Table des matières

Liste des tableaux

Table des figures

Introduction

Les réseaux de neurones artificiels sont devenus en quelques années des outils précieux dans des domaines très divers, tels que la robotique, les industries, l'économie, les banques, la vie sociale...[1]

Néanmoins, ils n'ont pas encore atteint leur ultime développement pour des raisons plus psychologiques que techniques, liées aux connotations biologiques du terme et au fait qu'ils sont considérés, à tort, comme des outils d'intelligence Artificielle[2]. Or l'intérêt des réseaux de neurones, dans le domaine des Sciences ne doit rien à la métaphore biologique, il est uniquement dû aux propriétés mathématiques spécifiques de ces réseaux. Les réseaux de neurones ont d'abord été développés pour résoudre des problèmes de contrôle, de reconnaissance de formes ou de mots, de décision, de mémorisation comme une alternative à l'intelligence artificielle, et en relation plus ou moins étroite avec la modélisation de processus cognitifs (capacité de connaître ou faire connaître) réels et des réseaux de neurones biologiques[4].

Auparavant, Newton et Leibniz ont inventé une méthode de prédiction dynamique. Newton l'appliquait avec succès au mouvement des planètes et de leurs satellites. Depuis, elle est devenue la grande méthode de prédiction des mathématiques appliquées. Sa portée est universelle. Tout ce qui est matériel, tout ce qui est en mouvement, peut être étudié avec les outils de la théorie des systèmes dynamiques[19]. Après, Einstein a mis en exergue quelques effets inexpliqués par la loi de Newton. Il a prédit les nouveaux effets de la gravitation, les effets de la lentille optique gravitationnelle et l'effet de la gravitation sur le temps dans la théorie de la relativité générale. On trouve aussi la prédiction dans de nombreux domaines comme l'industrie (exemple : asservissement des machines), l'économie (exemple : prédiction de la croissance économique)[23].

Une série temporelle, ou série chronologique, est une suite de valeurs numériques représentant l'évolution d'une quantité spécifique au cours du temps[11]. De telles suites de variables aléatoires peuvent être exprimées mathématiquement afin d'en analyser le comportement, généralement pour comprendre son évolution passée et pour en prévoir le comportement futur. Dans ce travail, nous nous intéressons à la série générée par le système de Lorenz, et nous essayons de le caractériser en faisant des prédictions à l'aide des réseaux de neurones artificiels[12].

Notre travail se divise en trois chapitres. Nous voyons dans le premier chapitre, l'étude des réseaux de neurones artificiels, les théories de base et l'algorithme d'optimisation. Le chapitre 2 est consacré à l'étude des systèmes dynamiques qui sont la base du système de Lorenz. Et dans le troisième chapitre les résultats des différentes prédictions montrant le caractère chaotique du système sont établis. Le langage c# est utilisé dans le traitement des données.

Chapitre 1

Réseaux de neurones artificiels

1.1 Historique

En 1943, les neurologues Warren Sturgis Mc Culloch et Walter Pitts ont mené les premiers travaux sur les réseaux de neurones à la suite de leur article fondateur : "What the frog's eye tells to the frog's brain"[2]. Ils ont modélisé le neurone biologique par un comportement booléen en ayant constitué un modèle simplifié de neurones biologiques appelé neurone formel. Ils ont montré théoriquement que les neurones formels simples pouvaient réaliser des fonctions logiques, arithmétiques et symboliques complexes. Ainsi, le neurone artificiel effectue un automate binaire qui réalise une somme pondérée de stimulis 'S' provenant d'autres neurones. Si cette somme est supérieure à une valeur seuil 'B_0' donné, alors le neurone est activé, sinon il ne transmet aucune information et ceci selon la fonction suivante :

. Si $S > B_0$, la sortie vaut 1 et le neurone est actif,

. Si $S < B_0$, la sortie vaut -1 et le neurone est inactif.

En 1949, le neurophysicien Hebb a établi le couplage synaptique d'apprentissage ayant eu un fondement biologique et a stipulé que : " Si deux neurones sont activés simultanément, alors la valeur des poids des connexions entre ces neurones est augmentée, sinon les connexions restent inchangées".

Le premier succès était apparu en 1957 quand Frank Rosenblatt a inventé le premier modèle artificiel nommé le « perceptron ». C'était le premier système qui pouvait apprendre par expérience, y compris, lorsque son instructeur a commis des erreurs. Il a construit le premier neuro-ordinateur basé sur ce modèle et l'a appliqué au domaine de la reconnaissance de forme du système visuel. Notons qu'à cette époque les moyens à sa disposition étaient limités et cela a été une prouesse technologique d'avoir pu réussir à faire fonctionner correctement cette machine pendant quelques minutes.

En 1969, M. Minsky et S. Papert ont publié un ouvrage qui a mis en exergue les limitations théoriques du perceptron. Ces chercheurs ont analysé, sous l'angle mathématique, ses performances et ont trouvé qu'il était incapable de résoudre la séparation pour l'opération logique « ou exclusif » et qu'en conséquence ce modèle ne présente aucun intérêt. Les limitations concernaient notamment l'impossibilité de traiter par ce modèle des problèmes non linéaires .Ils ont étendu implicitement ces limitations à tous modèles de réseaux de neurones artificiels. Leur objectif étant atteint, il y avait eu abandon financier des recherches dans le domaine surtout aux USA, les chercheurs se tournaient principalement vers l'Intelligence Artificielle et les systèmes à base de règles.

La découverte, en 1985, de l'algorithme de rétropropagation du gradient énoncé par Rumelhart et al. et celle de la nouvelle génération de réseau de neurones[7,8], le perceptron multicouche proposé par Werbos ont permis de lever toutes les limitations énoncées par Minsky et Papert, d'où le regain d'intérêt pour les réseaux de neurones. En effet, cet algorithme reste le modèle le plus étudié et le plus productif au niveau des applications (reconnaissance de la parole, reconnaissance de forme, vision artificielle, aide à la décision).

À partir de ce moment, la recherche sur les réseaux de neurones connaît un essor fulgurant et, au cours des années 90, les applications commerciales de ce succès académique se succèdent.

1.2 Origine neurobiologique des Réseaux de neurones artificiels

1.2.1 Le neurone :

Les réseaux de Neurones Artificiels ont pour origine un modèle de neurone biologique dont il ne garde d'ailleurs qu'une vision simplifiée. Le cerveau humain est composé d'un grand nombre de cellules nerveuses appelées ***neurones***, avec 10^3 à 10^4 connexions.
Un neurone est formé :

- d'un réseau convergent d'entrée appelé dendrites qui constituent la principale surface de réception du neurone.
- d'un élément de traitement appelé corps cellulaire ou soma qui contient le noyau du neurone et la machinerie biochimique nécessaire à la synthèse des enzymes et des autres molécules essentielles à la vie de la cellule.
- d'un réseau divergent de sortie appelé axones.

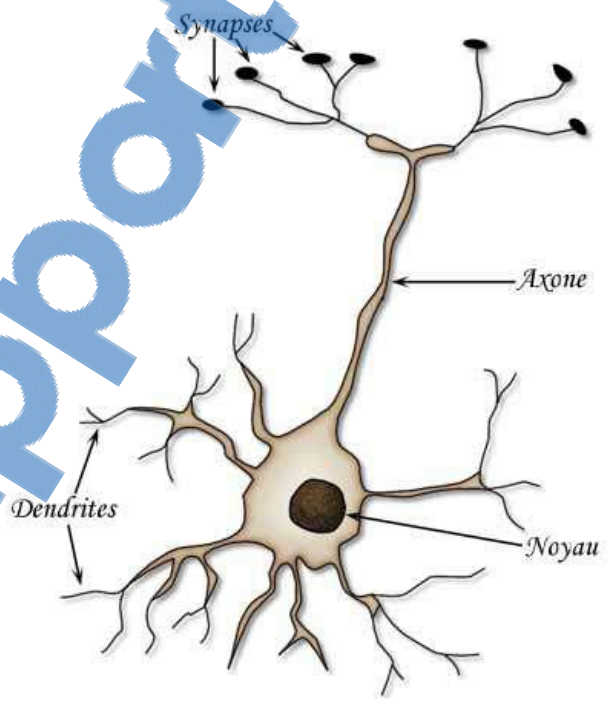

FIGURE 1.1 – Un neurone biologique et ses principaux composants

1.2.2 Traitement de l'information au niveau du cerveau

Le traitement de l'information par un réseau de neurone est de nature électrochimique. Chaque neurone est asservi au maintien d'un gradient électrique d'environ $-70mV$ entre l'intérieur et l'extérieur du neurone. Ainsi, le neurone est dit polarisé. Si l'influence des autres neurones sur le potentiel membranaire suffit pour dépolariser le neurone jusqu'à un certain seuil(environ $\theta = -50mV$) alors le corps cellulaire génère une impulsion électrique du type tout ou rien appelé potentiel d'action. Le potentiel ainsi généré se propage sans amortissement le long de l'axone et de ses ramifications. Quand le potentiel d'action atteint une synapse reliée à un autre neurone, il déclenche une émission chimique qui modifie le potentiel membranaire du neurone récepteur soit de façons excitatrice(dépolarisation) soit de façons inhibitrice (hyperpolarisation). Ainsi chaque neurone fait en permanence l'addition des signaux excitateurs et dépolarisants de ceux inhibiteurs et polarisants reçus par sa membrane et déclenche une impulsion nerveuse lorsque les conditions sont réalisées c'est à dire lorsque le potentiel d'action est atteint.

1.3 Modélisation ou le Neurone formel

Le neurone formel est la composante principale d'un réseau de neurones artificiels. Ils sont dotés de caractéristiques inspirées de celles des neurones biologiques que nous avons passées en revue dans la section précédente :
Le potentiel d'action des cellules nerveuses : il s'agit ici d'une valeur numérique, qui peut être transmise à des neurones en aval. Un neurone formel ne peut transmettre qu'une valeur unique qui correspond à son état d'activation[2].
Les dendrites des neurones biologiques leur permettent de recevoir différents signaux de l'extérieur. De la même manière, un neurone formel peut recevoir des signaux x_i de plusieurs neurones. Ces signaux sont combinés en un signal d'entrée unique.
Les synapses : Les nombres w_{ij} pondèrent les signaux émis par les différents neurones situés en amont où l'on retrouve l'analogue des synapses qui peuvent être inhibitrices ($w_{ij} < 0$), ou excitatrices ($w_{ij} > 0$).

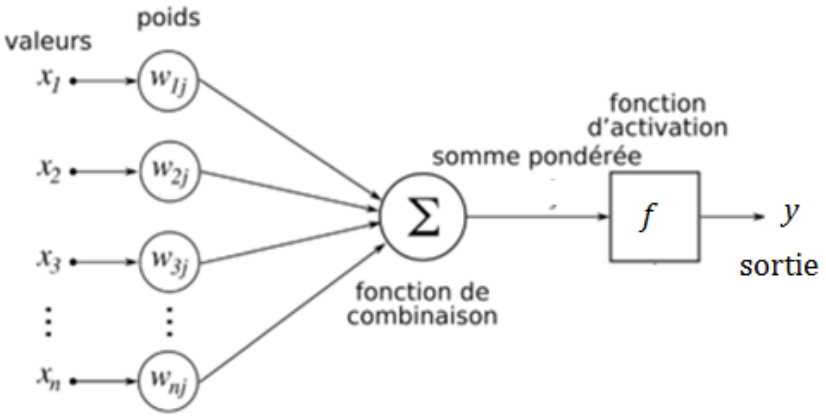

FIGURE 1.2 – Structure d'un neurone artificiel

Les x_i sont des variables d'entrées, les w_{ij} sont des paramètres des poids.

Les entrées peuvent être binaires $(0, 1)$ ou bipolaire $(-1, 1)$.

En règle générale, le calcul de la valeur de cette fonction peut se décomposer en deux étapes :

— Une combinaison linéaire des entrées

$$U = \sum_{i=1}^{n} w_{ij} x_i \tag{1.1}$$

— La sortie du neurone

$$y = f(U) = f\left(U = \sum_{i=1}^{n} w_{ij} x_i\right) \tag{1.2}$$

U est appelé potentiel du neurone.

NEURONE BIOLOGIQUE	NEURONE FORMEL
Dendrite	Signal d'entrée
Synapse	Poids
Soma	Fonction d'activation
Axone	Sortie

TABLE 1.1 – Analogie entre neurone biologique et neurone formel

1.3.1 Fonction de transfert ou fonction d'activation

Une fonction d'activation gère l'état du neurone formel[4], elle est utilisée pour la conversion du résultat de la combinaison linéaire des entrées d'un neurone en une valeur de sortie. La fonction d'activation introduit une non linéarité dans le comportement du neurone.

Voici quelques exemple de fonctions d'activation

- Pour une entrée binaire $(0, 1)$

$$f(x) = \begin{cases} 1 & \text{si} \quad x \geqslant 0 \\ 0 & \text{si} \quad x < 0 \end{cases} \tag{1.3}$$

$$f(x) = \begin{cases} 1 & \text{si} \quad x > 1 \\ x & \text{si} \quad 0 \leqslant x \leqslant 1 \\ 0 & \text{si} \quad x < 0 \end{cases} \tag{1.4}$$

$$f(x) = \frac{1}{1 + \exp^{-\alpha x}} \tag{1.5}$$

avec $\alpha > 0$ et $x \in R$

- Pour une entrée bipolaire $(-1, 1)$

$$f(x) = \begin{cases} 1 & \text{si} \quad x \geqslant 0 \\ -1 & \text{si} \quad x < 0 \end{cases} \tag{1.6}$$

$$f(x) = \begin{cases} 1 & \text{si} \quad x > 1 \\ x & \text{si} \quad -1 \leqslant x \leqslant 1 \\ -1 & \text{si} \quad x < -1 \end{cases} \tag{1.7}$$

$$f(x) = \frac{1 - \exp^{-\alpha x}}{1 + \exp^{-\alpha x}} \tag{1.8}$$

avec $\alpha > 0$ et $x \in R$

$$f(x) = hypertan(x) \tag{1.9}$$

1.4 Réseaux de neurones artificiels

Un Réseau de Neurones Artificiels est un ensemble de neurones formels fortement connectés et fonctionnant en parallèle. Chaque neurone formel calcule une sortie unique sur la base des informations qu'il reçoit.

1.4.1 Architecture générale d'un réseau de neurone artificiel

La structure d'interconnexion entre les neurones formels donne l'architecture d'un réseau de neurone[1]. Cette architecture des réseaux de neurones se divise en deux grandes familles :

— propagation vers l'avant de l'information (feedforward)

— modèle récurent (feedback network)

a) Propagation vers l'avant de l'information (feedforward)

La propagation des activations ou des informations se fait de l'entrée vers la sortie.

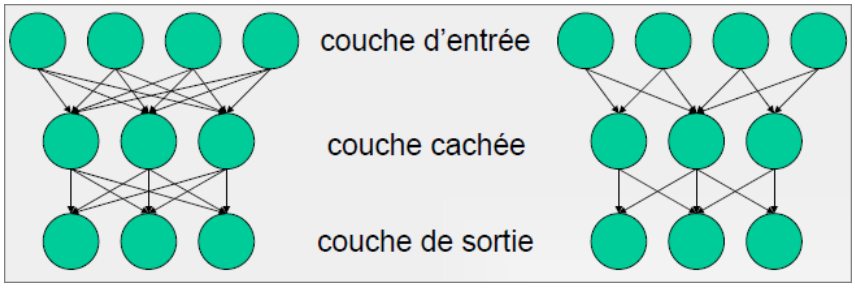

FIGURE 1.3 – Réseau multicouches et multicouches locales

★ **Réseau multicouche :** tous les neurones de la couche précédente sont connectés à tous les neurones de la couche suivante.

★ **Réseau multicouche locales :** tous les neurones de la couche précédente ne sont pas connectés totalement à tous les neurones de la couche suivante.

b) Modèle récurent (feedback network)

La propagation des informations se fait de l'entrée vers la sortie ou vice versa.

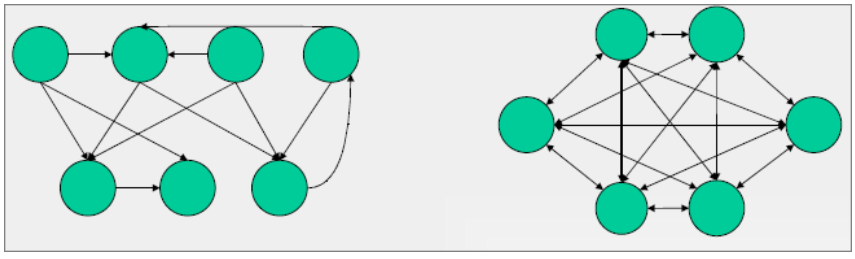

FIGURE 1.4 – Réseaux récurrents ou bouclés

Dans les réseaux bouclés, les connexions entre les neurones forment des boucles. Le réseau doit itérer pendant une longue période avant de produire une réponse. En se déplaçant dans le réseau et en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ.

