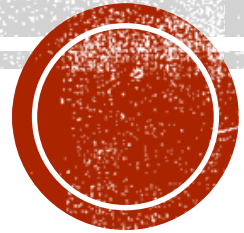
PRÉDICTION DU TEMPS D'ARRIVÉE LE BUS : P.T.A.B



TIPE : THAOUI MEHDI

Numéro d'inscription : 29919

PLAN DE LA PRÉSENTATION

- **Réseau neurones artificielles:**
  - -Définition du neurone formel
  - -Définition de l'apprentissage supervisé et ses types
  - -Algorithme de perceptron multicouche et de rétropropagation.
  - -Méthode de descente de gradient à pas optimale.
- **Théorie du chaos et application logistique:**
  - -Définition
  - -Observations
  - -Bifurcations
  - -Exposant de Lypanouv
- **Série temporelle et prédiction:**
  - -Définition et méthodes
  - -Type de prédiction
  - -Simulation et résultats

MOTIVATION

- Importance d'améliorer le transport en commun et de promouvoir l'usage ont contraint les entreprises du secteur à s'adapter au progrès technologique.

- Nécessité de prévoir le temps de trajet de bus afin de bien le planifier et donner une information judicieuse aux citoyens.

OBJECTIF

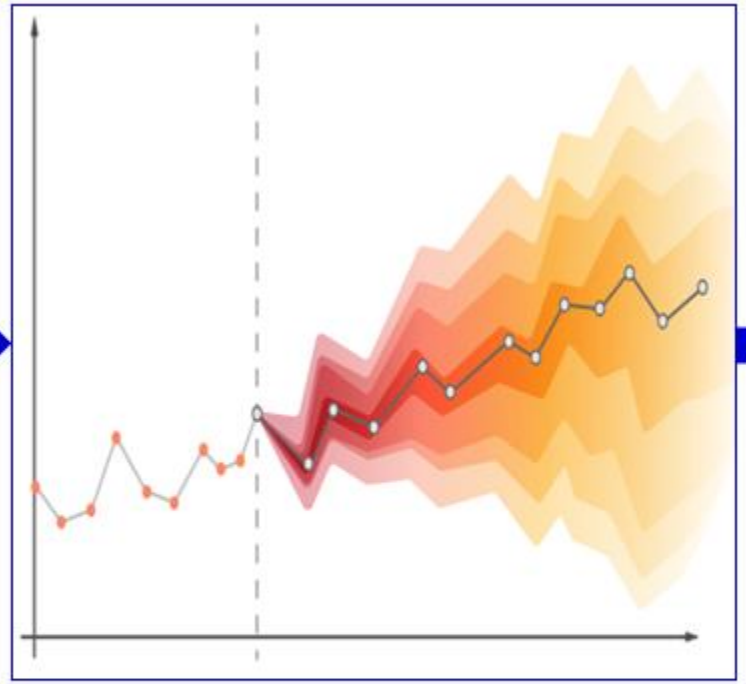
- Prédiction du temps d'arrivée des bus avec une marge d’erreur minimale

Figure 1.1 : Une simple application sur mobile pour prévoir le temps d’arrivée du bus

APPROCHE MÉTHODOLOGIQUE

- Machine learning est utilisé pour aider à la prédiction du temps, en développant un modèle capable d'estimer le temps restant avant l'arrêt donné.

Collecter les données comme localisation des bus, la vitesse du bus, les données météorologiques, le nombre des passagers



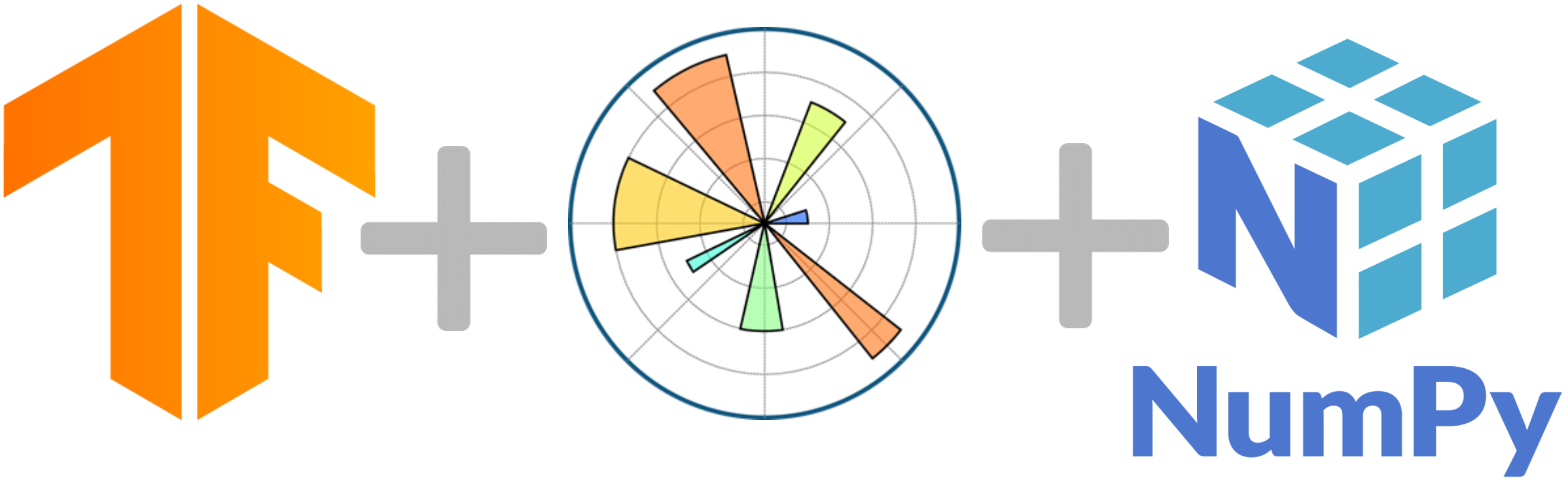
Entraîner un modèle de prédiction en choisissant un algorithme convenable de la machine learning

