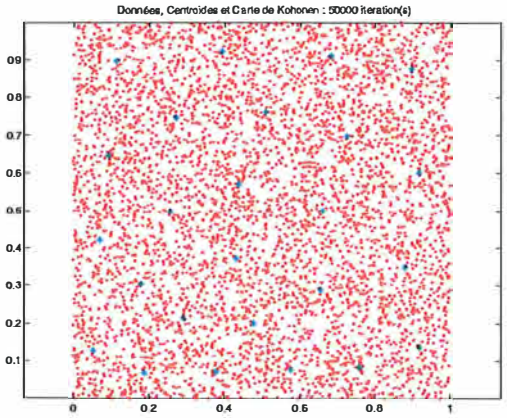


Si la carte a effectivement convergé, il est également clair qu'elle ne s'est pas complètement dépliée dans la distribution. Intuitivement, dans le cas d'un voisinage non nul, les neurones à l'intérieur de la carte ont tendance à tirer à eux les autres neurones, ce qui augmente la largeur de la bande de points sans aucun neurone, aux bords de la distribution.

Prenons donc le point de vue inverse : on va conserver un voisinage nul durant toute l'exécution de l'algorithme. Les paramètres sont donc les mêmes que ci-dessus, excepté pour le voisinage qui est nul et constant, du début à la fin :



Cette fois, la carte n'a pas convergé. Toutefois, la quantification vectorielle des points est effective. Elle peut être observée en retirant les liens entre neurones :



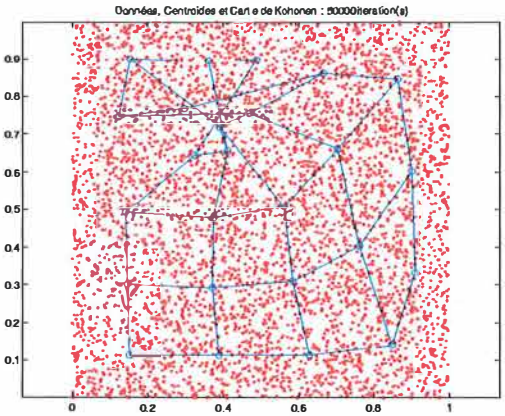
Les neurones se sont répartis à l'intérieur de la distribution, plus ou moins équitablement, avec un nombre légèrement plus important en-dessous qu'au-dessus, ce qui peut s'expliquer

une fois encore par leur position initiale respective, toujours identique à celles des exemples précédents.

Retenons donc que la distance de voisinage doit être non nulle au début de l'algorithme, doit décroître de sorte qu'à la fin, l'exécution se termine avec un voisinage nul.

B.3.2 Taux d'apprentissage

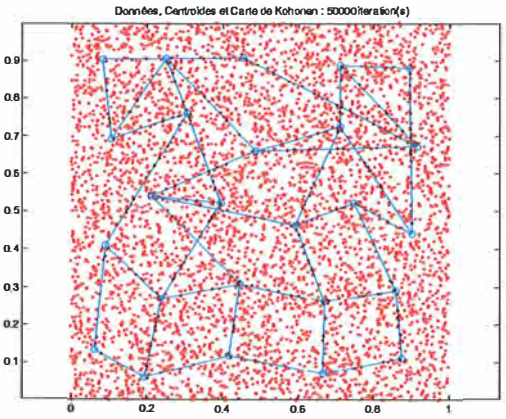
Observons la convergence pour les paramètres suivants : 25 neurones, un taux d'apprentissage variant exponentiellement de 0,002 à 0,001 et une distance de voisinage décroissant linéairement de 2 à 0 :



On voit directement que la carte n'a pas convergé. En effet, elle n'a pas pu se retourner complètement, ce qui se comprend facilement : étant donné que le taux d'apprentissage, facteur représentant l'importance de la modification de la position des neurones, a une faible valeur, les modifications des positions des neurones sont elles aussi faibles.