1.5 Règle d'apprentissage

Comme le cerveau humain, les réseaux de neurones artificiels peuvent apprendre par expérience[8].
L'apprentissage d'un réseau de neurone artificiel consiste à déterminer les poids entre les neurones, puis modifier la valeur des poids jusqu'à l'obtention du comportement désiré.
On distingue deux grandes classes d'algorithme d'apprentissage :

— l'apprentissage supervisé (back propagation)
— l'apprentissage non supervisé

1.5.1 Apprentissage supervisé :

Cet algorithme d'apprentissage ne peut être utilisé que lorsque les combinaisons d'entrées-sorties désirées sont connues à l'avance. L'ajustement des poids se fait directement à partir de l'erreur, soit la différence entre la sortie obtenue par le réseau a_r et la sortie désirée c_r. Le cycle est répété jusqu'à ce que le réseau classe correctement les motifs désirés, c'est-à-dire a_r proche de c_r

1.5.2 Apprentissage non supervisé :

Il n'y a pas de connaissances à priori des sorties désirées pour des entrées données. En fait, c'est de l'apprentissage par exploration où l'algorithme d'apprentissage ajuste les poids des liens entre les neurones de façon à maximiser la qualité de classification des entrées.

1.6 Les réseaux multicouches à propagation de l'information vers l'avant

Ces réseaux sont organisés en couches. La première couche est appelée couche d'entrée, la dernière est appelée couche de sortie et les couches intermédiaires, sont appelées couches cachées. Chaque couche transmet le résultat de son analyse à la couche suivante[7]. L'information donnée au réseau va donc se propager couche par couche, de la couche d'entrée vers la couche de sortie, en passant par une ou plusieurs couches intermédiaires(couches cachées). Pour l'activation des neurones, nous utilisons la fonction sigmoïde et la fonction tangente hyperbolique pour l'entrée bipolaire.

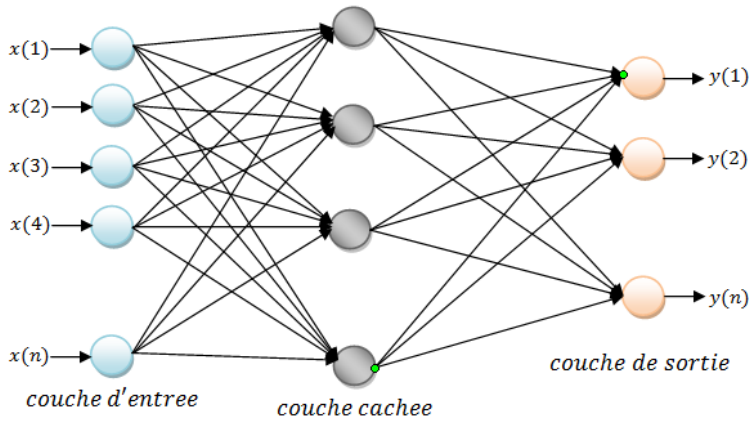

FIGURE 1.5 – Exemple d'un réseau multicouches à propagation de l'information vers l'avant ou perceptron multicouches

1.6.1 Modélisation de l'apprentissage(supervisé) par la méthode de descente de gradient

Cette méthode est efficace loin du minimum et permet uniquement de s'en approcher. Pour cette raison, la détermination du pas n'est pas fiable loin du minimum et il faut seulement vérifier que le pas n'est ni trop petit ni trop grand. Pour cela, on définit une fonction d'erreur ou fonction de coût :

$$E = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 \tag{1.10}$$

avec

ζ_i^μ : sortie désirée
O_i^μ : sortie donnée par le réseau

où

$$O_i^\mu = g\left(h_i^\mu\right) = g\left(\sum_j w_{ij}V_j^\mu\right) = g\left(\sum_j w_{ij}g\left(\sum_k w_{jk}\zeta_k^\mu\right)\right) \tag{1.11}$$

alors

$$E = \frac{1}{2}\sum_{i\mu}\left[\zeta_i^\mu - g\left(\sum_j w_{ij}g\left(\sum_k w_{jk}\zeta_k^\mu\right)\right)\right]^2 \tag{1.12}$$

La fonction de coût est une fonction continue dérivable de chaque poids, la méthode de descente de gradient consiste alors $\grave{a}$ minimiser l'erreur $E(\bar{w})$.

a) Connexion couche cachée-couche de sortie

La méthode de descente de gradient donne :

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} \tag{1.13}$$

où η : pas d'apprentissage
et

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_i^\mu}\frac{\partial O_i^\mu}{\partial w_{ij}} \tag{1.14}$$

où encore

$$E = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 \tag{1.15}$$

$$O_i^\mu = g\left(h_i^\mu\right) = g\left(\sum_j w_{ij}V_j^\mu\right) \tag{1.16}$$

On a

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial O_i^\mu}\left[\frac{1}{2}\sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right)^2\right]\frac{\partial}{\partial w_{ij}}\left[g\left(\sum_j w_{ij}V_j^\mu\right)\right] \tag{1.17}$$

$$= \frac{1}{2}(2)(-1)\sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right)V_j^\mu g\prime\left(\sum_j w_{ij}V_j^\mu\right) \tag{1.18}$$

$$= -\sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right)g\prime\left(h_i^\mu\right)V_j^\mu \tag{1.19}$$

alors

$$\Delta w_{ij} = \eta \sum_{i\mu} \left(\zeta_i^\mu - O_i^\mu\right) g\prime\left(h_i^\mu\right) V_j^\mu \tag{1.20}$$

En posant

$$\delta_i^\mu = \sum_i g\prime\left(h_i^\mu\right)\left(\zeta_i^\mu - O_i^\mu\right) \tag{1.21}$$

On obtient

$$\Delta w_{ij} = \eta \sum_\mu \delta_i^\mu V_j^\mu \tag{1.22}$$

b) Connexion couche d'entrée-couche cachée

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \tag{1.23}$$

où

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial O_i^\mu} \frac{\partial O_i^\mu}{\partial V_j^\mu} \frac{\partial V_j^\mu}{\partial w_{jk}} \tag{1.24}$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial O_i^\mu}\left[\frac{1}{2}\sum i\mu \left(\zeta_i^\mu - O_i^\mu\right)^2\right] \frac{\partial}{\partial V_j^\mu}\left[g\left(\sum_j w_{ij} V_i^\mu\right)\right] \frac{\partial}{\partial w_{jk}}\left[g\left(\sum_k w_{jk}\zeta_k^\mu\right)\right] \tag{1.25}$$

$$= -\sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right) w_{ij}\; g\prime\left(\sum_j w_{ij} V_j^\mu\right) \zeta_k^\mu\; g\prime\left(\sum_k w_{jk}\zeta_k^\mu\right) \tag{1.26}$$

$$= -\sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right)\; g\prime\left(h_i^\mu\right) w_{ij}\; g\prime\left(h_j^\mu\right) \zeta_k^\mu \tag{1.27}$$

alors

$$\Delta w_{jk} = \eta \sum_{i\mu}\left(\zeta_i^\mu - O_i^\mu\right)\; g\prime\left(h_i^\mu\right) w_{ij}\; g\prime\left(h_j^\mu\right) \zeta_k^\mu \tag{1.28}$$

on pose

$$\left(\zeta_i^\mu - O_i^\mu\right)\; g\prime\left(h_i^\mu\right) = \delta_i^\mu \tag{1.29}$$

on a

$$\Delta w_{jk} = \eta \sum_{i\mu} \delta_i^\mu w_{ij}\; g\prime\left(h_j^\mu\right) \zeta_k^\mu \tag{1.30}$$

En posant

$$\delta_j^\mu = \sum_i \delta_i^\mu w_{ij}\; g\prime\left(h_j^\mu\right) \tag{1.31}$$

On obtient finalement

$$\Delta w_{jk} = \eta \sum_\mu \delta_j^\mu \zeta_k^\mu \tag{1.32}$$

c) Généralisation : algorithme d'apprentissage par descente de gradient pour réseau multicouches

- On fait entrer un prototype à la fois
- Considérons un réseau à M couche
  V_i^m : sortie de la i^{eme} unité dans la m^{eme} couche
  où m : indice de couche et i : indice d'unité
  avec $m = 1, 2, ..., M$
  μ : indice de prototype

S'il y a p prototype $\mu = 1, 2, ..., p$
w_{ij}^m : poids de connexion reliant V_j^{m-1} *à* V_i^m

Le procédé de propagation est :

Etape 1 : Initialiser les poids à 2 petites valeurs aléatoires

Etape 2 : Choisir un prototype ζ_k^μ et l'appliquer à la couche d'entrée $m = 1$ c'est-à-dire $\zeta_k^\mu = V_k^1$

Etape 3 : Propager le signal vers l'avant *à* travers le réseau en utilisant

$$V_i^m = g\left(h_i^m\right) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right) \tag{1.33}$$

Pour chaque i et pour m jusqu'à ce que les sorties finales V_i^m ait été toutes calculées

Etape 4 : Calculer les δ pour la couche de sortie en comparant les sorties actuelles V_i^M avec les sorties désirées ζ_i^μ pour le prototype μ

$$\delta_i^M = g\prime\left(h_i^M\right)\left(\zeta_i^\mu - V_i^M\right) \tag{1.34}$$

Etape 5 : Calculer les δ pour les couches précédentes en propageant les erreurs vers l'arrière jusqu'à ce qu'un *delta* ait été calculé pour chaque unité suivant la formule :

$$\delta_i^{m-1} = g\prime\left(h_i^{m-1}\right)\sum_j w_{ji}^m \delta_j^m \tag{1.35}$$

pour m=$\mu, \mu - 1, ..., 3$

Etape 6 : Utiliser $\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1}$ pour mettre *à* jour toutes les connexion suivant la relation

$$w_{ij}^{new} = w_{ij}^{old} + |Deltaw_{ij}^m \tag{1.36}$$

Etape 7 : Retourner *à* l'étape 2 et répéter le processus pour les autres prototypes.

d) Problème de minima locaux

Étant donnée la solution initiale A, une descente de gradient donnera B comme solution finale, or B n'est qu'un minimum local de la surface, la solution optimale C est inaccessible par cette méthode.

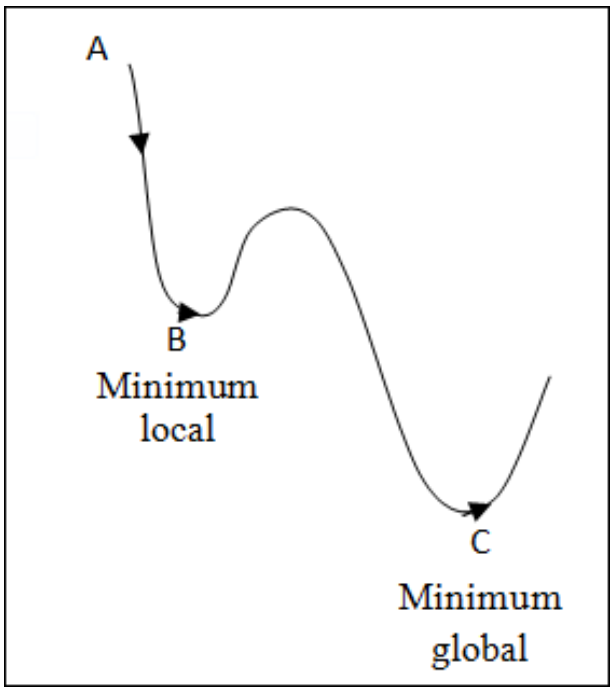

FIGURE 1.6 – minima locaux

Pour contourner ce problème, il est possible de combiner la recherche par descente de gradient avec une technique de recherche stochastique dite de $Monte - Carlo$

Quand une solution est obtenue, on explore la surface environnante par une série de sauts aléatoires. Si l'un de ceux-ci tombe sur un point plus bas que la solution courante, la recherche recommence à partir de ce nouveau point et ainsi de suite.

Chapitre 2

Système dynamique

2.1 Généralités

Depuis les travaux d'Isaac Newton (1687), l'idée est apparue que l'évolution temporelle d'un système physique quelconque est bien modélisée par une équation différentielle[20].

En mathématiques, la théorie du chaos étudie le comportement des systèmes dynamiques qui sont très sensibles aux conditions initiales, un phénomène généralement illustré par l'effet papillon[13]. Des différences infimes dans les conditions initiales (comme des erreurs d'arrondi dans les calculs numériques) entraînent des résultats totalement différents pour de tels systèmes, rendant en général toute prédiction impossible à long terme. Cela est valable même pour des systèmes déterministes, ce qui signifie que leur comportement futur est entièrement déterminé par leurs conditions initiales, sans intervention du hasard. En d'autres termes, la nature déterministe de ces systèmes ne les rend pas prévisibles. Ce comportement est connu sous le nom de chaos déterministe, ou tout simplement de chaos[13].

Le comportement chaotique est à la base de nombreux systèmes naturels, tels que la météo ou le climat. Ce comportement peut être étudié grâce à l'analyse par des modèles mathématiques chaotiques, ou par des techniques analytiques de récurrence et des applications de Poincaré. La théorie du chaos a des applications en météorologie[13].

L'évolution déterministe du système dynamique peut alors se modéliser de deux façons distinctes :

— une évolution continue dans le temps, représentée par une équation différentielle ordinaire. C'est a priori la plus naturelle physiquement, puisque le paramètre temps nous semble continu.

— une évolution discontinue dans le temps. Ce second cas est souvent le plus simple à décrire mathématiquement, même s'il peut sembler a priori moins réaliste physiquement. Cependant, l'étude théorique de ces modèles discrets est fondamentale, car elle permet de mettre en évidence des résultats importants, qui se généralisent souvent aux évolutions dynamiques continues.

2.2 Théorie du Chaos

La théorie du chaos s'attache principalement à la description de ces systèmes à petit nombre de degrés de liberté, souvent très simples à définir[23], mais dont la dynamique

nous apparaît comme très désordonnée.

2.2.1 Système chaotique

Un système chaotique est un système qui est étudié à partir d'une équation différentielle comme tout autre système mais dont la représentation dans un espace orthonormé cartésien donne une courbe complètement désordonnée. Cela est dû au fait que des petits écarts aux conditions initiales sont amplifiés de façon plus rapide au cours du temps.

Pour étudier un système chaotique il faut se placer dans l'espace des phases ou il apparaît clairement que le mouvement du corps étudié est alors chaotique.

2.2.2 Espace de phase

L'espace des phases est une structure correspondant à l'ensemble de tous les états possibles du système considéré. Ce peut être un espace vectoriel[21].

Pour un système possédant n degrés de liberté, par exemple, l'espace des phases Γ du système possède n dimensions, de telle sorte que l'état complet $x(t) \in \Gamma$ du système à l'instant t est en général un vecteur à n composantes.

2.2.3 Système dynamique différentiel conservatif

Pour un système possédant n degrés de libertés, l'espace des phases Γ du système possède $2n$ dimensions[21], de telle sorte que l'état complet $x(t) \in \Gamma$ du système à l'instant t est en général un vecteur à $2n$ composantes. On considère alors typiquement un système différentiel du premier ordre du type :

$$\frac{dx(t)}{dt} = f(x(t)) \tag{2.1}$$

où la fonction f définit le système dynamique étudié (c'est en général également un vecteur à n dimensions, c'est-à-dire un ensemble de n fonctions scalaires). Ce système physique, supposé conservatif, est déterministe si et seulement si la dynamique du système associe à chaque condition initiale x_0 un et un seul état final $x(t)$. Il faut pour cela qu'il existe une application bijective $\phi_t : \Gamma \longrightarrow \Gamma$ de l'espace des phases sur lui même telle que :

$$x(t) = \phi_t(x_0) \tag{2.2}$$

Lorsque le temps t varie, cette bijection engendre un flot sur Γ, c'est-à-dire un groupe continu à un paramètre ϕ_t. Cette modélisation mathématique correspond par exemple au flot hamiltonien de la mécanique classique.

2.2.4 Système dynamique à temps continu

Généralement, on peut représenter par une équation différentielle ce système dynamique en temps continu[20].
On va distinguer quelques types de différents systèmes.