Donner l'heure d'arrivée de bus

Figure 1.2 : Le processus de prévision le temps de bus

APPROCHE MÉTHODOLOGIQUE

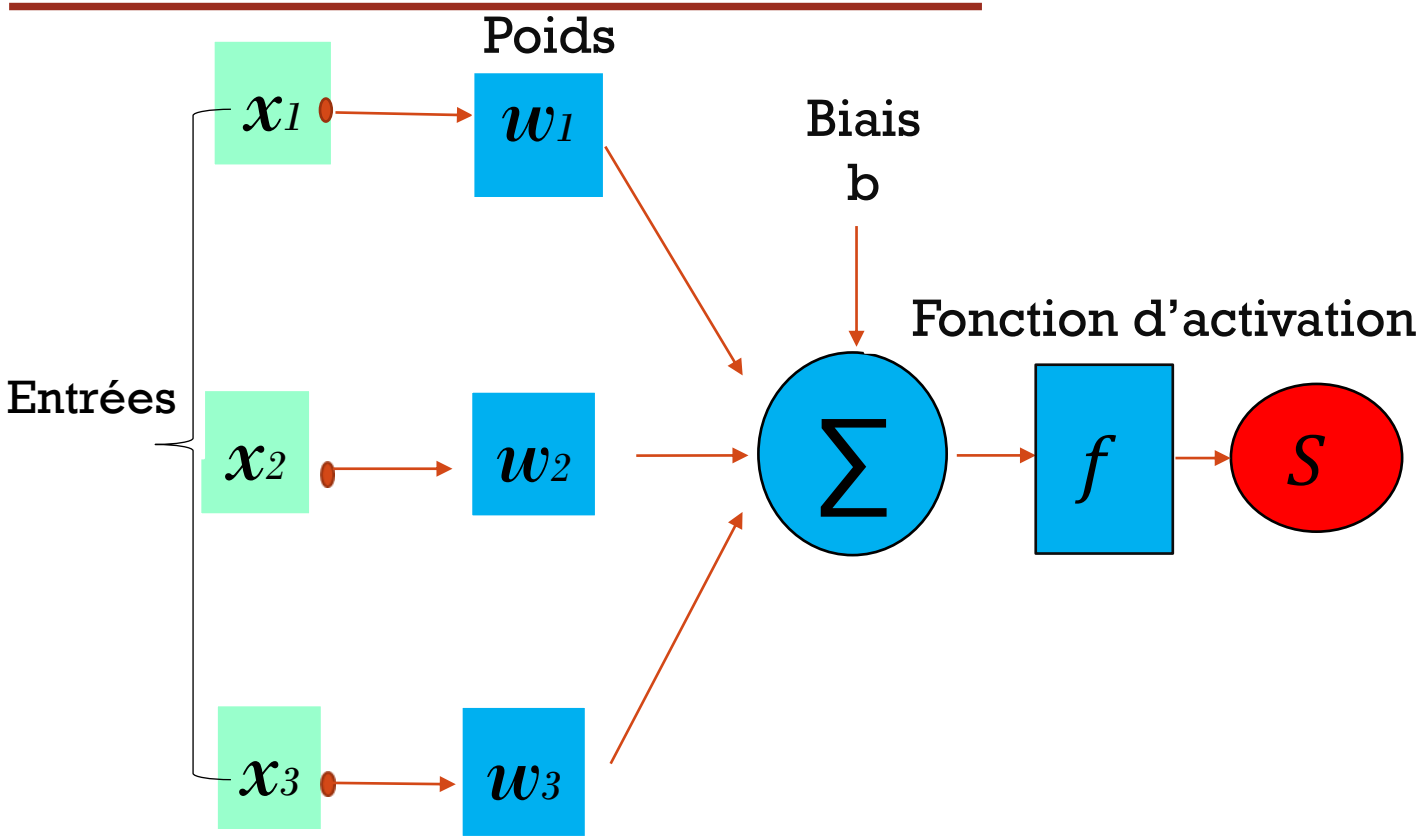
- Développer des modèles prédictifs en utilisant des techniques telles que les séries chronologiques et les réseaux neuronaux.