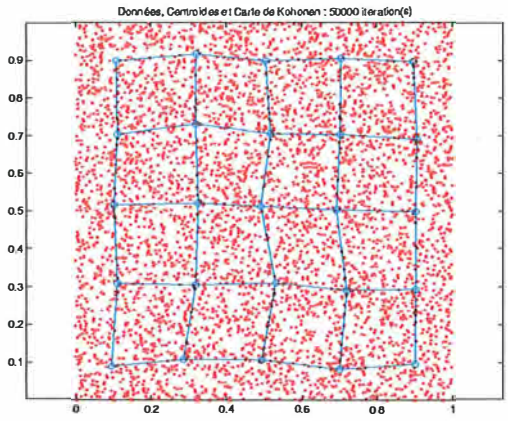
Il est donc préférable d'utiliser des valeurs plus importantes, pour « secouer » la carte, et permettre ainsi aux neurones de se placer les uns par rapport aux autres.

Observons donc le cas d'un taux d'apprentissage plus important, variant de 0,5 à 0,1, de façon exponentielle, les autres paramètres étant les mêmes que ci-dessus :



Si les neurones ont pu se déplacer suffisamment, il y a ici un autre problème : les changements de position des neurones sont ici trop importants, vu qu'en fin d'exécution, le taux d'apprentissage est encore de 0,1. La carte ne se stabilise donc pas, et chaque itération provoque un déplacement assez important du centroïde vainqueur.

Il faut donc terminer avec un taux d'apprentissage plus faible, par exemple un taux d'apprentissage variant de 0,5 à 0,001, en utilisant les mêmes valeurs pour les autres paramètres que lors de la simulation précédente :



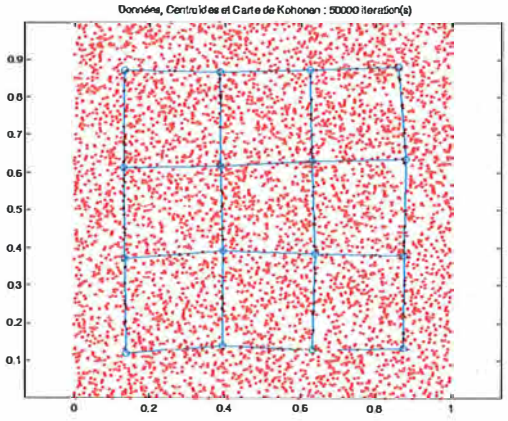
Nous obtenons enfin une « belle » convergence, la carte recouvrant correctement l'ensemble de la distribution.

Retenons que le taux d'apprentissage doit être élevé en début d'exécution de l'algorithme, pour « secouer » la carte et permettre aux neurones de se déplacer les uns par rapport aux autres, diminuer relativement vite pour que ce déplacement important des neurones ne perdure pas, et enfin terminer avec une valeur faible, pour assurer une bonne convergence locale, sans induire de trop grands changements dans la position des neurones.

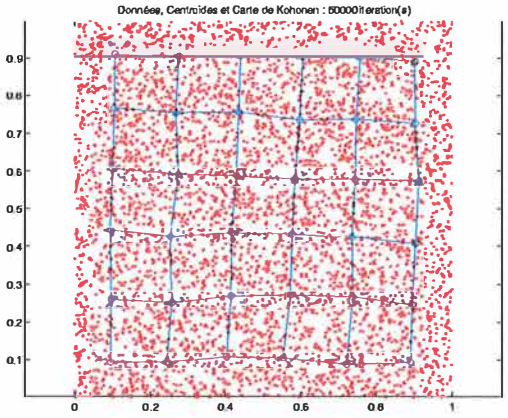
B.3.3 Le nombre de neurones

Voici quelques exemples de simulations, avec à chaque fois une distance de voisinage décroissant linéairement et un taux d'apprentissage décroissant exponentiellement de 0,02 à 0,001. Les autres paramètres utilisés sont précisés à chaque fois :

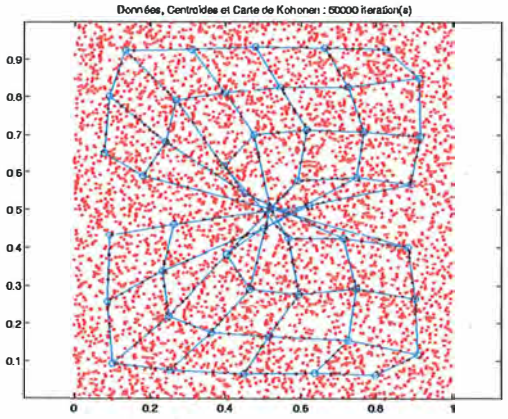
16 neurones, distance de voisinage diminuant de 2 à 0 :