— *Systèmes autonomes*

$$\dot{x} = f(x), x(t_0) = x_0 \tag{2.3}$$

— *Systèmes non-autonomes*

$$\dot{x} = f(t,x), x(t_0) = x_0 \tag{2.4}$$

— *Systèmes avec plusieurs variables d'états (autonomes ou non-autonomes)*

$$\begin{array}{l} \dot{x}_1 = f_1(t,x_1,x_2,...,x_m),\ x_1(t_0) = x_1^0 \\ \dot{x}_2 = f_2(t,x_1,x_2,...,x_m),\ x_2(t_0) = x_2^0 \\ . \\ . \\ . \\ \dot{x}_m = f_m(t,x_1,x_2,...,x_m),\ x_m(t_0) = x_m^0 \end{array} \tag{2.5}$$

En utilisant, la notation vectorielle pour ces systèmes :

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_m \end{pmatrix}$$

$$\vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_m \end{pmatrix}$$

alors, le système précédent s'écrit :

$$\frac{d}{dt}\vec{x} = \vec{f}(t,\vec{x}),\ \vec{x}(t_0) = \vec{x}_0 \tag{2.6}$$

— *Systèmes d'ordre $r \geqslant 2$ avec plusieurs variables d'état (autonomes ou non-autonomes)*

$$\frac{d^r}{dt^r}\vec{x} = \vec{f}\left(t,\vec{x},\frac{d}{dt}\vec{x},...,\frac{d^{r-1}}{dt^{r-1}}\vec{x}\right)\vec{x}(t_0) = \vec{x}_0,\ \frac{d}{dt}\vec{x}(t_0) = \vec{x}_1,...,\frac{d^{r-1}}{dt^{r-1}}\vec{x}(t_0) = \vec{x}_{r-1} \tag{2.7}$$

2.2.5 Systèmes dynamiques continus

C'est un système dont l'évolution des variables est de manière continue.

Nous allons montrer dans ce paragraphe, que les solutions des systèmes différentiels sont des systèmes dynamiques.

a) Résultats généraux sur les systèmes différentiels

Ce paragraphe a pour but de démontrer le lien entre les systèmes différentiels et les systèmes dynamiques continus.

Definition *(Problème de Cauchy)* Soient d un entier naturel positif, U un ouvert de $\mathbb{R}\times\mathbb{R}^d$ et f une application de U dans $\mathbb{R}^d$ au moins continu.
Soit l'application différentielle : $x\prime = f(t,x)$ pour tout couple appartenant à U.
Les conditions initiales sont fixées ; $t_0 \in \mathbb{R}$ et $x(t_0) = x_0 \in \mathbb{R}^d$.

Un problème de Cauchy consiste à déterminer un couple (I, x) où I est un intervalle de $\mathbb{R}$ tel que $t_0 \in I$ et x une fonction de I dans R^d vérifiant :

$$t \in I, x\prime(t) = f(t, x(t)) \; et \; x(t_0) = x_0$$

Théorème *(Cauchy-Lipschitz).* Soit le problème de Cauchy :

$$\begin{cases} \dot{x}(t) = f(t, x(t)) \; \forall t \in I \\ t_0 \in \mathbb{R}, \; x(t_0) = y_0 \in \mathbb{R}^d \end{cases} \tag{2.8}$$

Si f est continue sur $I \times \mathbb{R}^d$ dans $\mathbb{R}^d$ et est localement *lipschitzienne* en y, alors le problème de Cauchy admet une solution unique sur I.

b) Solutions des systèmes différentiels (Système dynamique continu)

Soient d un entier naturel positif, U un ouvert de $\mathbb{R} \times \mathbb{R}^d$ et f une application de U dans $\mathbb{R}^d$ de classe C^∞.
Considérons l'équation différentielle :

$$\begin{cases} \dot{x}(t) = f(t, x) \; \forall t \in U \\ t_0 \in \mathbb{R}, \; x(t_0) = y_0 \in \mathbb{R}^d \end{cases} \tag{2.9}$$

D'après le théorème de *Cauchy-Lipschitz* et puisque la fonction f est $C^\infty((\mathbb{R}\times\mathbb{R}^d), \mathbb{R}^d)$, il y a une existence et unicité de la solution sur un intervalle maximal, que nous supposons $\mathbb{R}$. La solution $x(t)$, ayant pour conditions initiales (t_0, x_0), est notée :

$$\begin{array}{l} \phi : \mathbb{R} \longrightarrow \mathbb{R}^d \\ t \longmapsto x(t) = \phi(t, t_0, x_0) t_0 \in \mathbb{R}, \; x(t_0) = x_0 \in \mathbb{R}^d \end{array} \tag{2.10}$$

Elle vérifie donc les deux relations : $\dot{\phi}(t, t_0, x_0) = f(\phi, t_0, x_0)$ et $\phi(t, t_0, x_0) = x_0$.

De ce fait, la solution ϕ est la courbe intégrale du système passant par le point (t_0, x_0). L'ensemble de toutes les courbes intégrales du système constitue un groupe à un paramètre $t \in \mathbb{R}$.

Ainsi, l'ensemble des solutions d'un système différentiel constitue un système dynamique. En d'autre termes, la fonction f, appelée champ de vecteurs, définit d'une part le système différentiel mais détermine également un système dynamique continu.

2.3 Système de Lorenz

Bien que le caractère vraisemblablement chaotique de la météorologie fût pressenti par Henri Poincaré, le météorologue Edward Lorenz est néanmoins considéré comme étant le premier à le mettre en évidence, en 1963[13,19].

Mathématiquement, le couplage de l'atmosphère avec l'océan est décrit par le système d'équations aux dérivées partielles couplées de Navier-Stokes de la mécanique des fluides. Ce système d'équations était beaucoup trop compliqué à résoudre numériquement pour les premiers ordinateurs existant au temps de Lorenz. Celui-ci eut donc l'idée de chercher un modèle très simplifié de ces équations pour étudier une situation physique particulière : le phénomène de convection de Rayleigh-Bénard. Il aboutit alors à un système dynamique

différentiel possédant seulement trois degrés de liberté, beaucoup plus simple à intégrer numériquement que les équations de départ.
Ce système différentiel s'écrit :

$$\begin{cases} \frac{dx(t)}{dt} = P_r\,(y(t) - x(t)) \\ \frac{dy(t)}{dt} = rx(t) - x(t)z(t) - y(t) \\ \frac{dz(t)}{dt} = x(t)y(t) - bz(t) \end{cases} \tag{2.11}$$

avec

- P_r est le nombre de Prandtl (toujours égal $\grave{a}$ 10).
- b : paramètre réel est égal $\grave{a}$ $\frac{8}{3}$.
- r le nombre de Rayleigh réduit sur un Rayleigh critique.
- $x(t)$ est proportionnel à l'intensité du mouvement de convection, $y(t)$ est proportionnel à la différence de température entre les courants ascendants et descendants, et $z(t)$ est proportionnel à l'écart du profil de température vertical par rapport à un profil linéaire

2.3.1 Point fixe

Tout point $\vec{x}$ de l'espace de phase, $F(\vec{x} = 0)$. Alors, les points fixes du système de Lorenz sont les solutions (x, y, z) constantes du système différentiel.

2.3.2 Étude de quelques modèles de Lorenz

Le but de cette étude est de connaître les trajectoires des évolutions temporelles de *x, y, z*, en faisant alors varier r et chaque valeur de r est associée aux conditions initiales (x_0, y_0, z_0)

Nous prenons $(x_0, y_0, z_0) = (1, 1, 1)$
Pour $r = 0.5$, voici la courbe correspondant aux évolutions temporelles de *(x, y, z)*

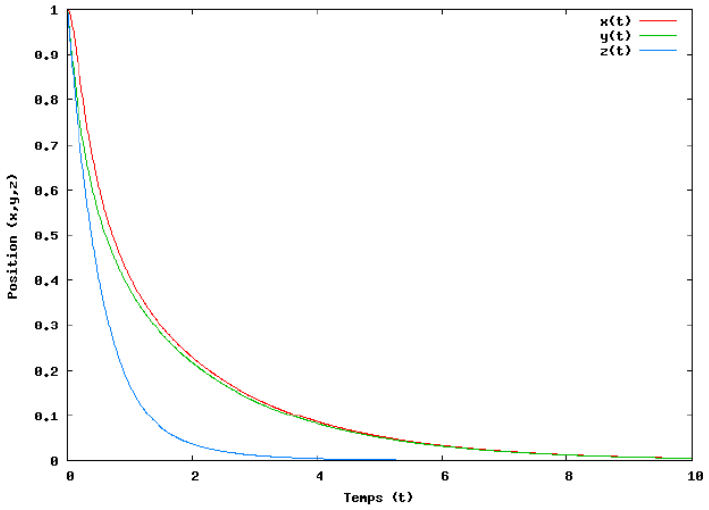

FIGURE 2.1 – Évolution temporelles de x, y, z avec $r = 0.5$

Pour cette valeur de r, les courbes des trois variables *(x,y,z)* convergent vers *(0,0,0)*.
On peut constater aussi que $y(t)$ et $x(t)$ ont le même comportement car ces courbes sont

très proches.

Donc, il reste à étudier les comportements de $x(t)$ et $z(t)$ pour les autres valeurs de r, avec $0 < r < 1$ et on verra ce qui ce passe.

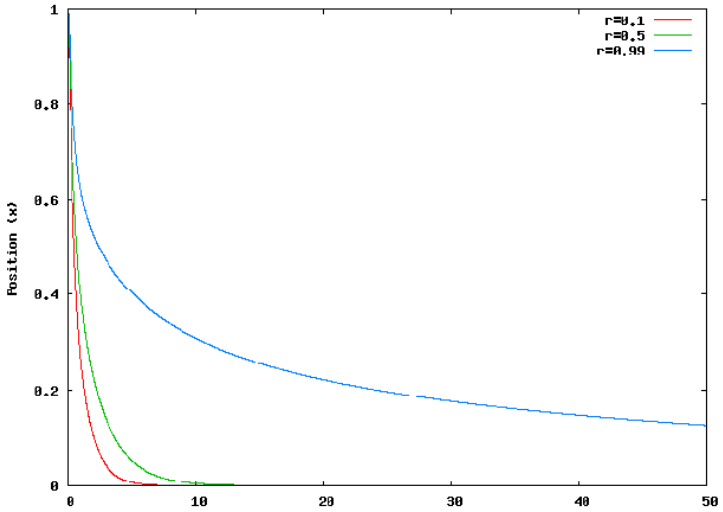

FIGURE 2.2 – Évolution temporelle de $x(t)$ pour différentes valeurs de r

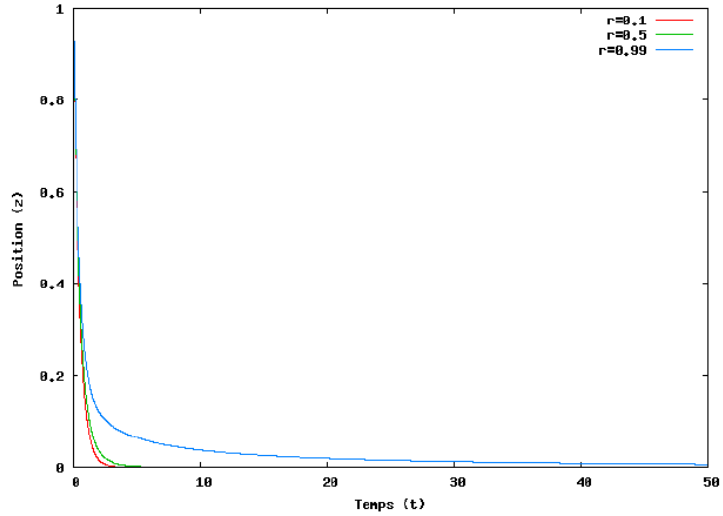

FIGURE 2.3 – Évolution temporelle de $z(t)$ pour différentes valeurs de r

Les deux graphes ci-dessus montrent que, pour les trois valeurs de r, les deux variables tendent toujours vers 0 des que t augmente. On note aussi que plus r est petit, plus $x(t)$ et $z(t)$ tendent rapidement vers 0.

Maintenant, prenons $r = 0.5$ avec $P_0 = (0, 0, 0)$; $P_1 = (-1, 2, 1/2)$; $P_2 = (2, 2, -3)$ où P_0, P_1, P_2 sont des conditions initiales.

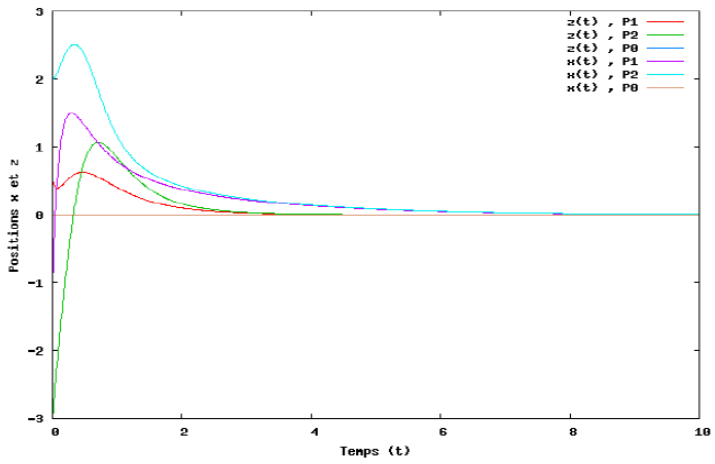

FIGURE 2.4 – Différentes conditions initiales pour r=0.5

Le point $P_0 = (0, 0, 0)$ est un point fixe, il est invariant dans le temps. De plus, quelles que soient leurs conditions initiales, les points convergent vers ce point fixe. C'est donc un ***point fixe stable*** et il est unique.

De plus, les trajectoires font penser à un nœud. On peut aussi visualiser les trajectoires en trois dimensions obtenues avec des conditions initiales différentes.

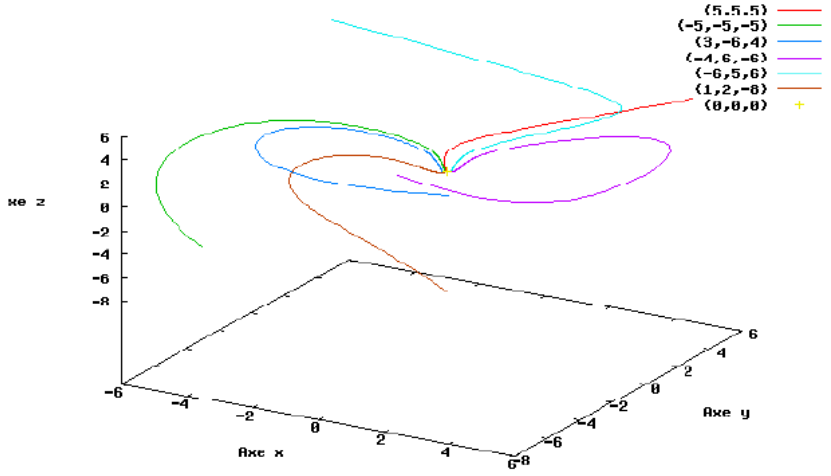

FIGURE 2.5 – Trajectoires avec r=0.5

On peut étudier aussi les trajectoires des trois variables pour $r > 1$ avec plusieurs conditions initiales.

En remarquant que pour $1 < r < 24$ le modèle de Lorenz admet trois points fixes, et les trajectoires convergent vers les points fixes stables c'est-à- dire vers les points fixes différents de $(0, 0, 0)$. Alors, on le voit très bien en regardant les trajectoires en trois dimensions.

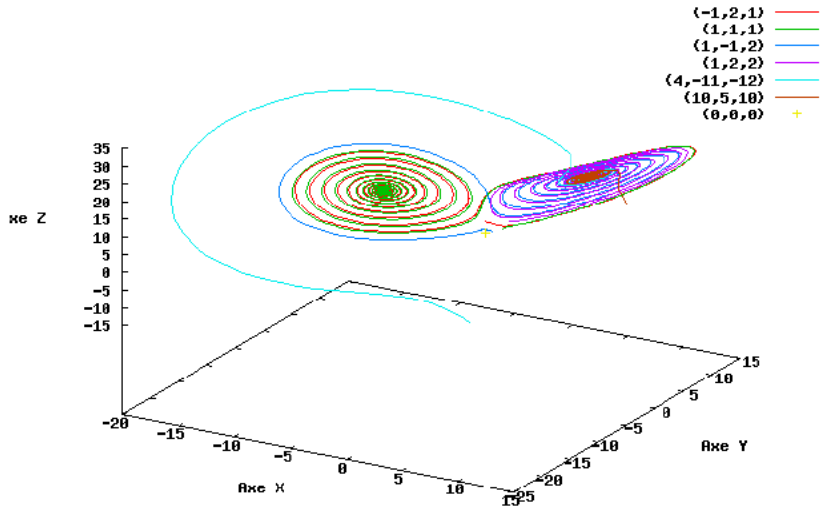

FIGURE 2.6 – Trajectoires avec $r = 15$

Pour $r > 25$, les deux points fixes stables précédemment deviennent instables, alors les trajectoires quittent les deux points.