- **Figure 1.3 :** les trois bibliothèques essentielles pour notre prédiction.

RÉSEAU NEURONES ARTIFICIELS

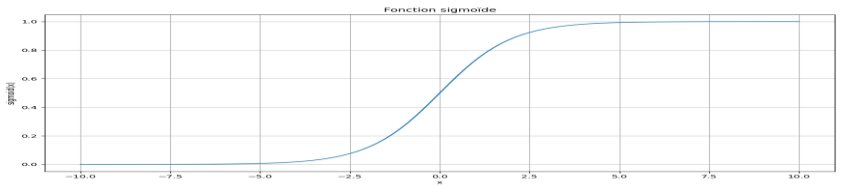
La neurone formel



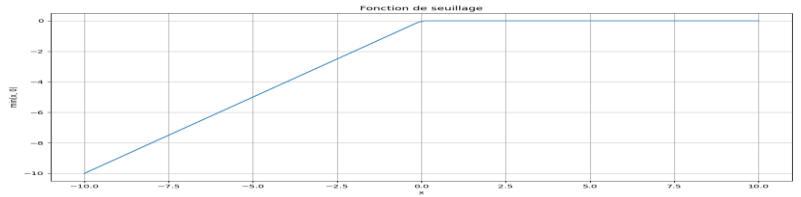
$$S = f\left(\sum_{i=0}^{n} x_i w_i + b\right)$$

Fonction d'activation f :

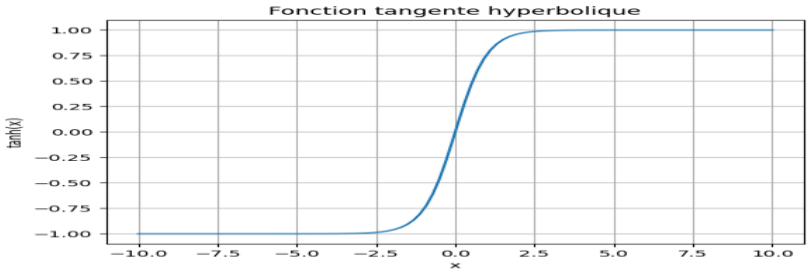
- Sigmoïde: $f(x) = \frac{1}{1+e^{-x}}$



- Relu : $f(x) = \min(x, 0)$



- Tangente hyperbolique: $f(x) = \tanh(x)$



APPRENTISSAGE SUPERVISÉ

Définition:

- La phase d'apprentissage supervisée consiste à comparer la sortie estimée parle réseau avec la sortie désirée et mettre à jour les poids synaptiques jusqu'à ce que le réseau donne un résultat acceptable.