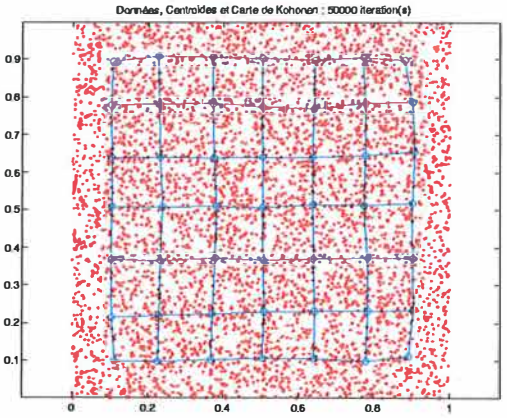
36 neurones, distance de voisinage allant de 2 à 0 :



49 neurones, distance de voisinage variant de 2 à 0 :

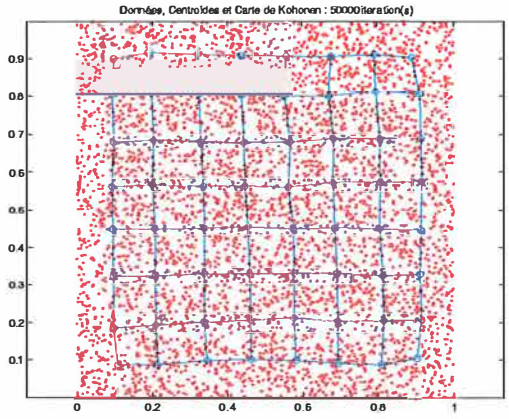


La convergence est mauvaise ... En changeant les valeurs des paramètres : 49 neurones, distance de voisinage entre 3 et 0 :

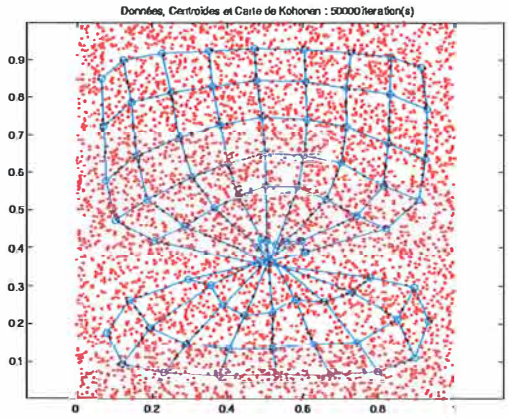


La convergence est nettement meilleure. Il y aurait donc une relation entre le nombre de neurones sur le côté de la carte et la distance de voisinage ?

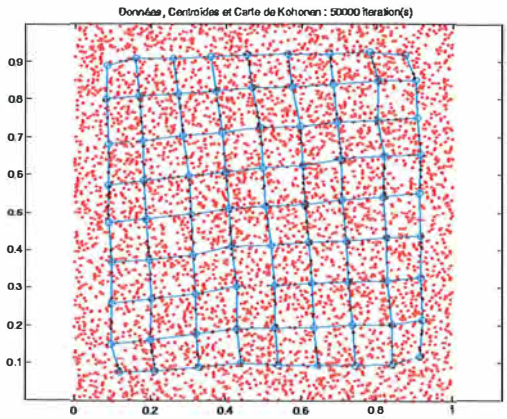
Effectuons une autre simulation, avec comme paramètres : 64 neurones, distance de voisinage variant de 3 à 0 :



Nous avons augmenté le nombre de neurones, sans augmenter la distance de voisinage, et nous avons pourtant une convergence ! Continuons : 81 neurones, distance de voisinage variant entre 3 et 0 :



Le même problème apparaît. Il nous faut donc à nouveau changer les paramètres : 81 neurones, distance de voisinage variant de 4 à 0 :



Et la carte converge convenablement.