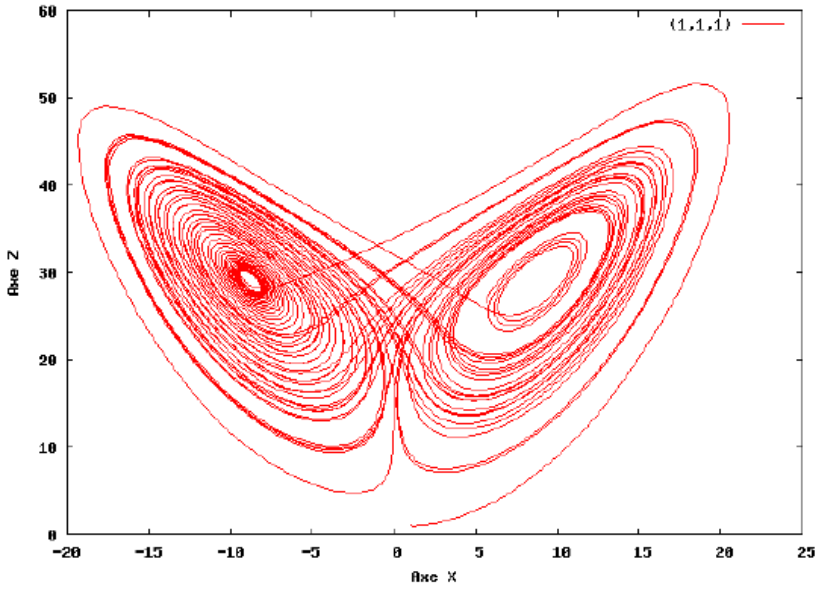

FIGURE 2.7 – Projection sur (x, z) pour $r = 30$

2.3.3 Simulation

Lorenz a choisi étudier son système avec les paramètres suivants.

$$\begin{aligned} P_r &= 10 \\ b &= 8/3 \\ r &= 28 \end{aligned}$$

c'est a dire :

$$\begin{cases} \frac{dx}{dt} = 10\,(y - x) \\ \frac{dy}{dt} = 28x - xz - y \\ \frac{dz}{dt} = xy - \frac{8}{3}z \end{cases}$$

Notre but est d'abord la résolution du système de façon numérique à partir des conditions initiales données et de visualiser la trajectoire à trois dimension de ce système par la *méthode de Runge-Kutta d'ordre 4*.

2.3.4 Méthode de Runge-Kutta d'ordre 4

a) Méthode

Les méthodes utilisées pour résoudre des systèmes d'équations différentielles du premier ordre à valeur initiale sont simplement des généralisations de la méthode de Runge-Kutta d'ordre 4 relative à une équation différentielle du premier ordre à valeur initiale. Pour cela, on divise l'intervalle $[a, b]$ en N sous intervalles identiques avec les nœuds

$t_j = a + jh$ pour $j = 0, 1, ..., N$
avec $h = \frac{b-a}{N}$: pas de la subdivision.
On approche $u_i(t_j)$ par w_{ij}
pour $j = 0, 1, ..., N$ et $i = 1, 2, ..., m$
$\Rightarrow$ Pour les conditions initiales on a :

$$\begin{aligned} w_{1,0} &= \alpha_1 \\ w_{2,0} &= \alpha_2 \\ &\vdots \\ w_{m,0} &= \alpha_m \end{aligned}$$

Si on suppose que les valeurs $w_{1,j}, w_{2,j}...., w_{m,j}$ ont été calculées, on obtient $w_{1,j+1}, w_{2,j+1}...., w_{m,j+1}$ en calculant d'abord :
$k_{1,i} = hf_i(t_j, w_{1,j}, w_{2,j}, ..., w_{m,j})$, pour $i = 1, 2, ..., m$
$k_{2,i} = hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{1,1}, w_{2,j} + \frac{1}{2}k_{1,2}, ..., w_{m,j} + \frac{1}{2}k_{1,m}\right)$ pour $i = 1, 2, ..., m$
$k_{3,i} = hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{2,1}, w_{2,j} + \frac{1}{2}k_{2,2}, ..., w_{m,j} + \frac{1}{2}k_{2,m}\right)$ pour $i = 1, 2, ..., m$
$k_{4,i} = hf_i(t_j + h, w_{1,j} + k_{3,1}, w_{2,j} + k_{3,2}, ..., w_{m,j} + k_{3,m})$ pour $i = 1, 2, ..., m$
$w_{i,j+1} = w_{i,j} + \frac{1}{6}[k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}]$ pour $i = 1, 2, ..., m$

b) Algorithme de Runge-Kutta pour des systèmes d'équations différentielles

Il permet d'approcher la solution du système d'ordre m de problème $\grave{a}$ valeurs initiales
$u\prime_i = f_i(t, u_1, u_2, ..., u_m) \quad i = 1, 2, ..., m$
$u_i(a) = \alpha_i \qquad i = 1, 2, ..., m$
en $N + 1$ nœuds également espacés de $[a, b]$

Step1 *set* $h = (b - a)/N \qquad t = a$

Step2 *For* $i = 1, 2, ...m$ *set* $w_i = \alpha_i$

Step3 *output* $(t,\ w_1,\ w_2,\ .....,\ w_m)$

Step4 *For* $j = 1,\ 2,\ .....,\ N$ *do steps* $5 - 11$

Step5 *For* $i = 1,\ 2,\ .....,\ m$ *set* $k_{1,i} = hf_i(t,\ w_1,\ w_2,\ ....., \ w_m)$

Step6 *Fori* $= 1,\ 2,\ .....,\ m$ *set*$k_{2,i} = hf_i\left(t + \frac{h}{2}, w_1 + \frac{1}{2}k_{1,1}, w_2 + \frac{1}{2}k_{1,2}, ..., w_m + \frac{1}{2}k_{1,m}\right)$

Step7 $For i = 1,\ 2,\ ....,\ m \quad set k_{3,i} = hf_i\left(t + \frac{h}{2}, w_1 + \frac{1}{2}k_{2,1}, w_2 + \frac{1}{2}k_{2,2}, ..., w_m + \frac{1}{2}k_{2,m}\right)$

Step8 $For\ i = 1,\ 2,\ ....,\ m \quad set\ k_{4,i} = hf_i\left(t + h, w_1 + k_{3,1}, w_2 + k_{3,2}, ..., w_m + k_{3,m}\right)$

Step9 $For\ i = 1,\ 2,\ ....,\ m \quad set\ w_i = w_i + \left(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}\right)/6$

Step10 $set\ t = a + jh$

Step11 $output\ \left(t, w_1, w_2, ..., w_m\right)$

Step12 $STOP$

Chapitre 3

Résultats et données

3.1 Langage utilisé

Nous avons choisi dans notre travail le langage de programmation ***c sharp (c#)***.

C# est un langage de programmation orienté objet commercialisé par Microsoft depuis 2002, et destiné à développer sur la plateforme Microsoft.NET. Nous avons utilisé ce langage, parce qu'il a beaucoup d' avantages par rapport aux autres langages de programmation.

Dans le domaine de la technologie, il est utilisé pour développer des application web, des services web, des applications de bureau et des commandes. Pour cela, nous devons créer tous les codes de programmation en c sharp. En c# une application classes comporte une méthode "Main" et possibilité de l'héritage.

Les développements avec des briques logiciels prêtes emploi sont accélérés, le déploiement devient facile, les conflits de versions lors de l'exécution du code pour minimiser et enfin c# fournit un environnement d'exécution de code sécurisé et performant.

On peut dire que, c# est le moteur qui exécute, contrôle et sécurise toutes les applications.

3.2 Série temporelle

Les séries temporelles constituent une branche de l'économétrie dont l'objet est l'étude des variables au cours du temps. Parmi ses principaux objectifs figurent la détermination de tendances au sein de ces séries ainsi que la stabilité des valeurs (et de leur variation) au cours du temps. On distingue notamment les modèles linéaires (principalement AR et MA, pour Auto-Regressive et Moving Average) les modèles conditionnels (notamment ARCH, pour Auto-Regressive Conditional Heteroskedasticity). L'analyse de ces séries touche énormément des domaines de la vie professionnelle, et plus précisément celui de l'informatique décisionnelle. En informatique, il s'agit d'une structure fondée sur les bases de données, fournissant ainsi le volume nécessaire d'information permettant de dresser une chronique historique des événements passés. Dessus viendrait se greffer un protocole d'extraction des données, intégré suivant un modèle judicieusement adapté à l'analyse que l'on voudrait faire. Enfin, au sommet de cette pyramide, la réponse à la question posée

au départ, qui sera la prévision.

3.2.1 Définition

Contrairement à l'économétrie traditionnelle, le but de l'analyse des séries temporelles n'est pas de relier des variables entre elles, mais de s'intéresser à la « dynamique » d'une variable. Cette dernière est en effet essentielle pour deux raisons : les avancées de l'économétrie ont montré qu'on ne peut relier que des variables qui présentent des propriétés similaires, en particulier une même stabilité ou instabilité ; les propriétés mathématiques des modèles permettant d'estimer le lien entre deux variables dépendent de leur dynamique.

La suite d'observations $(x(t), t \in T)$ *d'une variable* x *à différentes dates* t *est appelée série temporelle. Habituellement,* T *est dénombrable, de sorte que* $t = 1, \ldots, T$.

3.2.2 Représentation

On représente en général les séries temporelles sur des graphiques de valeurs (ordonnées) en fonction du temps (abscisses). Lorsqu'une série est stable autour de sa moyenne, on parle de série stationnaire. Inversement, on trouve aussi des séries non stationnaires. Lorsqu'une série croît sur l'ensemble de l'échantillon et donc possède une moyenne qui n'est pas constante, on parle de tendance. Enfin lorsqu'on observe des phénomènes qui se reproduisent à des périodes régulières, on parle de phénomène saisonnier.

Dans notre cas, l'étude du mouvement d'un fluide soumis à des échanges thermiques dans le domaine de la météorologie du système chaotique est la suivante :

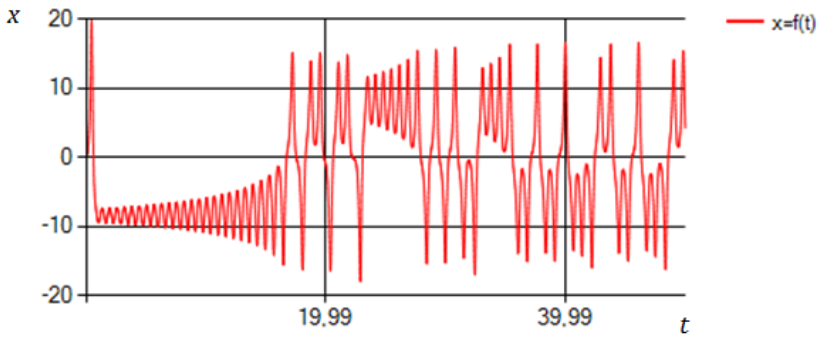

FIGURE 3.1 – Évolution de la série temporelle $x(t)$ déterminée par la méthode de Runge-Kutta

3.3 Intégration du système tridimensionnel de Lorenz

Les valeurs de la série sont obtenues, après l'intégration de ce système d'équations différentielles du premier ordre de Lorenz par la méthode de Runge-Kutta d'ordre 4, nous avons les estimations des 5000 valeurs pour chaque variable x et z avec un pas de 0.01.

Le tableau 3.1 donne une partie de ces valeurs, et dans la figure 3.2 nous avons le graphe de $x = f(z)$

i	x	z
1	0,09512136	0,00048007
2	0,18277226	0,00187674
3	0,26597656	0,00419490
4	0,34732875	0,00752004
5	0,42910495	0,01201374
6	0,51335403	0,01791098
7	0,60197340	0,02553329
8	0,69677274	0,03530171
9	0,79952869	0,04775895
10	0,91203249	0,06359859
11	1,03613263	0,083702860
12	1,17377362	0,109190998
13	1,32703219	0,141480479
14	1,49815150	0,182364186
15	1,68957384	0,234107244
16	1,90397191	0,299568062
17	2,14427827	0,382348909
18	2,41371209	0,486982245
19	2,71580161	0,619159640
20	3,05439964	0,786010473
100	-5,16136058	23,45859307
101	-5,23733618	23,30077656
200	-9,87885906	28,28347428
300	-4,93984634	28,07002722
400	-3,63556329	18,92749540
500	-7,55962763	17,36243109
600	13,01847570	27,63569682
700	-5,84515167	32,31473989
800	-0,80765655	28,97119355
900	1,963846016	18,48305596
1000	-1,92018499	13,35769123
1500	1,413547130	14,68311915
2000	2,037581223	19,82786198
2500	4,922149210	36,30380556
3000	0,075134447	12,60736028
3500	-10,85283280	31,39709798
4000	-10,01116733	17,23893490
4500	-10,14244165	35,89759727
4990	3,40394421	29,56016532
4994	4,34620388	28,23279245
4995	4,50975458	27,93976918
4996	4,65226961	27,65910805
4997	4,77704860	27,38988625
4998	4,88707799	27,13146356
4999	4,98504859	26,88343498
5000	5,07337437	26,64559042

TABLE 3.1 – Valeurs de x et z

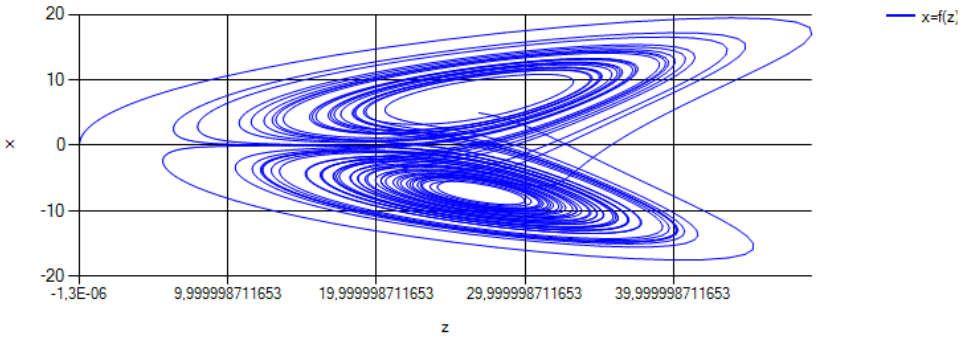

FIGURE 3.2 – Graphe $x = f(z)$

3.4 Architecture optimale du réseau

Pour un problème de prédiction, il est nécessaire d'avoir un réseau optimal car trop peu de paramètres perdraient les informations de la série et trop de paramètres consommeraient beaucoup de temps. Lorsqu'on utilise le Perceptron Multi-Couches, if faut choisir pour l'architecture optimale du réseau le nombre d'unités d'entrées, le nombre de couches et d'unités cachées, et le nombre d'unités de sortie.

3.4.1 Nombre de couche cachées

On utilise entre la couche d'entrée et la première couche cachée et entre les couches cachées elle-mêmes, la fonction sigmoïde et entre la dernière couche cachée et la couche de sortie la fonction identité.

En fait, le théorème de *Cybenko* montre qu'une seule couche cachée est suffisante pour approche toute fonction continue[24].

3.4.2 Nombre d'unités d'entrées

Ce nombre dépend du problème à traiter. Il est donné par l'algorithme de *Takens* qui fait appel au système dynamique en physique.

Algorithme de Takens

1– Soit une suite de données (série temporelle) $u(1), u(2), ..., u(i), ..., u(n), ...$

2– Construire une séquence de vecteurs à partir de cette suite (vecteur à n composantes).

$$\bar{x}\,(i) = \begin{pmatrix} u(i) \\ u(i+\tau) \\ u(i+2\tau) \\ . \\ . \\ . \\ u(i+(n-1)\tau) \end{pmatrix}$$

τ : paramètre de délais
on prend n assez grand

3– Définir la matrice de covariance à partir de
$\theta =< \bar{x}(i) \quad \bar{x}(i)^t >$
avec θ : matrice carrée (n, n) et $\bar{x}(i)^t$ transposée de $\bar{x}(i)$

4– Calculer les vecteurs propres de θ ainsi que ses valeurs propres λ qui sont rangées par ordre décroissant.

5– L'erreur d'approximation moyenne est $\epsilon_l = \sqrt{\lambda_{l+1}}$

6– Tracer ϵ_l en fonction de l.

7– La première valeur de l correspondant au premier plateau de la courbe donne la dimension de plongement de l'espace de phase reconstruit et qui est aussi égal au nombre d'unités d'entrée du réseau de neurones artificiel.

3.4.3 Nombre d'unités de sortie

Pour un problème de prédiction, on a besoin d'une seule unité de sortie.