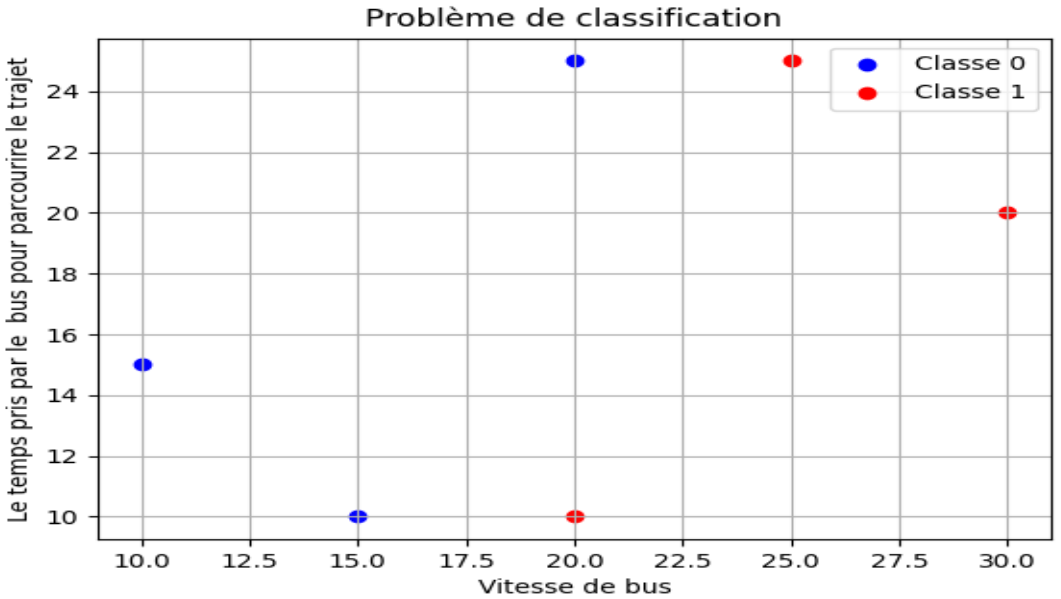


Figure 2.1: Problème de classification

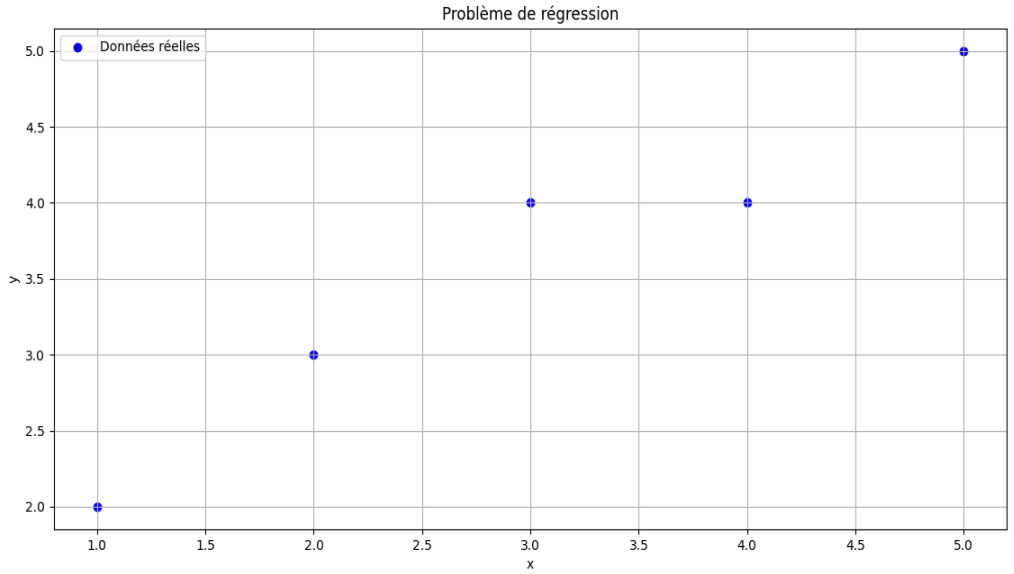


Figure 2.2: Problème de régression

APPRENTISSAGE SUPERVISÉ

Les types d'apprentissage supervisé:

- Régression: Il s'agit de trouver une relation entre les variables d'entrée et de sortie, d'où une prédiction des valeurs numérique.
- Classification: C'est un modèle qui consiste à prédire des catégories discrètes ou des étiquettes de classe.