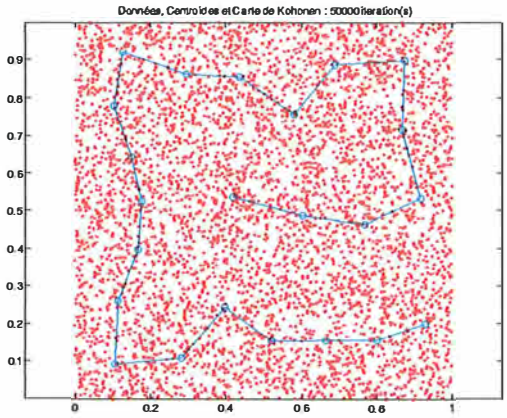
Inutile de multiplier les exemples. Les illustrations ci-dessus nous ont permis de mettre clairement en évidence l'existence d'une relation entre le nombre de neurones, dans ce cas-ci sur le côté de la carte carrée, et la distance de voisinage initiale.

B.3.4 La forme de la carte

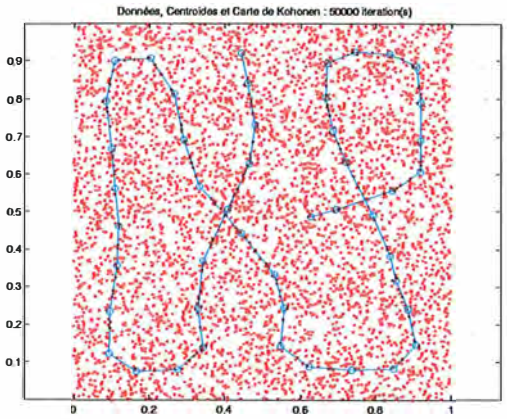
Le chapitre 2, section 2.3.3 d), présente les trois formes les plus utilisées des cartes de Kohonen : les ficelles (1 dimension), les cartes carrées et rectangulaires (2 dimensions). Illustrons à présent ces différents types de cartes. Notons qu'à partir d'ici, les initialisations des neurones au début de l'algorithme se basent sur les valeurs respectives d'un certain nombre (autant que de neurones) de données contenues dans l'ensemble. Ces quelques données sont choisies de façon aléatoire.

B.3.4.1 Les cartes à une dimension

Voici une ficelle de Kohonen avec 23 neurones, un taux d'apprentissage variant exponentiellement de 0,02 à 0,001, et une distance de voisinage variant de 2 à 0, linéairement :

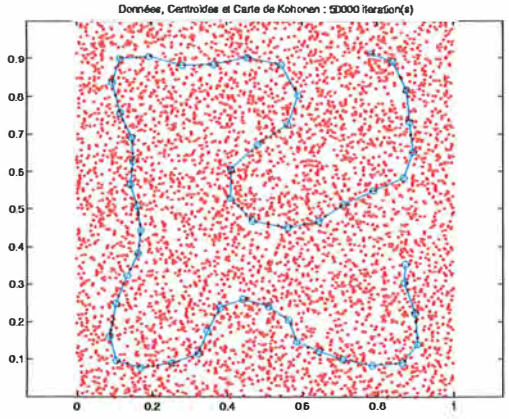


On peut bien entendu faire varier le nombre de neurones : 47 neurones, distance de voisinage décroissante linéairement de 2 à 0, taux d'apprentissage décroissant exponentiellement de 0,02 à 0,001 :

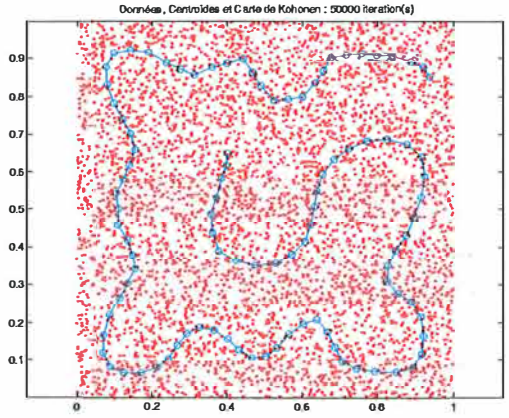


Cet exemple nous montre qu'ici aussi, il faut faire attention à la distance de voisinage.