3.4.4 Nombre d'unités cachées

De façon pratique, le nombre d'unités d'entrée et le nombre d'unités de sortie ayant été déterminés on fait varier la structure du réseau en donnant différentes valeurs au nombre d'unités cachées. Le nombre d'unités cachées du réseau optimal que l'on adoptera sera celui qui donnera l'erreur d'apprentissage la plus faible.

Dans notre travail, l'algorithme de Takens donne deux unités d'entrées et l'erreur d'apprentissage la plus faible correspond à deux unités cachées. Alors l'architecture optimale du réseau est de, 2 unités d'entrées, 2 unités cachées et 1 unité de sortie.

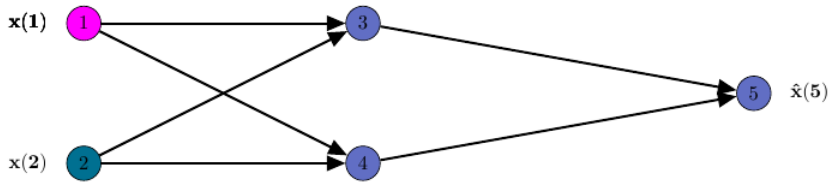

FIGURE 3.3 – Architecture optimale du réseau

3.5 Apprentissage du réseau

Nous avons un réseau optimal $(2, 2, 1)$. On introduit 10 prototypes pour chaque époque. On a pour la première époque :

$$\begin{pmatrix} 1^e prototype & x(1) & , & x(2) & , & x(3) , \hat{x}(3) \\ 2^{\text{è}} prototype & x(2) & , & x(3) & , & x(4) , \hat{x}(4) \\ 3^{\text{è}} prototype & x(3) & , & x(4) & , & x(5) , \hat{x}(5) \\ . & & & & & \\ . & & & & & \\ . & & & & & \\ 10^{\text{è}} prototype & x(10) & , & x(11) & , & x(12) , \hat{x}(12) \end{pmatrix}$$

avec $x(3)...x(12)$: valeurs attendues et $\hat{x}(3)...\hat{x}(12)$: valeurs données par le réseau après apprentissage.

Après une époque, on calcule l'erreur quadratique normalisée

$$NMSE = \frac{1}{N\sigma^2} \sum_{k=1}^{N} (x(k) - \hat{x}(k))^2 \tag{3.1}$$

avec N : nombre de prototypes

σ^2 : variance de la série $\sigma = E\left[X - E(X)\right]^2$

$x(k)$: valeur attendue

$\hat{x}(k)$: valeur donnée par le réseau

Après la *première époque* $\longrightarrow NMSE_1$
On reprend les 10 prototypes $\Rightarrow$ *deuxième époque* $\longrightarrow NMSE_2$
L'apprentissage se fait jusqu'à 10 époques.

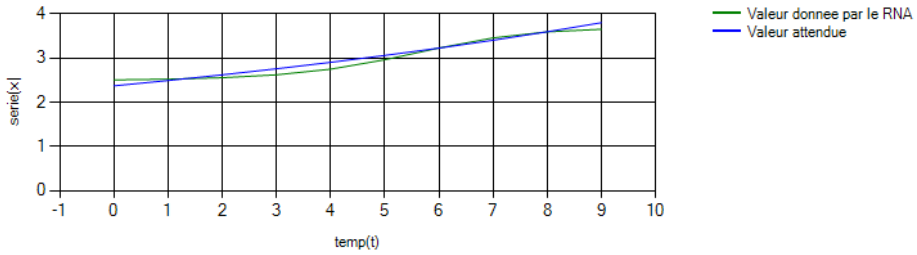

FIGURE 3.4 – Courbes qui représentent les valeurs attendues et les valeurs données par le réseau

Ces courbes montrent les sorties attendues et les sorties données par le réseau après l'apprentissage du réseau. Ici, les erreurs d'apprentissage ou les NMSE sont très faibles, alors les valeurs données par le réseau sont satisfaisantes. Le réseau est capable de connaitre les sorties désirées. Les NMSE en fonction de l'époque sont données par la figure suivante :

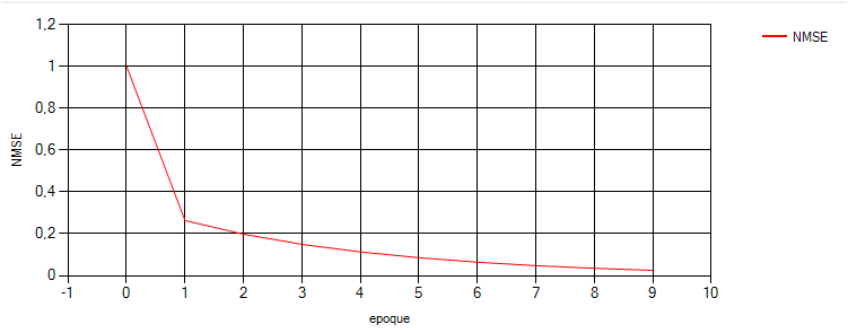

FIGURE 3.5 – Erreur quadratique normalisée

3.6 La prédiction

La prédiction d'une série temporelle est composée de 2 phases :
Premièrement : *l'apprentissage du réseau* sur un certain nombre de prototypes.
Deuxièmement : *la prédiction proprement dite* sur les prototypes qui n'ont pas utilisés lors de la phase d'apprentissage.

Soit une série temporelle : $x(1), x(2), ..., x(i), ..., x(n), ...$

3.6.1 Prédiction à un pas en avant

Ce type de prédiction consiste à faire la prédiction d'une seule valeur future de la série. Pour généraliser, choisissons 10 premiers prototypes à n nombre d'unités d'entrées, on obtient :

$$\begin{pmatrix} x(1),\ x(2),.....,x(n-1)\ , & x(n) & , & x(n+1)\ , & \hat{x}(n+1) \\ x(2),\ x(3),.....,x(n) & , x(n+1) & , & x(n+2)\ , & \hat{x}(n+2) \\ . & & & & \\ . & & & & \\ . & & & & \\ x(10),\ x(11),.....,x(n+8)\ , & x(n+9) & , & x(n+10)\ , & \hat{x}(n+10) \end{pmatrix}$$

avec $x(n+1), x(n+2), ..., x(n+10)$: valeurs attendues et $\hat{x}(n+1), \hat{x}(n+2), ..., \hat{x}(n+10)$: valeurs prédites donnée par le réseau.

∗ **Courbes qui représentent ces deux valeurs après calculs des valeurs prédites**

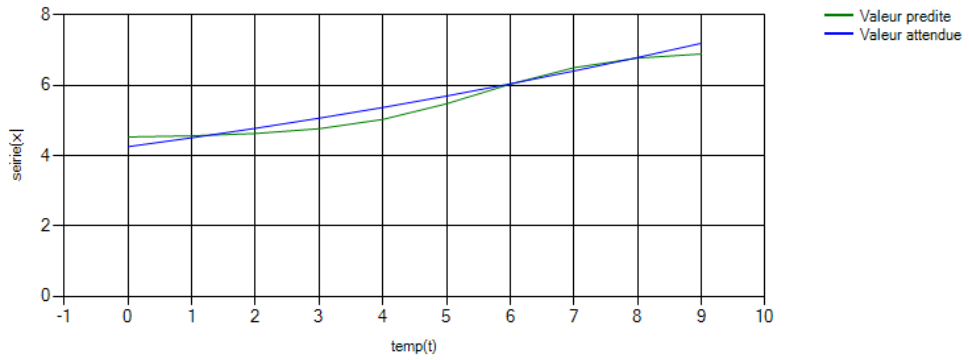

FIGURE 3.6 – Résultat pour la prédiction à un pas en avant

Les deux courbes sont presque confondues c'est à dire que, les valeurs attendues et les valeurs prédites coïncident.

3.6.2 Prédiction à plusieurs pas en avant

Le réseau est itéré en bouclage fermé autrement dit la sortie obtenue par le réseau est systématiquement rétro propagée en entrée à l'itération suivante.
Pour généraliser ce type de prédiction, prenons n nombre d'unités d'entrées et p prototypes jusqu'à $k^{ème}$ itération. Nous avons :

$$\begin{pmatrix} 1^{è}\ itération & x(p) & , & x(p+1) & , ... & , & x(p+n-1)\ , & \hat{x}(p+n) \\ 2^{è}\ itération & \hat{x}(p+n) & , & x(p) & , & x(p+1) & , ... & , x(p+n-2)\ , \hat{x}(p+n+1) \\ 3^{è}\ itération & \hat{x}(p+n+1), & \hat{x}(p+n) & , & x(p) & , ... & , & x(p+n-3)\ , \hat{x}(p+n+2) \\ . & & & & & & & \\ . & & & & & & & \\ . & & & & & & & \\ k^{è}\ itération & \hat{x}(p+n+k-2), & ... & , ... & ... & , & ... & , \hat{x}(p+n+k-1) \end{pmatrix}$$

Après plusieurs itérations, nous trouvons les résultats donnés par les courbes suivantes $(3.7, 3.8, 3.9)$.

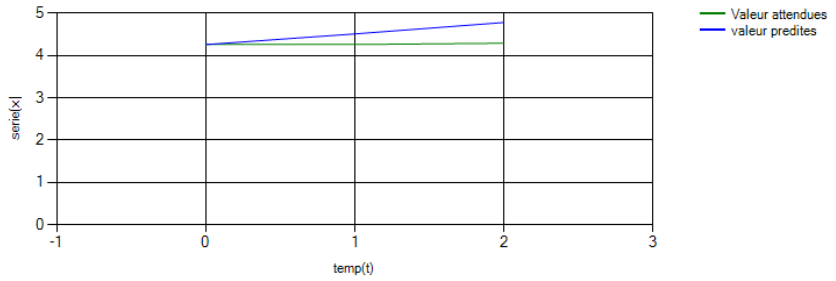

FIGURE 3.7 – Prédiction 3 pas en avant

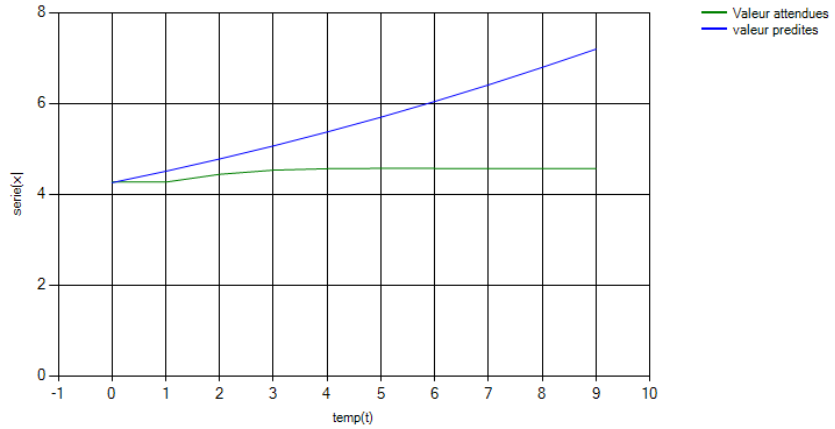

FIGURE 3.8 – Prédiction 10 pas en avant

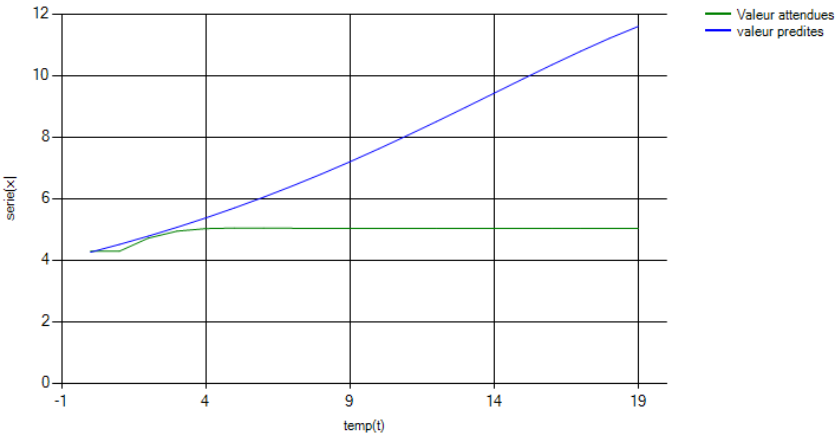

FIGURE 3.9 – Prédiction 20 pas en avant

Plus on augmente les itérations et, plus les écarts entre les valeurs prédites et valeurs attendues sont élevées. Ceci démontre le caractère chaotique du système de Lorenz.

Conclusion

Les Réseaux de Neurones Artificiels sont considérés comme des approches très intéressantes dans le domaine de l'intelligence artificielle. Ils sont connus par leur puissance d'apprentissage et de généralisation. Les réseaux de neurones artificiels sont capables de faire des prédictions de séries temporelles.

En premier lieu, l'objectif pédagogique de notre travail visé par ce survol du monde biologique est la mise en exergue d'une organisation structurelle des neurones. Chaque structure est dotée d'une fonction particulière et les structures adaptent leur comportement par des mécanismes d'apprentissage. L'apprentissage implique des modifications physiques des connexions entre neurones. Ensuite, il s'agit de développer un modèle de prédiction basé sur les réseaux de neurones artificiels à propagation de l'information vers l'avant ou les perceptrons multicouches. Nous avons cité aussi quelques théories des RNA très essentielles.

Puis nous avons étudié le système de Lorenz qui est un système dynamique. Ce système dynamique est dit "chaotique" car son espace des phases présente simultanément le phénomène de sensibilité aux conditions initiales et une forte récurrence. Pour cela, le comportement futur est entièrement déterminé par leur conditions initiales, sans intervention du hasard.

Finalement, nous avons fait la prédiction de la série temporelle générée par le système de Lorenz par le réseaux de neurones artificiels multicouches. L'Algorithme de Takens a permis de déterminer le nombre d'unités d'entrée du réseau.

La prédiction d'une série temporelle est composée de deux phases : l'apprentissage du réseau sur un certain nombre de prototypes et la prédiction proprement dite. Pour les prédictions à un pas en avant, les sorties données par le réseau et celles attendues coïncident, tandis que pour les prédictions à plusieurs pas en avant, celles-ci divergent lorsque le nombre d'itérations augmente. Ceci montre le caractère chaotique du système.

Bibliographie

[1] François Blayo et Michel Verleysen (1998), "Les réseaux de neurones artificiels", Presses Universitaires de France, *Que Sais-je No 3042*, 1ère édition.

[2] Léon Personnaz et Isabelle Rivals (2003), *Réseaux de neurones formels pour la modélisation, la commande et la classification*, CNRS Éditions.

[3] Gérard Dreyfus, Jean-Marc Martinez, Mannuel Samuelides, Mirta Gordon, Fouad Badran, Sylvie Thiria (2008), "Apprentissage statistique : réseaux de neurones, cartes topologiques, machines à vecteurs supports" Eyrolles.

[4] Eric Davalo, Patrick Naïm (1990), Des Réseaux de neurones, Eyrolles.

[5] Christopher M. Bishop (2006), Pattern Recognition And Machine Learning, Springer.

[6] Richard O. Duda, Peter E. Hart, David G. Stork (2001), Pattern classification, Wiley-interscience.

[7] Marc Parizeau (2004), Réseaux de neurones *(Le perceptron multicouche et son algorithme de retropropagation des erreurs)*, Université Laval, Laval, 272 p.

[8] Marc Parizeau (10 septembre 2004), Le perceptron multicouche et son algorithme de rétropropagation des erreurs .Département de génie électrique et de génie informatique Université Laval.

[9] MOON Francis C (1992), “chaotic and fractal dynamics”, New York Wiley.

[10] Daniel T (1997), Des données à la connaissance, Larose.

[11] Gourieroux et Monfort (1995), Séries temporelles et modèles dynamiques, Economica Paris.

[12] Régis Bourbonnais, Michel Terraza (2004) : Analyse des séries temporelles : applications à l'économie et à la gestion, Dunod.

[13] Edward N. Lorenz ; (1993), Un battement d'aile de papillon au Brésil peut-il déclencher une tornade au Texas ?, Alliage, 42-45. (1972), Traduction française du texte de la conférence, publié (en anglais) dans : The essence of chaos, Lecture Series, University of Washington Press. Ce livre contient une série de conférences de vulgarisation données à l'université de Washington (Seattle).

[14] Régis Bourbonnais, Michel Terraza (1998) : Analyse des séries temporelles en économie, PUF.

[15] George Box, Gwilym Jenkins (1970) : Time series analysis : Forecasting and control, Holden-Day.

[16] Georges Bresson, Alain Pirotte (1995) : Économétrie des séries temporelles : théorie et applications, PUF.