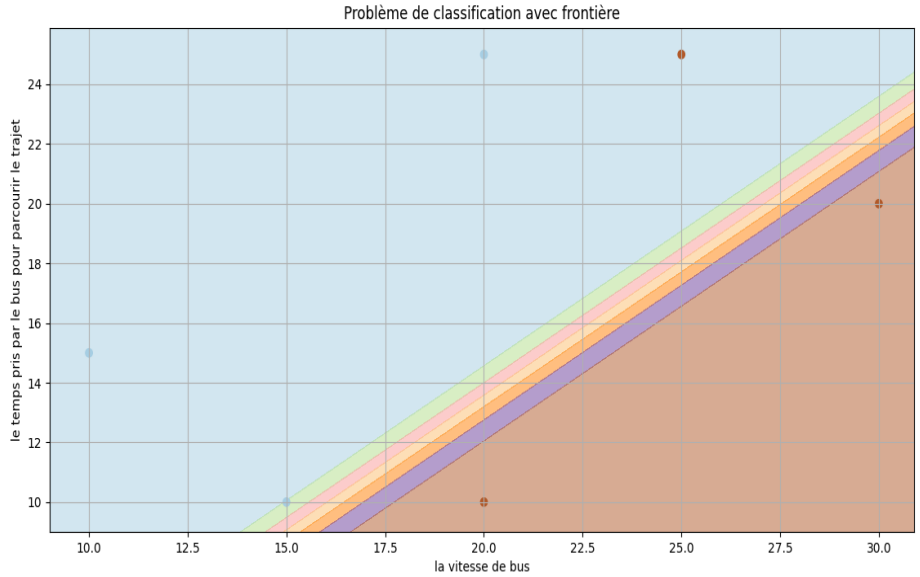


Figure 2.3 : Frontière de décision

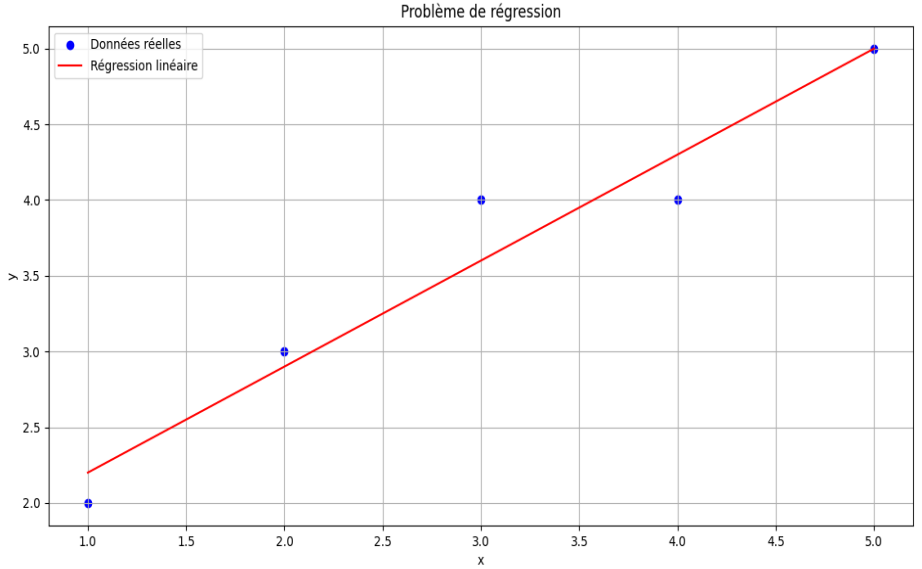


Figure 2.4 : Fonction de régression

ALGORITHME PERCEPTRON MULTICOUCHE

- **Étapes de l'algorithme de perceptron multicouche**

I. Initialisation des poids

II. Phase de propagation avant

a. Les valeurs d'entrée sont introduites dans le réseau neuronal.

a. Les calculs se propagent à travers les couches du réseau.

b. Chaque neurone effectue une somme pondérée des valeurs d'entrée et applique une fonction d'activation pour produire sa sortie.

III. Calcul de l'erreur

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2} \qquad MAE = \frac{1}{n}\sum_i |y_i - \hat{y}_i|$$

IV. Phase de rétropropagation de l'erreur

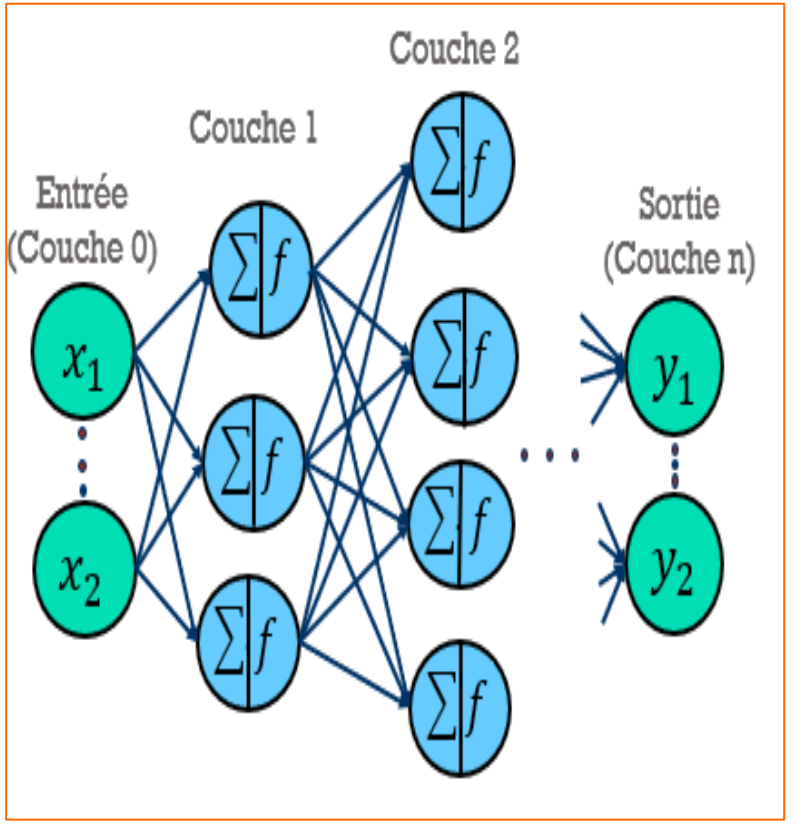
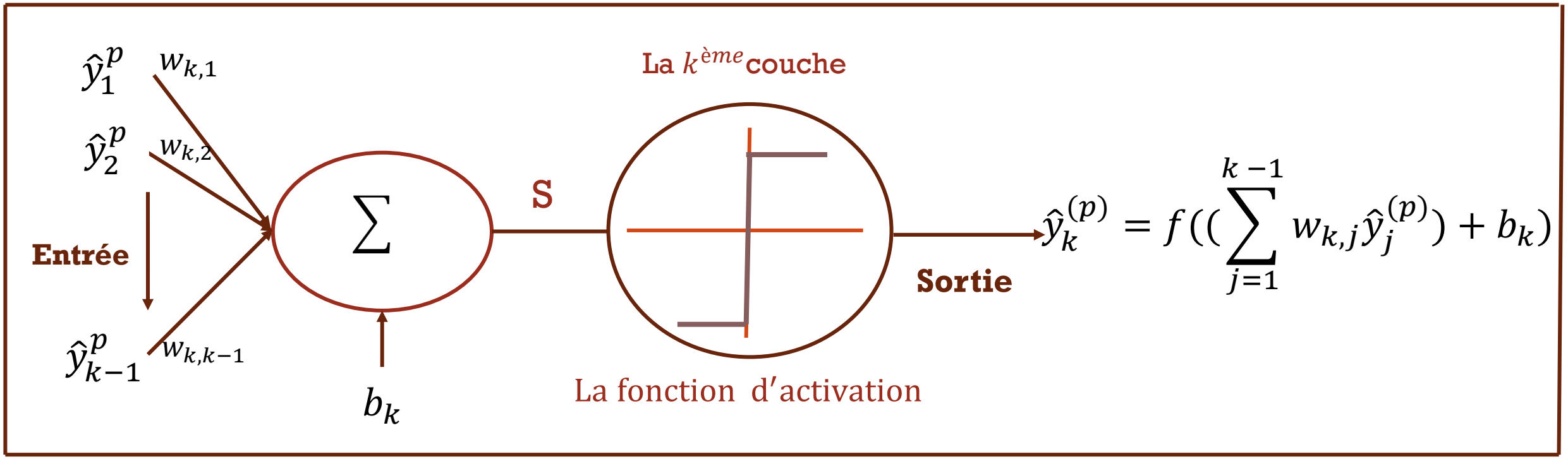