Prenons par exemple 51 neurones, une distance de voisinage variant linéairement de 5 à 0, et un taux d'apprentissage variant exponentiellement de 0,02 à 0,001 :



En augmentant encore : 101 neurones, une distance de voisinage décroissant selon une loi linéaire de 6 à 0, et un taux d'apprentissage décroissant selon une loi exponentielle de 0,025 à 0,001

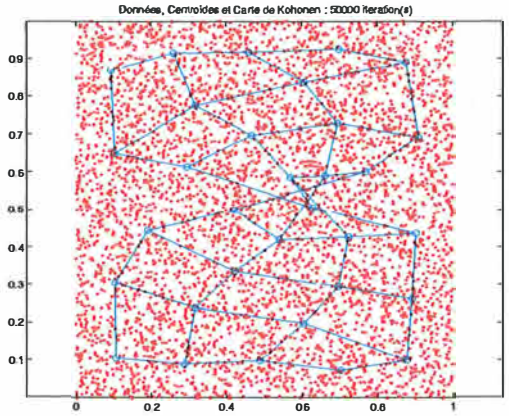


Notons au passage que le taux d'apprentissage a également été revu légèrement à la hausse pour donner une meilleure convergence.

B.3.4.2 Les cartes à deux dimensions

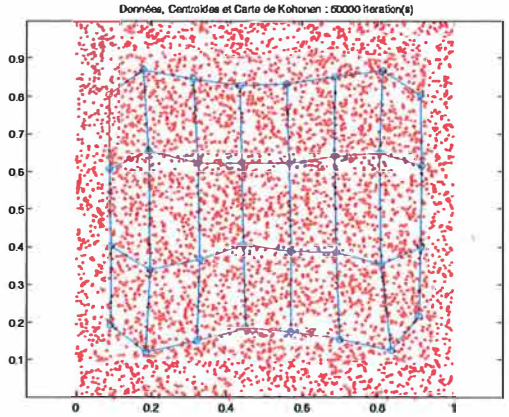
Nous avons déjà présenté en détail le comportement des cartes carrées, tous les premiers exemples comportant 25 centroïdes répartis sur une carte 5x5. Ne sera donc présenté ici que le cas des cartes rectangulaires.

Considérons les paramètres suivants : 32 neurones, distance de voisinage décroissant linéairement de 2 à 0, taux d'apprentissage décroissant exponentiellement de 0,02 à 0,001 :

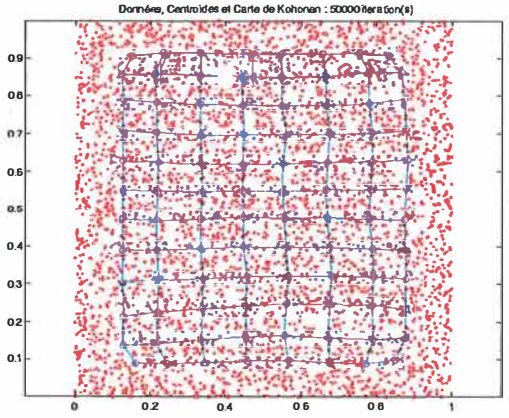


Ici aussi, le problème de la distance de voisinage, trop faible, se présente et ce, d'autant plus tôt que les longueur et largeur de la carte sont différentes.

En prenant une distance de voisinage variant plutôt de 3 à 0, tout le reste étant identique :



En augmentant à nouveau le nombre de neurones et la distance de voisinage : 96 neurones, distance de voisinage linéairement décroissante de 8 à 0, taux d'apprentissage exponentiellement décroissant de 0,02 à 0,001 :

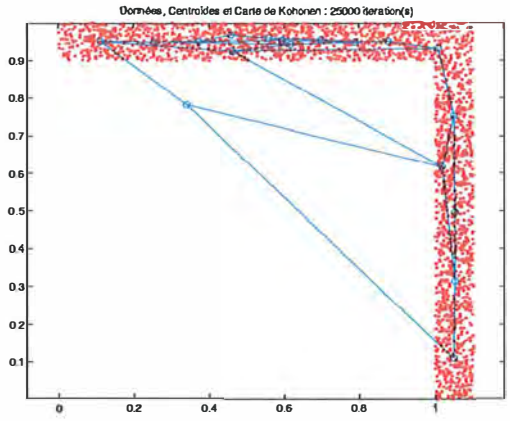


Retenons que la distance de voisinage est fonction du nombre de neurones sur le côté de la carte ; la longueur de ce côté étant fonction de la forme de la carte utilisée.