[17] Christian Gouriéroux et Alain Monfort (1995) : Séries temporelles et modèles dynamiques, Economica. Alain Pirotte (22 juin 2004), L'économétrie : Des origines aux développements récents, Paris, CNRS, 242 p.

[18] Ralf Vandenhouten (1998) : Non-stationary Time Series Analysis of Complex Systems and Applications in Physiology, Aachen, Shaker Verlag GmbH.

[19] Aurélien Alvarez (2013) : Destination Systèmes dynamiques avec Poincaré. Le Pommier.

[20] John H. Hubbard et Beverly H. West (1999), Équations différentielles et systèmes dynamiques, Cassini, (ISBN 284225015X).

[21] Diederich Hinrichsen (en) et Anthony J. Pritchard (2005), Mathematical Systems Theory. Modelling, State Space Analysis, Stability and Robustness, New York : Springer. (ISBN 978-3-540-44125-0)

[22] Boris Hasselblatt et Anatole Katok (1997), Introduction to the Modern Theory of Dynamical Systems : With a supplement by Anatole Katok and Leonardo Mendoza, Cambridge University Press, coll. « Encyclopedia of Mathematics and Its Applications » (no 54), (ISBN 0-521-57557-5)

[23] James Gleick (trad. Christian Jeanmougin) (1998-1999), La Théorie du Chaos [« Chaos : Making a New Science »], Flammarion, coll. « Champs », Paris, 431 p. (ISBN 2-08081-219-X)

[24] Approximation by superpositions of a sigmoidal function in Mathematics of control signals and systems 25 304-314

Annexes (Codes sources)

Annexes A
A.1 : Codes sources : Class Reseau

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace affiche
{
    class Reseau
    {
        private int entree;
        private int cachee;
        private int sortie;
        private double[,] matIn;
        private double[,] matOut;
        private double[,] poidsEntree;
        private double[,] poidsSortie;
        private int nProt;
        private double[,] Z3;
        private double[,] Z2;
        private double[,] A;
        private double[,] Z;
        private double cout1;
        private double cout2;
        private double pas;
        private int Epoque;
        double[,] dEdW2;
        double[,] dEdW1;
        private double Cout;
        private double scal;
        public Reseau(int dimIn, int dimHid, int dimOut)
        {
            this.entree = dimIn;
            this.cachee = dimHid;
            this.sortie = dimOut;
            this.cout1 = 0;
            this.cout2 = 0;
            this.Cout = 1;
            this.poidsEntree = new double[dimIn, dimHid];
            this.poidsSortie = new double[dimHid, dimOut];
            // this.Z3 = new double[this.nProt, dimOut];
            //this.Z2 = new double[this.nProt, this.cachee];
```

```
        //this.A = new double[this.nProt, this.cachee];
        //this.Z = new double[this.nProt, this.sortie];

        for (int i = 0; i < dimIn; i++)
            for (int j = 0; j < dimHid; j++)
            {
                Random r = new Random();
                this.poidsEntree[i, j] = r.NextDouble();
                // Console.WriteLine("W1["+i+","+j+"] = "+poidsEntree[i, j]);
            }
        for (int i = 0; i < dimHid; i++)
            for (int j = 0; j < dimOut; j++)
            {
                Random r = new Random();
                this.poidsSortie[i, j] = r.NextDouble();
                // Console.WriteLine("W2[" + i + "," + j + "] = " +
                    poidsSortie[i, j]);
            }

    }
    public double[,] propagation()
    {
        this.Z2 = new double[this.nProt, this.cachee];
        this.A = new double[this.nProt, this.cachee];
        this.Z = new double[this.nProt, this.sortie];
        this.Z3 = new double[this.nProt, this.sortie];
        for (int i = 0; i < this.nProt; i++)
        {
            for (int j = 0; j < this.cachee; j++)
            {
                double value = 0;
                for (int k = 0; k < this.entree; k++)
                {
                    value += this.matIn[i, k] * this.poidsEntree[k, j];
                }
                this.Z2[i, j] = value;
                this.A[i, j] = this.hypertan(value);
                // Console.WriteLine("A = " + A[i, j]);
            }
        }
        //z2 = a*W2
        for (int i = 0; i < this.nProt; i++)
        {
            for (int j = 0; j < this.sortie; j++)
            {
                double value = 0;
                for (int k = 0; k < this.cachee; k++)
                {
                    value += A[i, k] * this.poidsSortie[k, j];
                }
                this.Z3[i, j] = value;
                this.Z[i, j] = this.lineaire(value);
            }
```

```
    }
    return Z;
}
public void FonctionCoutPrim()
{
    // double[,] yH = this.propagation();
    double[,] delta1 = new double[this.nProt, this.sortie];
    this.dEdW2 = new double[this.cachee, this.sortie];
    this.dEdW1 = new double[this.entree, this.cachee];
    double[,] delta2 = new double[this.nProt, this.cachee];
    for (int i = 0; i < this.nProt; i++)
    {
        for (int j = 0; j < this.sortie; j++)
        {

            delta1[i, j] = -(this.matOut[i, j] - propagation()[i, j]) *
                this.lineaireDerivee(this.Z3[i, j]);
            //Console.WriteLine("delta1 = " + delta1[i, j]);
        }
    }
    //----------------------

    for (int i = 0; i < this.cachee; i++)
    {
        for (int j = 0; j < this.sortie; j++)
        {
            double value = 0;
            for (int k = 0; k < this.nProt; k++)
            {
                value += this.getPas() * this.transpose(this.A, this.
                    nProt, this.cachee)[i, k] * delta1[k, j];
                // Console.WriteLine("valeur = " + value);
            }
            dEdW2[i, j] = value;
            this.cout1 += dEdW2[i, j];
            // Console.WriteLine("dEdW2 = "+value);
        }
    }
    Console.WriteLine("cout1 = " + this.cout1);
    //---------------------
    double[,] XD = new double[this.nProt, this.cachee];
    double nb = 0;
    for (int i = 0; i < this.nProt; i++)
    {
        for (int j = 0; j < this.cachee; j++)
        {
            double value = 0;
            for (int k = 0; k < this.sortie; k++)
            {
                value += delta1[i, k] * this.transpose(this.poidsSortie,
                    this.cachee, this.sortie)[k, j];

            }
```

```
                XD[i, j] = value;
                nb += value;

            }
        }
        //-----------------------
        double[,] V = new double[this.nProt, this.cachee];
        for (int i = 0; i < this.nProt; i++)
        {
            for (int j = 0; j < this.cachee; j++)
            {
                //double value = 0;
                //for (int k = 0; k < this.nProt; k++)
                //{
                //value += -this.sigmoidDeriv(this.transpose(this.Z2, this.
                    cachee, this.sortie)[k,i]) * XD[k, j];
                delta2[i, j] = -this.deriveeHypertan(this.Z2[i, j]) * nb;
                //}
                //delta2[i, j] = value;
                // Console.WriteLine("delta2 = " + delta2[i, j]);
            }
        }
        //----------------------------
        for (int i = 0; i < this.entree; i++)
        {
            for (int j = 0; j < this.cachee; j++)
            {
                double value = 0;
                for (int k = 0; k < this.nProt; k++)
                {
                    value += this.getPas() * this.transpose(this.matIn, this.
                        nProt, this.entree)[i, k] * delta2[k, j];
                }
                dEdW1[i, j] = value;
                this.cout2 += dEdW1[i, j];
                // Console.WriteLine("dE/dW1 = " + dEdW1[i, j]);
            }
        }
        Console.WriteLine("Cout2 = " + this.cout2);
    }
    public void miseAjour()
    {
        //------poids de sortie
        for (int i = 0; i < this.cachee; i++)
        {
            for (int j = 0; j < this.sortie; j++)
            {
                this.poidsSortie[i, j] = this.poidsSortie[i, j] - this.
                    getScal() * this.dEdW2[i, j];
                // Console.WriteLine("PoidSortie = " + this.poidsSortie[i, j
                    ]);
            }
        }
```

```
    //- poids entree
    for (int i = 0; i < this.entree; i++)
    {
        for (int j = 0; j < this.cachee; j++)
        {
            this.poidsEntree[i, j] = this.poidsEntree[i, j] - this.
                getScal() * this.dEdW1[i, j];
            //Console.WriteLine("PoidEntree = " + this.poidsEntree[i, j])
                ;
        }
    }
}
public void apprendre()
{
    int epoque = 0;
    while (epoque <= this.getEpoque())
    {
        Console.WriteLine("Epoque = " + epoque);
        double cout = this.getCout();
        Console.WriteLine("Coutqw = " + cout);
        this.FonctionCoutPrim();
        this.miseAjour();
        this.Cout = this.coutCalc();
        if (this.Cout > cout) break;
        else cout = this.Cout;
        Console.WriteLine("Coutwwe = " + cout);

        epoque++;
    }
}
public double returnMax(double[,] input)
{
    double max = -1000;
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.entree; j++)
        {
            if (input[i, j] >= max)
                max = input[i, j];
        }
    return max;
}
public double returnMaxO(double[,] input)
{
    double max = -1000;
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.sortie; j++)
        {
            if (input[i, j] >= max)
                max = input[i, j];
        }
    return max;
}
```

```
public double returnMin(double[,] input)
{
    double max = 1000;
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.entree; j++)
        {
            if (input[i, j] <= max)
                max = input[i, j];
        }
    return max;
}
public double returnMin0(double[,] input)
{
    double max = 1000;
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.sortie; j++)
        {
            if (input[i, j] <= max)
                max = input[i, j];
        }
    return max;
}
public double[,] normIn(double[,] entre)
{
    double[,] mat = new double[this.nProt, this.entree];
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.entree; j++)
        {
            //mat[i, j] = entre[i, j] /Math.Abs(this.returnMax(entre));
            mat[i, j] = 2 * ((entre[i, j] - this.returnMin(entre)) / (
                this.returnMax(entre) - this.returnMin(entre))) - 1;
        }
    return mat;
}
public double[,] denormIn(double[,] entre, double max, double min)
{
    double[,] mat = new double[this.nProt, this.entree];
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.entree; j++)
        {
            // mat[i, j] = entre[i, j] *Math.Abs(max);
            mat[i, j] = (max - min) * ((entre[i, j] + 1) / 2) + min;
        }
    return mat;
}
public double[,] normOut(double[,] sortie)
{
    double[,] mat = new double[this.nProt, this.sortie];
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.sortie; j++)
        {
            //mat[i, j] = sortie[i, j] /Math.Abs( this.returnMax0(sortie)
                );
```

```
            mat[i, j] = 2 * (sortie[i, j] - this.returnMin0(sortie)) / (
                this.returnMax0(sortie) - this.returnMin0(sortie)) - 1;
        }
    return mat;
}
public double[,] dnormOut(double[,] sortie, double max, double min)
{
    double[,] mat = new double[this.nProt, this.sortie];
    for (int i = 0; i < this.nProt; i++)
        for (int j = 0; j < this.sortie; j++)
        {
            //mat[i, j] = sortie[i, j] * Math.Abs(max);
            mat[i, j] = (max - min) * ((sortie[i, j] + 1) / 2) + min;
        }
    return mat;
}
public double[,] transpose(double[,] mat, int dl, int dc)
{
    double[,] resultat = new double[dc, dl];
    for (int i = 0; i < dl; i++)
    {
        for (int j = 0; j < dc; j++)
        {
            resultat[j, i] = mat[i, j];
        }
    }
    return resultat;
}
public double coutCalc()
{
    double E = 0;
    for (int i = 0; i < this.nProt; i++)
    {
        for (int j = 0; j < this.sortie; j++)
            E += Math.Pow(this.matOut[i, j] - this.propagation()[i, j],
                2);
    }
    Console.WriteLine("Cout(EPOK) = " + E);
    return E / 2;

}
public double sigmoidDeriv(double x)
{
    return Math.Exp(-x) / (Math.Pow((1 + Math.Exp(-x)), 2));
}
public double sigmoid(double x)
{
    return 1 / (1 - Math.Exp(-x));
}
public double lineaire(double x)
{
    return x;
}
```

```
public double lineaireDerivee(double x)
{
    return 1;
}
public double hypertan(double x)
{
    return Math.Tanh(x);
}
public double deriveeHypertan(double x)
{
    return 1 / Math.Pow(Math.Cosh(x), 2);
}
public void setMatIn(double[,] mat)
{
    this.matIn = mat;
}
public void setMatOut(double[,] mat)
{
    this.matOut = mat;
}
public void setNProt(int n)
{
    this.nProt = n;
}
public double getCout1()
{
    return this.cout1;
}
public double getCout2()
{
    return this.cout2;
}
public double getCout()
{
    return this.Cout;
}
public double getPas()
{
    return this.pas;
}
public void setPas(double p)
{
    this.pas = p;
}
public int getEpoque()
{
    return this.Epoque;
}
public void setEpoque(int p)
{
    this.Epoque = p;
}
public double getScal()
```

```
        {
            return this.scal;
        }
        public void setScal(double p)
        {
            this.scal = p;
        }
        public int getEntree()
        {
            return this.entree;
        }
        public int getCachee()
        {
            return this.cachee;
        }
        public int getSortie()
        {
            return this.sortie;
        }
        public int getNProt()
        {
            return this.nProt;
        }
    }
}
```

A.2 : Codes sources : Class runge-kutta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace affiche
{
    class runge_kutta
    {
        private int a, b, m, N; // [a , b] ,N= nombre de subdivision , m =
            nombre d'equation
        int[] alfa;// avec alpha1[0]= x(0)
        double[] w;
        double[,] k;
        double h, t;
        double[] tab_w1;
        double[] tab_w2;
        double[] tab_w3;
        public runge_kutta(int io_a, int io_b, int io_nb_equa, int io_N)
        {
            a = io_a; b = io_b; m = io_nb_equa; N = io_N;
            alfa = new int[m + 1];
            w = new double[m + 1];
            k = new double[m + 2, m + 1];
            tab_w1 = new double[N + 1];
            tab_w2 = new double[N + 1];
```

```
    tab_w3 = new double[N + 1];
    h = (b - a) / Convert.ToDouble(N);
    t = a;
}
public void condition_init(int[] tab)// tab contient la liste des
    conditions initiales de chaque variable
{
    alfa = tab;
}
public void step1()
{
    h = (b - a) / Convert.ToDouble(N);
    t = a;
}
public void step2()
{

    for (int i = 1; i <= m; i++)
    {
        w[i] = alfa[i];
    }
}
public void step3()
{

    tab_w1[0] = w[1];
    tab_w2[0] = w[2];
    tab_w2[0] = w[3];

}
public void step4()
{
    for (int j = 1; j <= N; j++)
    {

        for (int i = 1; i <= m; i++)
        {
            k[1, i] = h * f(i, w[1], w[2], w[3]);
        }
        for (int i = 1; i <= m; i++)
        {
            k[2, i] = h * f(i, w[1] + k[1, 1] / 2, w[2] + k[1, 2] / 2, w
                [3] + k[1, 3] / 2);
        }
        for (int i = 1; i <= m; i++)
        {
            k[3, i] = h * f(i, w[1] + k[2, 1] / 2, w[2] + k[2, 2] / 2, w
                [3] + k[2, 3] / 2);
        }
        for (int i = 1; i <= m; i++)
        {
            k[4, i] = h * f(i, w[1] + k[3, 1], w[2] + k[3, 2], w[3] + k
                [3, 3]);
```

```
                }
                for (int i = 1; i <= m; i++)
                {
                    w[i] = w[i] + (k[1, i] + 2 * k[2, i] + 2 * k[3, i] + k[4, i])
                        / 6;
                }
                tab_w1[j] = w[1];
                tab_w2[j] = w[2];
                tab_w3[j] = w[3];

            }
        }
        // fonction qui gere les fonctions dans step4
        public double f(int i, double x, double y, double z)
        {

            double res = 0;// res : resultat
            switch (i)
            {
                case 1: res = -10 * x + 10 * y; break;
                case 2: res = -x * z + 28 * x - y; break;
                case 3: res = x * y - (8 / 3) * z; break;
            }
            return res;

        }

        // milay be
        public double[] get_tab_w1()
        {
            return tab_w1;
        }
        public double[] get_tab_w2()
        {
            return tab_w2;
        }
        public double[] get_tab_w3()
        {
            return tab_w3;
        }
    }
}
```

A.3 : Class Takens

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace affiche
{
    class Takens
    {
```

```
        double[] x_b;
        double[] x;
        int ndim;
        //------------------constructeur----------------------
        public Takens(int d, double[] vect)
        {
            ndim = d;
            x_b = new double[ndim];
            x = new double[ndim];
            for (int i = 0; i < d; i++)
                x_b[i] = vect[i];
        }