Figure 2.5 : Perceptron multicouche

ALGORITHME DU RÉTROPROPAGATION

- L'erreur entre la sortie du réseau et la sortie désirée : $e_k^p = (y_k^p - \hat{y}_k^p)$
- La fonction d'erreur : $E^P = \frac{1}{2}\sum_k (e_k^p)^2$



Cherchons $\Delta_p w_{kj} = -\eta \frac{\partial E^p}{\partial w_{kj}}$.

ALGORITHME DE RÉTROPROPAGATION

Utilisons la formule de la chaîne : $\frac{\partial E^p}{\partial w_{kj}} = \frac{\partial E^p}{\partial e_k^p}\frac{\partial e_k^p}{\partial \hat{y}_k^p}\frac{\partial \hat{y}_k^p}{\partial v_k^p}\frac{\partial v_k^p}{\partial w_{kj}}$

avec : $\frac{\partial e_k^p}{\partial \hat{y}_k^p} = -1$ | $\frac{\partial \hat{y}_k^p}{\partial v_k^p} = f'(v_k^p)$ | $\frac{\partial v_k^p}{\partial w_{kj}} = \hat{y}_j^p$ | $\frac{\partial E^p}{\partial e_k^p} = e_k^p$

Alors $\Delta_p w_{k,j} = \eta \delta_k^p \hat{y}_j^p$ avec $\delta_k^p = \frac{\partial E^p}{\partial v_k^p} = -e_k^p f'(v_k^p)$

- Connexion Couche d'entrée couche cachée:

D'après la règle de chaîne : $\delta_j^p = \frac{\partial E^p}{\partial \hat{y}_j^p}\frac{\partial \hat{y}_j^p}{\partial v_j^p}$ .Or on a aussi: $\frac{\partial \hat{y}_j^p}{\partial v_j^p} = f'\left(v_j^p\right)$

$E^p = \frac{1}{2}\sum_k (e_k^p)^2 \Longrightarrow \frac{\partial E^p}{\partial \hat{y}_j^p} = \sum_k e_k^p \frac{\partial e_k^p}{\partial \hat{y}_j^p} = \sum_k e_k^p \frac{\partial e_k^p}{\partial v_k^p}\frac{\partial v_k^p}{\partial \hat{y}_j^p}$. Or $e_k^p = y_k^p - f(v_k^p)$

$\frac{\partial e_k^p}{\partial v_k^p} = -f'(v_k^p)\ \frac{\partial v_k^p}{\partial \hat{y}_j^p} = w_{kj}$; Donc , $\frac{\partial E^p}{\partial \hat{y}_j^p} = -\sum_k e_k^p f'(v_k^p) w_{kj} = -\sum_k \delta_k^p w_{kj}$

Finalement nous trouvons : $\delta_j^p = -f'\left(v_j^p\right)\sum_k \delta_k^p w_{kj}$, $\Delta_p w_{ji} = \eta \delta_j^p \hat{y}_i^p$