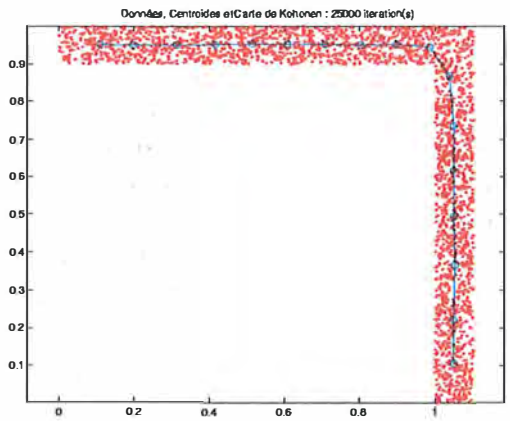
B.3.4.3 Quelle carte utiliser ?

Pour répondre à cette question, considérons un autre exemple. Nous utilisons ici une distribution de points en deux segments situés en angle droit l'un par rapport à l'autre. Cette distribution comporte 2500 points, qui seront présentés 10 fois à la carte. Observons le comportement d'une carte carrée et d'une ficelle.

Dans le cas de la carte carrée, les paramètres utilisés sont : 18 neurones, taux d'apprentissage exponentiellement décroissant, compris entre 0,02 et 0,001, distance de voisinage allant de 3 à 0 selon une loi linéaire et décroissante :



La convergence est très mauvaise (il y a même un neurone mort), ce qui est compréhensible vu la forme de la distribution. Si on utilise à la place une ficelle, avec comme paramètres 17 neurones, un taux d'apprentissage variant exponentiellement de 0,02 à 0,001, et une distance de voisinage variant linéairement de 3 à 0 :



Nous pouvons maintenant répondre à la question de départ : la forme de la carte à utiliser est fonction des données sur lesquelles on applique l'algorithme de Kohonen. Lorsque les données sont en deux ou trois dimensions, le choix est relativement aisé puisqu'il suffit de visualiser les données pour choisir. Mais lorsque les données sont de dimension supérieure, il devient alors important de prendre le temps de regarder les données dont on dispose, pour apprendre à les connaître et les comprendre, en se faisant une intuition de ce qu'elles sont. Le choix de la carte sera plus judicieux si on se donne le temps de bien observer les données à notre disposition.

B.3.5 Note sur la valeur des paramètres

Notre description des paramètres ne pouvait se terminer sans une note reprenant un point de vue plus pratique.

Remettons les cartes de Kohonen dans le cadre de l'utilisation qui va en être faite lors de la suite de ce mémoire. Les cartes de Kohonen sont utilisées en tant qu'outil de quantification vectorielle, nécessaire au niveau de la méthode décrite au chapitre 3 qui nous permet de décrire l'évolution d'une série de données temporelles. Cette méthode se sert abondamment de l'algorithme de Kohonen en l'appliquant sur une série de sous-ensembles construits à partir de la base de données initiale.

Vu le nombre de fois où cet algorithme va être employé, nous avons jugé intéressant de fixer une fois pour toutes certains paramètres, en vue d'accélérer l'exécution de la méthode sans devoir continuellement demander à l'utilisateur les valeurs qu'il souhaite affecter aux différents paramètres.

Plus concrètement, l'étude des cartes de Kohonen ci-dessus nous a permis de décrire les variables suivantes :

- la distance de voisinage,
- le taux d'apprentissage,
- le nombre de neurones,
- la forme de la carte.

Pour rappel, il a été choisi d'initialiser les neurones sur base de valeurs à l'intérieur de la base de données. De plus, nous avons pu constater, par exemple, que l'utilisation d'une loi exponentielle ou hyperbolique pour le taux d'apprentissage n'avait pas beaucoup d'impact sur la qualité de la convergence. Nous choisissons donc de fixer également ces lois de décroissance comme suit :

- loi de décroissance exponentielle pour le taux d'apprentissage,
- loi de décroissance linéaire pour la distance de voisinage.

Ces deux lois seront toujours utilisées dans le reste de ce travail.

Il nous reste les quatre paramètres principaux de la carte. Peuvent-ils, eux aussi, être fixés une fois pour toutes ? Oui et non. Oui parce certains d'entre eux sont liés, non parce qu'il n'y a alors plus de sens à appliquer un algorithme où tout est définitivement fixé.

Nous avons pu mettre en évidence, dans notre étude des cartes, un lien entre le nombre de neurones et la distance de voisinage. De même, la forme de la carte et la distance de voisinage sont liées. Par contre, le taux d'apprentissage semble être indépendant des autres valeurs. Ces informations peuvent être codées dans l'algorithme de Kohonen.