        //-----------------covariance------------------------
        public Matrix prod_vec(Matrix X, Matrix X_b)
        {
            Matrix mat = new Matrix(ndim, ndim);
            for (int i = 0; i < ndim; i++)
            {
                for (int j = 0; j < ndim; j++)
                    mat.setValeur(i, j, X.getValeur(0, i) * X_b.getValeur(j, 0));
            }

            return mat;
        }

        //------------------------ conversion vercteur en matrice 1d
            -------------------------
        public Matrix vect_col(double[] vect)
        {
            Matrix matri = new Matrix(ndim, 1);
            for (int k = 0; k < ndim; k++)
                matri.setValeur(k, 0, vect[k]);

            return matri;
        }

        public Matrix vect_lig(double[] vect)
        {
            Matrix matri = new Matrix(1, ndim);
            for (int k = 0; k < ndim; k++)
                matri.setValeur(0, k, vect[k]);

            return matri;
        }
    }
}
```

A.4 : Class Prediction

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace affiche
{
    class Prediction
    {
        private double[,] matriceEntree;
        private double[,] sortie;
        private double[,] matriceSortie;
        private double[,] msortieUnpas;
        private double[,] msortiePluspas;
        private int nbUnpas;
        private int nbPluspas;
        public Prediction()
        {
        }
        public double[,] predictionUnPas(int nb, int indicePrototype, Reseau
            reseau, Serie_temporelles serietemporelles)
        {
            matriceEntree = new double[nb, reseau.getEntree()];
            sortie = new double[nb, reseau.getSortie()];
            matriceSortie = new double[nb, reseau.getSortie()];
            msortieUnpas = new double[nb, reseau.getSortie()];
            for (int i = 0; i < nb; i++)
            {
                for (int j = 0; j < reseau.getEntree(); j++)
                {
                    matriceEntree[i, j] = serietemporelles.getX()[i + j + 1 +
                        indicePrototype];
                    matriceSortie[i, 0] = serietemporelles.getX()[reseau.
                        getEntree() + i + 1 + indicePrototype];
                    msortieUnpas[i, 0] = serietemporelles.getX()[reseau.getEntree
                        () + i + 1 + indicePrototype];
                }
            }
            double[,] InN = new double[nb, reseau.getEntree()];
            double[,] OutN = new double[nb, reseau.getSortie()];
            reseau.setNProt(nb);
            InN = reseau.normIn(matriceEntree);
            OutN = reseau.normOut(matriceSortie);
            this.matriceSortie = OutN;

            for (int i = 0; i < nb; i++)
            {
                for (int j = 0; j < reseau.getEntree(); j++)
                {
                    Console.WriteLine("MatriceEntree = " + InN[i, j]);
                }
                Console.WriteLine("MatriceSortie = " + OutN[i, 0]);
            }
            reseau.setMatIn(InN);
            reseau.setMatOut(OutN);
            sortie = reseau.propagation();
```

```
    return sortie;
}
public double[,] predictionPlusieurPas(int nbPas, int indicePrototype,
    Reseau reseau, Serie_temporelles serietemporelles)
{
    matriceEntree = new double[1, reseau.getEntree()];
    double[,] InN = new double[1, reseau.getEntree()];
    double[,] sortie_k = new double[nbPas, 1];
    matriceSortie = new double[nbPas, reseau.getSortie()];
    msortiePluspas = new double[nbPas, reseau.getSortie()];
    for (int k = 0; k < nbPas; k++)
    {
        if (k == 0)
        {
            for (int i = 0; i < 1; i++)
            {
                for (int j = 0; j < reseau.getEntree(); j++)
                {
                    matriceEntree[i, j] = serietemporelles.getX()[i + j +
                         1 + indicePrototype + k];
                    matriceSortie[k, 0] = serietemporelles.getX()[reseau.
                        getEntree() + i + 1 + indicePrototype + k];
                    msortiePluspas[k, 0] = serietemporelles.getX()[reseau
                        .getEntree() + i + 1 + indicePrototype + k];
                }
            }
            sortie = new double[1, reseau.getSortie()];
            reseau.setNProt(1);
            InN = reseau.normIn(matriceEntree);
            reseau.setMatIn(InN);
            sortie = reseau.propagation();
            sortie_k[k, 0] = sortie[0, 0];
            for (int i = 0; i < 1; i++)
            {
                for (int j = 0; j < reseau.getEntree(); j++)
                {
                    Console.WriteLine("MatriceEntree = " + InN[i, j]);
                }
            }
        }
        else
        {
            int pas_i = k;
            for (int i = 0; i < 1; i++)
            {
                for (int j = 0; j < reseau.getEntree(); j++)
                {
                    if (j >= pas_i)
                    {
                        matriceEntree[i, j] = InN[0, j - pas_i];
                    }
                    else
                    {
```

```
                    matriceEntree[i, j] = sortie_k[pas_i - 1, 0];
                }
                pas_i--;
            }
        }
        sortie = new double[1, reseau.getSortie()];
        reseau.setNProt(1);

        reseau.setMatIn(matriceEntree);
        sortie = reseau.propagation();
        sortie_k[k, 0] = sortie[0, 0];
        matriceSortie[k, 0] = serietemporelles.getX()[reseau.
            getEntree() + 1 + indicePrototype + k];
        msortiePluspas[k, 0] = serietemporelles.getX()[reseau.
            getEntree() + 1 + indicePrototype + k];
    }

}
double[,] OutN = new double[nbPas, reseau.getSortie()];
OutN = reseau.normOut(matriceSortie);

this.matriceSortie = OutN;
return sortie_k;
}
public double[,] getMatSortie()
{
    return this.matriceSortie;
}
public double[,] getSortie()
{
    return this.sortie;
}
public double[,] getMSortieUnpas()
{
    return this.msortieUnpas;
}
public double[,] getMSortiePluspas()
{
    return this.msortiePluspas;
}
public int getNbUnpas()
{
    return nbUnpas;
}
public int getNbPluspas()
{
    return nbPluspas;
}
public void setNbUnpas(int n)
{
    this.nbUnpas = n;
}
public void setNbPluspas(int n)
```

```
        {
            this.nbPluspas = n;
        }

    }
}
```

A.5 : Class Jacobi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace affiche
{
    class Jacobi
    {
        public Jacobi()
        {

        }
        //----------------------- tri decroissante
            ----------------------------------------

        private double[] dec_sort(double[] tab)
        {
            double tmp;
            for (int i = 0; i < tab.Length; i++)
                for (int j = 0; j < tab.Length; j++)
                    if (tab[i] < tab[j])
                    {
                        tmp = tab[i];
                        tab[i] = tab[j];
                        tab[j] = tmp;
                    }
            return tab;
        }
        //------------cosinus et sinus------------------
        private double cos(double a, double b)
        {
            return (Math.Sqrt(0.5 * (1 + (b / Math.Sqrt(a * a + b * b)))));
        }

        private double sin(double a, double b)
        {
            return (a / (2 * cos(a, b) * Math.Sqrt(a * a + b * b)));
        }

        //--------------diagonalization----------------------
        public Matrix diagonalization(Matrix m)
        {
            double c, d;
            // Matrix mat = new Matrix(m.getNbLignes(),m.getNbColonnes());
```

```
    int lmax = m.max_ligne(m);

    int cmax = m.max_colonne(m);

    Matrix P = new Matrix(m.getNbColonnes(), m.getNbColonnes()).
        MatriceUnit(m.getNbColonnes());
    double a = 0, b = 0;
    a = 2 * m.getValeur(lmax, cmax);
    b = m.getValeur(lmax, lmax) - m.getValeur(cmax, cmax);
    if (m.getValeur(lmax, lmax) != m.getValeur(cmax, cmax))
    {
        d = sin(a, b);
        c = cos(a, b);
    }
    else
    {
        c = Math.Sqrt(2) / 2;
        d = Math.Sqrt(2) / 2;
    }
    for (int i = 0; i < P.getNbLignes(); i++)
    {
        for (int j = 0; j < P.getNbColonnes(); j++)
        {
            if ((i == j) && (j != lmax) && (j != cmax))
                P.setValeur(i, j, 1);
            else if ((i == cmax) && (j == lmax))
                P.setValeur(i, j, -d);
            else if ((i == lmax) && (j == lmax) || (i == cmax) && (j ==
                cmax))
                P.setValeur(i, j, c);
            else if ((i == lmax) && (j == cmax))
                P.setValeur(i, j, d);
            else P.setValeur(i, j, 0);
        }
    }

    Matrix trans_P = new Matrix(P.transpose());

    Matrix M = new Matrix(m.getNbLignes(), m.getNbColonnes());
    M = M.multiplie(M.multiplie(trans_P, m), P);

    return M;
}
//------------------ valeur propre ---------------------------
public double[] val_prop(Matrix mat)
{
    double[] valprop = new double[mat.getNbLignes()];
    Matrix matrix = new Matrix(mat.getNbLignes(), mat.getNbColonnes());
    Jacobi jc = new Jacobi();
    matrix = jc.diagonalization(mat);
    for (int i = 0; i < matrix.getNbLignes(); i++)
    {
        for (int j = 0; j < matrix.getNbColonnes(); j++)
```

```
            {

                if (i == j)
                {
                    valprop[i] = matrix.getValeur(i, i);
                }
            }
        }

        return dec_sort(valprop);
    }
    //----------------- nombre d'unite d'entree ------------------
    public double[] erreur_approx(double[] mat)
    {
        double[] lambda = new double[mat.Length];
        for (int i = 0; i < mat.Length; i++)
        {
            lambda[i] = Math.Sqrt(mat[i] + 1);
            Console.WriteLine("Erreur[" + i + "]= " + lambda[i]);
        }
        return lambda;
    }
    //------------------ retour au premier plateau de la courbe------
    public int nbUniteEntree(double[] lambda)
    {
        int count = 0;
        double erreur, c1, c2;
        c1 = lambda[0];
        c2 = lambda[1];

        for (int i = 2; i < lambda.Length; i++)
        {
            erreur = Math.Abs(c2 - c1);
            //Console.WriteLine("erreur" + erreur);
            if (erreur < 0.005)
            {
                count = i;// -1;
                break;
            }
            else
            {
                c1 = c2;
                c2 = lambda[i];
            }
        }
        return count;
    }
  }
}
```

A.6 : Class Matrix

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace affiche
{
    class Matrix
    {
        private double[,] m_contenue;
        private int m_nbLignes;
        private int m_nbColonnes;

        public Matrix(int lign, int col)
        {
            m_nbLignes = lign;
            m_nbColonnes = col;

            m_contenue = new double[m_nbLignes, m_nbColonnes];
        }

        public Matrix(double[,] mat)
        {
            m_nbLignes = mat.Length;
            m_nbColonnes = mat.Length;
            m_contenue = mat;
        }

        // public Matrix(double[] vecteur)
        //{
        // m_nbLignes = 1;
        //m_nbColonnes = vecteur.Length;
        //m_contenue = new double[m_nbLignes,m_nbColonnes];
        //m_contenue = vecteur;
        //}

        public Matrix(Matrix a)
        {
            m_nbLignes = a.getNbLignes();
            m_nbColonnes = a.getNbColonnes();
            m_contenue = new double[m_nbLignes, m_nbColonnes];
            for (int i = 0; i < m_nbLignes; i++)
                for (int j = 0; j < m_nbColonnes; j++)
                {
                    setValeur(i, j, a.getValeur(i, j));
                }
        }

        //---------------multiplication matrice-----------------
        public Matrix multiplie(Matrix a, Matrix b)
        {
            Matrix resultat = new Matrix(a.getNbLignes(), b.getNbColonnes());
            if (a.getNbColonnes() != b.getNbLignes())
                throw new CannotUnloadAppDomainException("On ne pas multipier
                    les deux matrices");
```

```
    for (int i = 0; i < a.getNbLignes(); i++)
    {
        for (int j = 0; j < b.getNbColonnes(); j++)
        {
            double value = 0;
            for (int k = 0; k < b.getNbLignes(); k++)
            {
                value += a.getValeur(i, k) * b.getValeur(k, j);
            }
            resultat.setValeur(i, j, value);
        }
    }
    return resultat;
}

public Matrix multiplie(Matrix a, double b)
{
    Matrix resultat = new Matrix(a.getNbLignes(), a.getNbColonnes());

    for (int i = 0; i < a.getNbLignes(); i++)
    {
        for (int j = 0; j < a.getNbColonnes(); j++)
        {
            resultat.setValeur(i, j, a.getValeur(i, j) * b);
        }
    }

    return resultat;
}
//--------------------transpose---------------------
public Matrix transpose()
{
    Matrix resultat = new Matrix(m_nbColonnes, m_nbLignes);
    for (int i = 0; i < m_nbLignes; i++)
    {
        for (int j = 0; j < m_nbColonnes; j++)
        {
            resultat.setValeur(j, i, getValeur(i, j));
        }
    }
    return resultat;
}

public Matrix transpose(Matrix a)
{
    Matrix resultat = new Matrix(a.getNbColonnes(), a.getNbLignes());
    for (int i = 0; i < a.getNbLignes(); i++)
    {
        for (int j = 0; j < a.getNbColonnes(); j++)
        {
            resultat.setValeur(j, i, a.getValeur(i, j));
        }
```

```
    }
    return resultat;
}
//--------------matrice identite-----------------
public Matrix MatriceId(int dim)
{
    m_nbLignes = dim;
    m_nbColonnes = dim;
    Matrix Id = new Matrix(dim, dim);
    for (int i = 0; i < m_nbLignes; i++)
        for (int j = 0; j < m_nbColonnes; j++)
            Id.setValeur(i, j, (i == j) ? 1 : 0);

    return Id;
}
//-------------matrice unitaire--------------------
public Matrix MatriceUnit(int dim)
{
    m_nbLignes = dim;
    m_nbColonnes = dim;
    Matrix Unit = new Matrix(dim, dim);
    for (int i = 0; i < m_nbLignes; i++)
        for (int j = 0; j < m_nbColonnes; j++)
            Unit.setValeur(i, j, 1.0);

    return Unit;
}
//-------------max ligne et max colonne--------------
public int max_ligne(Matrix m)
{
    double max = m.getValeur(0, 1);
    int r_max = 0;
    for (int i = 0; i < m.getNbLignes(); i++)
        for (int j = 0; j < m.getNbColonnes(); j++)
            if (i != j)
            {
                if (m.getValeur(i, j) >= max)
                {
                    max = m.getValeur(i, j);
                    r_max = i;
                }

            }
    return r_max;
}

//---------------recherche colonne max ---------------
public int max_colonne(Matrix m)
{
    double max = m.getValeur(0, 1);
    int c_max = 0;
    for (int i = 0; i < m.getNbLignes(); i++)
        for (int j = 0; j < m.getNbColonnes(); j++)
```

```
            if (i != j)
            {
                if (m.getValeur(i, j) >= max)
                {
                    max = m.getValeur(i, j);
                    c_max = j;
                }

            }
    return c_max;
}
//--------------getter et setter------------
public double[,] getContenue()
{
    return m_contenue;
}
public void setContenue(double[,] ct)
{
    this.m_contenue = ct;
}

public int getNbLignes()
{
    return m_nbLignes;
}

public void setNbLignes(int lgn)
{
    this.m_nbLignes = lgn;
}
public int getNbColonnes()
{
    return m_nbColonnes;
}
public void setNbColonnes(int col)
{
    this.m_nbColonnes = col;
}
//--------------------------------------
public double getValeur(int i, int j)
{
    if (i >= m_nbLignes)
        throw new CannotUnloadAppDomainException("Nombre de lignes en
            dehors de la limite");
    if (j >= m_nbColonnes)
        throw new CannotUnloadAppDomainException("Nombre de colonnes en
            dehors de la limite");

    return m_contenue[i, j];
}

public void setValeur(int i, int j, double value)
{
```

```
            if (i >= m_nbLignes)
                throw new CannotUnloadAppDomainException("Nombre de lignes en
                    dehors de la limite");
            if (j >= m_nbColonnes)
                throw new CannotUnloadAppDomainException("Nombre de colonnes en
                    dehors de la limite");

            m_contenue[i, j] = value;
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace affiche
{
    public partial class ValRungeKutta : Form
    {
        private List<double> xList = new List<double>(5000);
        private List<double> zList = new List<double>(5000);
        private List<double> yList = new List<double>(5000);
        double[] x;
        public ValRungeKutta()
        {
            InitializeComponent();

        }

        private void dataGridView1_CellContentClick(object sender,
            DataGridViewCellEventArgs e)
        {
            //graphe(panel1);

        }
        private void button1_Click(object sender, EventArgs e)
        {
             runge_kutta rk = new runge_kutta(0, 50, 3, 5000);

            int l = 5000;

            int[] tablo = new int[4];
            tablo[1] = 0; // x = 0
            tablo[2] = 1; // y = 1
            tablo[3] = 0; // z = 0
            //tab_w1[0] = w[1];
            //tab_w2[0] = w[2];
            //tab_w2[0] = w[3];

            rk.condition_init(tablo);
            rk.step1();
            rk.step2();
            rk.step3();
            rk.step4();


             x = new double[l];
```

```
            x = rk.get_tab_w1();
            xList = doubleToListe(x,l);
            double[] y = new double[l];
            y = rk.get_tab_w2();
            double[] z = new double[l];
            z = rk.get_tab_w3();
            zList = doubleToListe(z, l);
            for (int i = 1; i <= l; i++)
            {



                dataGridView1.Rows.Add(i, 0, x[i], y[i], z[i]);


            }


        }

        private List<double> doubleToListe(double[]var, int l)
        {
            List<double> list = new List<double>(l);
            for (int i = 0; i < l; i++){
                list.Add(var[i]);

             }

                return list;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            ChartRungeKuta f2 = new ChartRungeKuta(xList,zList,x);
            f2.Show();
        }

        private void ValRungeKutta_Load(object sender, EventArgs e)
        {

        }
    }
}
```