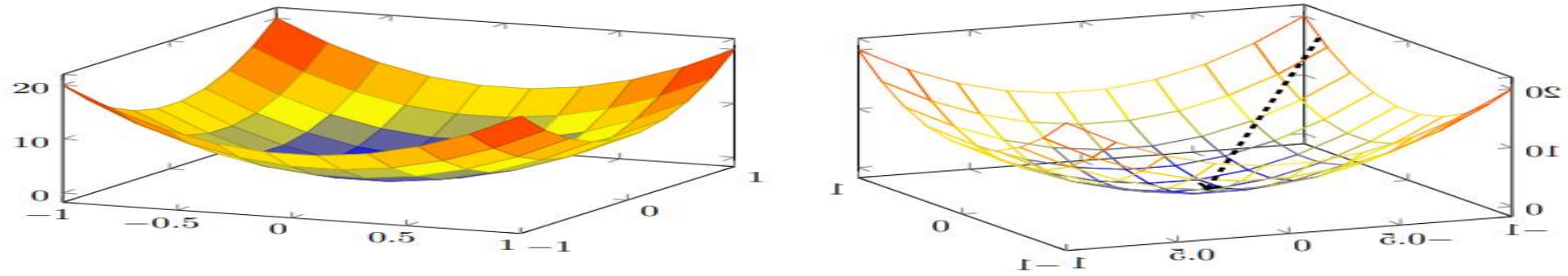
MÉTHODE DE DESCENTE DE GRADIENT

- **Objectif:**

$$\begin{cases} \text{Trouver } x^\star \in \mathbb{R}^n \text{ tel que:} \\ f(x^\star) = \min\limits_{x \in \mathbb{R}^n} f(x) \end{cases}$$

avec f: $\mathbb{R}^n \to \mathbb{R}$, j de classe C^1 au moins (C^2 si besoin)

- Exemple : une fonction quadratique strictement convexe (le cas idéal)



- Condition Nécessaire : $\nabla f(x) = 0$
- Condition Suffisante : f strictement convexe

MÉTHODE DE DESCENTE DE GRADIENT

- **Application aux cas de la résolution d'un système linéaire**

- **Théorème:** Soit $A \in S_n^{++}(\mathbb{R}), AX_0 = B \Leftrightarrow X_0$ minimise La fonction $f : \mathbb{R}^n \to \mathbb{R}$:

$$\frac{1}{2} < AX, X > - < B, X >$$

MÉTHODE DE DESCENTE DE GRADIENT

- **Principe (1):**

On part x_0 donné par l'utilisateur, on définit x_{n+1}:

$$f(x_{n+1}) < f(x_n)$$

Pour cette raison, on doit déterminer :

- La direction de descente $d^{(n)}$
- Le pas de descente $\alpha^{(n)}$

Puis $x_{n+1} = x_n + \alpha^{(n)} d^{(n)}$

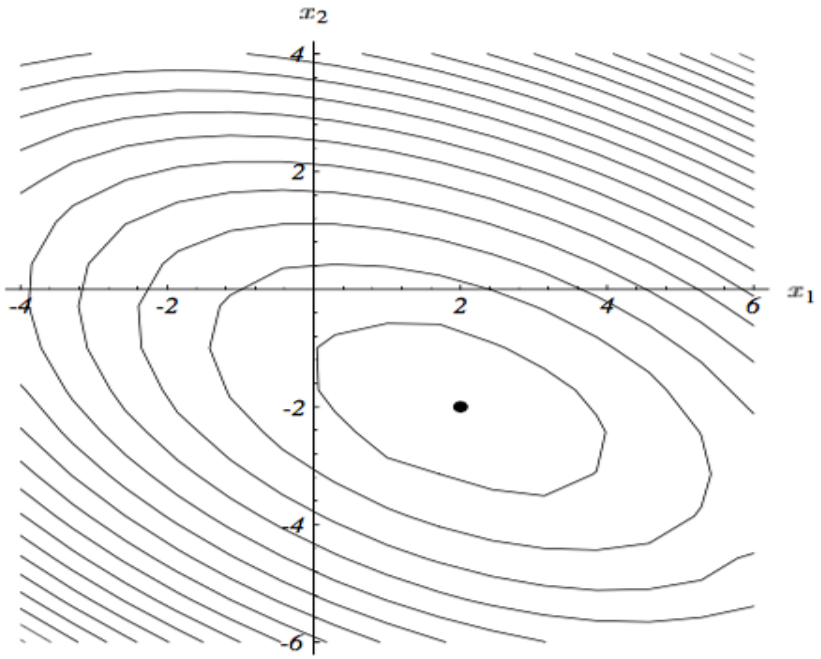


Figure 3.1 : Contours de forme quadratique. chaque courbe ellipsoïdale a une constante $f(x)$