Plus simplement, nous avons décidé de fixer le taux d'apprentissage à une valeur suffisamment élevée, soit 0,5, de telle sorte que la décroissance se termine en 0,01. Le but est de laisser le temps à la carte de se déplier puis de laisser jouer l'effet de quantification vectorielle lorsque le nombre de voisins diminue.

La distance de voisinage est également codée dans l'algorithme en fonction de la forme de la carte et du nombre de neurones présents sur la carte. Si nous notons :

- n : le nombre de neurones présents sur la carte et
- $distmax$: la distance de voisinage au début de l'algorithme,

nous pouvons écrire les équations définissant la distance de voisinage en fonction de la forme de la carte :

- ficelle : $distmax = n/2$,
- carte carrée : $distmax = \sqrt{n}$,
- carte rectangulaire : $distmax = \lceil \sqrt{n} \rceil$.

La distance de voisinage décroît donc linéairement de *distmax* à 0.

En conclusion, il ne reste que deux paramètres à l'algorithme de Kohonen tel que nous l'avons implémenté : le nombre de neurones et la forme de la carte. Les autres valeurs sont fixées de sorte qu'elles nous assurent une bonne convergence.

Bibliographie

[1] *Handbook Of Neural Computation.* IOP Publishing Ltd and Oxford University Press, 1997.

[2] Cottrell M. de Bodt E. Grégoire P. Simulating interest rate structure evolution on a long term horizon, a kohonen map application. *Proceedings of Neural Networks in The Capital Markets*, 1996. Californian Institute of Technology, World Scientific Ed., Passadena.

[3] Cottrell M. de Bodt E. Grégoire P. Simulation de l'évolution de la structure à terme des taux d'intérêt : une approche paramétrique. *Banque & Marchés, n° 36*, pages 21–28, Septembre-Octobre 1998.

[4] Snedecor G. W. Cochran W. G. *Statistical methods.* The Iowa State University Press, sixth edition edition, 1972.

[5] Dayhoff Judith. *Neural Network Architecture : An introduction.* Van Nostrand Reinhold, 1990.

[6] A. Lendasse, M. Verleysen, E. de Bodt, P. Gregoire, and M. Cottrell. Forecasting time-series by Kohonen classification. In M. Verleysen, editor, *Proceedings of ESANN'98*, pages 221–226, Bruxelles, 1998. Editions D Facto.

[7] Ljung Lennart. *System Identification, Theory for the User.* Prentice Hall, 2nd edition edition, 1999.

[8] Blayo F. Verleysen M. *Les Réseaux de Neurones Artificiels.* Que sais-je? Presses Universitaires de France, 1996.

[9] Weigend A. Gershenfeld N. *Time series prediction, Forecasting the future and understanding the past.* Addisson Wesley Publisher, 1994.

[10] Wasserman Philip. *Neural Computing : Theory and Practice.* Van Nostrand Reinhold, 1989.

[11] Kohonen Teuvo. *Self-Organizing Maps*, volume Volume 30 of *Springer series in information sciences.* Springer, 2nd edition, 1997.

[12] van der Waerden B. L. *Mathematical statistics.* Springer-Verlag, 1969.

[13] von der Malsburg C. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14 :85–100, 1973.

[14] Page reprenant tous les fichiers de la competition Santa Fe. `http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html`.

[15] Fichiers de la compétition Santa Fe, serie A. `http://www.ece.ogi.edu/~ericwan/data.html`.

[16] Modèles neuronaux appliqués à la série Santa Fe A. `http://www.auto.ucl.ac/~lendasse/pdf/ohio.ppt`.

Description du contenu du CD

Répertoire « Texte » :

- un fichier contenant le texte de ce mémoire, au format PostScript : **memoire.ps**

Répertoire « Code source » :

- un sous répertoire « C++ », contenant tout le code développé dans le cadre de ce mémoire. Les librairies pour la manipulation de matrices ont été implémentées par M. John A. Lee. Merci de le tenir informé de toute utilisation de son code : **lee@dice.ucl.ac.be**.
- un sous répertoire « Matlab », contenant les scripts écrits pour la génération des résultats graphiques.