B.2 : Class ChartRungeKutta

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
using System.Windows.Forms.DataVisualization.Charting;

namespace affiche
{
    public partial class ChartRungeKuta : Form
    {
        double[] xRungeKuta= new double[5000];
        Reseau reseau = new Reseau(2, 2, 1);
        Serie_temporelles serieTemporelles = new Serie_temporelles();
        public ChartRungeKuta(List<double> xList, List<double> zList,double[]
            xSerieTemporelles)
        {
            this.xRungeKuta = xSerieTemporelles;
            InitializeComponent();
            ChartCreation(zList, xList,chart1,"x=f(z)");
        }

        private void ChartRungeKuta_Load(object sender, EventArgs e)
        {

        }

        public void ChartCreation(List<double> zVal, List<double> xVal, Chart
            chrt, string nomSerie)
        {
            chrt.Series.Clear();
            ChartArea ca = new ChartArea();
            ca.AxisX.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisY.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisX.LineColor = Color.Blue;
            ca.AxisY.LineColor = Color.Blue;
            ca.AxisY.LineWidth = 100;
            ca.AxisX.LineWidth = 100;
            chrt.ChartAreas.Add(ca);

            Series sr = new Series { Name = nomSerie, ChartType =
                SeriesChartType.Line, Color = Color.Blue };

           // Series sr8 = new Series { Name = "mofo", ChartType =
               SeriesChartType.Line, Color = Color.Blue };


            for (int i = 0; i < zVal.Count; i++)
            {
                sr.Points.AddXY(zVal.ElementAt(i), xVal.ElementAt(i));
                //sr8.Points.AddXY(zVal.ElementAt(i)*10, xVal.ElementAt(i)*10);
            }

            chrt.Series.Add(sr);
            //chrt.Series.Add(sr8);
        }

        private void button1_Click(object sender, EventArgs e)
```

```
{
    serieTemporelles.setHorizon(2000);
    int prototype = 10;
    serieTemporelles.setL(5000);
    serieTemporelles.setPrototypeMax(prototype);
    serieTemporelles.matrice(100);
    serieTemporelles.matriceentre(xRungeKuta,2000);


    double[,] matriceEntree = serieTemporelles.getMatriceEntree();
    double[,] matriceSortie = serieTemporelles.getMatriceSortie();
    double[,] InN = new double[prototype, 2];
    double[,] OutN = new double[prototype, 1];
    reseau.setNProt(prototype);
    reseau.setPas(0.1);
    reseau.setScal(0.4);
    reseau.setEpoque(50);
    InN = reseau.normIn(matriceEntree);
    OutN = reseau.normOut(matriceSortie);
    double maxIn = reseau.returnMax(matriceEntree);
    double maxOut = reseau.returnMaxO(matriceSortie);
    double minIn = reseau.returnMin(matriceEntree);
    double minOut = reseau.returnMinO(matriceSortie);
    reseau.setMatIn(InN);
    reseau.setMatOut(OutN);
    reseau.apprendre();

    double[,] Z = reseau.propagation();


    double[] ZList = new double[5000];
    double[] Ountlist = new double[5000];

    int count = 0;

    for (int i = 0; i < prototype; i++)
        for (int j = 0; j < 1; j++)
        {
            ZList[count] = reseau.dnormOut(Z, maxOut, minOut)[i, j];
            Ountlist[count] = reseau.dnormOut(OutN, maxOut, minOut)[i, j
                ];
            count++;
        }

    NMSE f3 = new NMSE(ZList, Ountlist);
    f3.Show();
}


private void button2_Click(object sender, EventArgs e)
{
    Prediction prediction = new Prediction();
```

```
    prediction.setNbUnpas(10);
    double[,] st = new double[prediction.getNbUnpas(), reseau.getSortie
        ()];
    st = prediction.predictionUnPas(prediction.getNbUnpas(),
        serieTemporelles.getHorizon() + reseau.getNProt() + 1, reseau,
        serieTemporelles);
    double maxS = reseau.returnMaxO(prediction.getMSortieUnpas());
    double minS = reseau.returnMinO(prediction.getMSortieUnpas());
    double[] SortieReseauList = new double[5000];
    double[] MatriceSortielist = new double[5000];
    reseau.setNProt(prediction.getNbUnpas());
    int count = 0;
    for (int i = 0; i < prediction.getNbUnpas(); i++)
        for (int j = 0; j < 1; j++)
        {
            SortieReseauList[count] = reseau.dnormOut(st, maxS, minS)[i,
                0];
            MatriceSortielist[count] = prediction.getMSortieUnpas()[i, j
                ];

            count++;
        }

    PredictionUnpas predictionUnPas = new PredictionUnpas(
        SortieReseauList, MatriceSortielist);
    predictionUnPas.Show();
}

private void button3_Click(object sender, EventArgs e)
{
    Prediction prediction = new Prediction();
    int prototype = 10;
    reseau.setNProt(prototype);
    prediction.setNbPluspas(30);
    double[,] sp = new double[prediction.getNbPluspas(), reseau.
        getSortie()];
    sp = prediction.predictionPlusieurPas(prediction.getNbPluspas(),
        serieTemporelles.getHorizon() + prototype + 1, reseau,
        serieTemporelles);
    reseau.setNProt(prediction.getNbPluspas());
    double maxSP = reseau.returnMaxO(prediction.getMSortiePluspas());
    double minSP = reseau.returnMinO(prediction.getMSortiePluspas());
    double[] SortieReseauList = new double[5000];
    double[] MatriceSortielist = new double[5000];
    int count = 0;
    for (int i = 0; i < prediction.getNbPluspas(); i++)
        for (int j = 0; j < 1; j++)
        {
            SortieReseauList[count] = reseau.dnormOut(sp, maxSP, minSP)[i
                , j];
            MatriceSortielist[count] = prediction.getMSortiePluspas()[i,
                j];
            count++;
```

```
            }
            PlusieurPas predictionplusierPas = new PlusieurPas(SortieReseauList,
                MatriceSortielist);
            predictionplusierPas.Show();
        }

        private void chart1_Click(object sender, EventArgs e)
        {

        }

        private void button4_Click(object sender, EventArgs e)
        {
            Architecture architecture = new Architecture();
            architecture.Show();
        }

        private void button5_Click(object sender, EventArgs e)
        {
            {
                int epoque = 0;
                while (epoque <= reseau.getEpoque())
                {
                    Console.WriteLine("Epoque = " + epoque);
                    double cout = reseau.getCout();
                    Console.WriteLine("Coutqw = " + cout);
                    reseau.FonctionCoutPrim();
                    reseau.miseAjour();
                    epoque++;
                    double[] coutqwList = new double[5000];
                    double[] epoqueqwlist = new double[5000];

                    int count = 0;
                    for (int i = 0; i < reseau.getEpoque(); i++)
                        for (int j = 0; j < 1; j++)
                        {
                            coutqwList[count] = cout;
                            epoqueqwlist[count] = epoque;
                            count++;
                        }
                }
                //Erreur er = new Erreur();
                //er.Show();

            }

        }


    }
}
```

B.3 : Class NMSE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace affiche
{
    public partial class NMSE : Form
    {
        public NMSE(double[] ZList, double[] Ountlist)
        {
            int n = Ountlist.Length;
            InitializeComponent();

            for (int i = 0; i <10; i++)
            {

                dataGridView1.Rows.Add(ZList[i], Ountlist[i]);
             }

            chart1.Series.Clear();
            ChartArea ca = new ChartArea();
            ca.AxisX.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisY.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisX.LineColor = Color.Blue;
            ca.AxisY.LineColor = Color.Blue;
            ca.AxisY.LineWidth = 100;
            ca.AxisX.LineWidth = 100;
            chart1.ChartAreas.Add(ca);

            Series sr = new Series { Name = "Valeur donnee par le RNA",
                ChartType = SeriesChartType.Line, Color = Color.Green };

            Series sr8 = new Series { Name = "Valeur attendue", ChartType =
                SeriesChartType.Line, Color = Color.Blue };


            for (int i = 0; i < 10; i++)
            {
                sr.Points.AddXY(i*500, ZList[i]*500);
                sr8.Points.AddXY(i*500, Ountlist[i]*500);
            }

            chart1.Series.Add(sr);
            chart1.Series.Add(sr8);
```

```
            /*for (int i = 0; i < n; i++) {
                chart1.Series.ElementAt(0).Points.AddXY(i, ZList[i]);
                chart1.Series.ElementAt(1).Points.AddXY(i, Ountlist[i]);
            }*/


        }

        private void NMSE_Load(object sender, EventArgs e)
        {

        }



        private void button2_Click(object sender, EventArgs e)
        {
            this.Hide();
        }

        private void chart1_Click(object sender, EventArgs e)
        {

        }

        private void chart1_Click_1(object sender, EventArgs e)
        {

        }

        private void dataGridView1_CellContentClick(object sender,
            DataGridViewCellEventArgs e)
        {

        }
    }
}
```

B.3 : Class Erreur

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace affiche
{
    public partial class Erreur : Form
    {
```

```
        public Erreur(double[] epoquelist, double[] coutList)
        {
            InitializeComponent();
            int n = 10;
            for (int i = 0; i < n; i++)
            {
                dataGridView1.Rows.Add(i, epoquelist[i], coutList[i]);


            }

            chart1.Series.Clear();
            ChartArea ca = new ChartArea();
            ca.AxisX.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisY.LabelStyle.Font = new Font("Calibri", 20);
            ca.AxisX.LineColor = Color.Blue;
            ca.AxisY.LineColor = Color.Blue;
            ca.AxisY.LineWidth = 100;
            ca.AxisX.LineWidth = 100;
            chart1.ChartAreas.Add(ca);

            Series sr = new Series { Name = "NMSE", ChartType = SeriesChartType.
                Line, Color = Color.Red };

            for (int i = 0; i < n; i++)

            {

                sr.Points.AddXY( i , epoquelist[i] );

            }

            chart1.Series.Add(sr);

        }

        private void chart1_Click(object sender, EventArgs e)
        {

        }

        private void dataGridView1_CellContentClick(object sender,
            DataGridViewCellEventArgs e)
        {

        }
    }
}
```

B.3 : Class PlusieurPas

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace affiche
{
    public partial class PlusieurPas : Form
    {
       public Reseau reseau ;
       public Serie_temporelles serieTemporelles ;
       private int m_pas;
       private Prediction prediction;
       public PlusieurPas(Serie_temporelles serieTemporelles, Reseau reseau)
        {
            this.reseau = reseau;
            this.serieTemporelles = serieTemporelles;
            prediction = new Prediction();
            InitializeComponent();


        }
        private void handlePrediction(int pas) {

            this.m_pas = pas;
            int prototype = 10;
            reseau.setNProt(prototype);
            prediction.setNbPluspas(pas);
            double[,] sp = new double[pas, reseau.getSortie()];

            sp = prediction.predictionPlusieurPas(pas, serieTemporelles.
                getHorizon() + prototype + 1, reseau, serieTemporelles);
            //double maxSP = reseau.returnMaxO(prediction.getMSortiePluspas());
            //double minSP = reseau.returnMinO(prediction.getMSortiePluspas());
            double[] SortieReseauList = new double[pas];
            double[] MatriceSortielist = new double[pas];
            int count = 0;
            for (int i = 0; i < prediction.getNbPluspas(); i++)
                for (int j = 0; j < 1; j++)
                {
                  // SortieReseauList[count] = reseau.dnormOut(sp, maxSP, minSP)
                      [i, j];
                    SortieReseauList[i] = sp[i, j];
                    MatriceSortielist[i] = prediction.getMSortiePluspas()[i, j];
                    count++;
                }
            showValueOnDataGridView(SortieReseauList, MatriceSortielist);
            chart(SortieReseauList, MatriceSortielist);
        }

        private void showValueOnDataGridView(double[] SortieReseauList, double
```

```
    [] MatriceSortielist)
{
    for (int i = 0; i < this.m_pas; i++) dataGridView1.Rows.Add(i,
        SortieReseauList[i], MatriceSortielist[i]);
}

private void chart(double[] SortieReseauList, double[]
    MatriceSortielist)
{
    chart1.Series.Clear();
    ChartArea ca = new ChartArea();
    ca.AxisX.LabelStyle.Font = new Font("Calibri", 20);
    ca.AxisY.LabelStyle.Font = new Font("Calibri", 20);
    ca.AxisX.LineColor = Color.Blue;
    ca.AxisY.LineColor = Color.Blue;
    ca.AxisY.LineWidth = 100;
    ca.AxisX.LineWidth = 100;
    chart1.ChartAreas.Add(ca);

    Series sr = new Series { Name = "Valeur predite", ChartType =
        SeriesChartType.Line, Color = Color.Green };

    Series sr8 = new Series { Name = "Valeur attendue", ChartType =
        SeriesChartType.Line, Color = Color.Blue };


    for (int i = 0; i < this.m_pas; i++)
    {
        sr.Points.AddXY(i , SortieReseauList[i]);
        sr8.Points.AddXY(i , MatriceSortielist[i]);
    }

    chart1.Series.Add(sr);
    chart1.Series.Add(sr8);
}

private void button1_Click(object sender, EventArgs e)
{
    handlePrediction(3);
}

private void button2_Click(object sender, EventArgs e)
{
    handlePrediction(10);
}

private void button3_Click(object sender, EventArgs e)
{
    handlePrediction(20);
}

private void dataGridView1_CellContentClick(object sender,
    DataGridViewCellEventArgs e)
```

```
        {

        }

        private void chart1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

B.3 : Class Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace affiche
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new ValRungeKutta());
        }
    }
}
```

Titre : Prédiction de la série temporelle générée par le système de Lorenz par réseaux de neurones artificiels multicouches.

Résumé : Notre travail a pour but de prédire la série temporelle obtenue par le système dynamique chaotique de Lorenz en utilisant le réseaux de neurones artificiels multicouches. De ce fait, nous nous intéressons au cas d'un réseau à propagation de l'information vers l'avant avec une seul couche cachée. Nous utilisons deux méthodes de prédiction ; la prédiction à un pas en avant, qui consiste à prédire une seule valeur future de la série et la prédiction à plusieurs pas en avant qui nous donne des prédictions de plusieurs valeurs futures de la série. Nous avons constaté que dans la prédiction à un pas en avant, les valeurs prédites et les valeurs attendues coïncident, tandis que lors des prédictions à plusieurs pas en avant, celles-ci divergent, preuve du caractère chaotique du système de Lorenz.

Mots clés : Réseaux de neurones artificiels, rétro-propagation, système de Lorenz, séries temporelles chaotiques, prédiction.

Title : Prediction of the time series generated by the Lorenz system by multilayer artificial neural networks.

Abstract : Our work aims to predict time series obtained by Lorenz chaotic dynamic system using multilayer neural networks. Therefore, we are interesting in the case of feed forward neural network with a single hidden layer. In this work, we use two methods of prediction : one step prediction what predict only one future value of the series and a long term prediction what give us prediction of several future values of the series. In prediction, forward, we have notice that the predicted values and the expected values coincide, while in predictions by several steps forward, these diverge, prove of the chaotic character of the system of Lorenz.

Keywords : Artificial neural networks, back propagation, Lorenz system, chaotic time series, prediction.

Encadreur :
Mr RABOANARY Roland
Professeur Titulaire
Département de Physique
facultés des sciences

Impétrant :
RAKOTONOAVY Arinoro Charles
Email : noavy018@gmail.com
Tel :(+261) 34 08 700 08
CU Ankatso II Bloc 06 Porte 01
Tana 101, Madagasikara