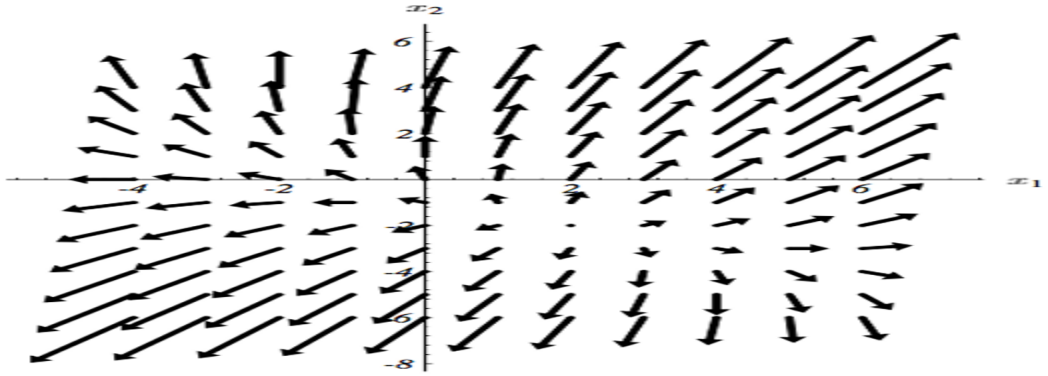


Figure 3.2 : Gradient de f′(x) de la forme quadratique. P*our* chaque x, le gradient des points est dans le sens de la plus forte augmentation de $f'(x)$, et est orthogonal aux courbes de niveau

MÉTHODE DE DESCENTE DE GRADIENT

- **Principe (2):**

Pas de descente α : recherche linéaire.

Direction de descente :

À chaque étape, la direction d_k est une direction descendante, autrement dit, $\min_{[0,b]} g_k(t) = \min_{[0,b]} f(x_k + td_k) < f(x_k)$

MÉTHODE DE DESCENTE DE GRADIENT

- **Convergence:**

Définition: Une suite $(x_k)_k$ de $\mathbb{R}^n$ est dite :

- stationnaire pour f si $\lim_{k\to+\infty} \nabla f(x_k) = 0$
- minimisante pour f si $\lim_{k\to+\infty} f(x_k) = \inf_{x\in\mathbb{R}^n} f(x)$

- **Proposition 1:** si f est convexe et différentiable sur $\mathbb{R}^n$, alors toute suite bornée stationarisante pour f est minimisante pour f .
- **Proposition 2:** si f continuement différentiable sur $\mathbb{R}$ et coersive, alors la suite $(x_k)_k$ est bornée, admet au moins une soussuite convergente et toute suite convergente et stationnaire.

Théorème

Supposons que la fonction f est convexe et différentiable sur $\mathbb{R}^n$:

- Si f est coersive, alors la suite $(x_k)_k$ admet au moins une soussuite convergente et toutes les soussuites convergentessont minimisantes.
- Si f est fortement convexe, alors la suite $(x_k)_k$ est minimisante et convergevers une une solution optimale d'un problème sans contraintes.

ALGORITHME DE DESCENTE DE GRADIENT À PAS OPTIMALE

- **Algorithme de méthode de gradient à pas optimale**

Fonction(Gradient-Optimal)$(X_0 \in \mathbb{R}^n, R_0 = AX_0 - B)$

Tant que $\frac{\|R_n\|}{\|R_0\|} \preccurlyeq \varepsilon, faire$:

$$\alpha_{n+1} \leftarrow \frac{\|R_n\|^2}{<AR_n, R_n>} \ , \ X_{n+1} \leftarrow X_n - \alpha_{n+1} R_n \ , R_{n+1} \leftarrow AR_n - B$$

Fin tant que

Fin

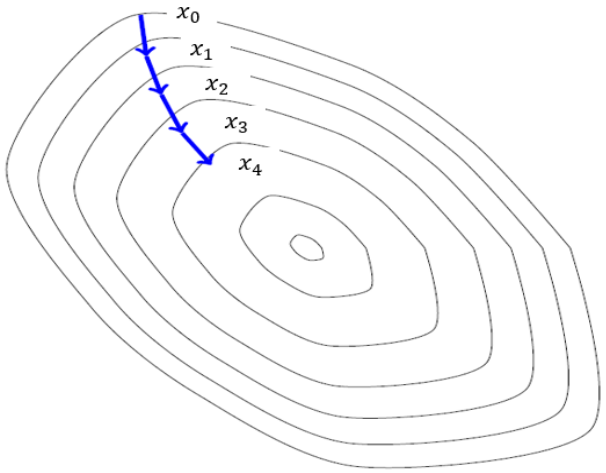


Figure 3.3 : illustration de la méthode du gradient

- **Théorème:**

Soit A de $S_n^{++}(\mathbb{R})$ et X_o. L'algorithme du gradient converge vers la solution $\overline{X}$ du système $AX = B$ et on a pour tout $n \in \mathbb{N}$,

$$\| X_n - X \| \preccurlyeq \left(\frac{M(A) - 1}{M(A) + 1} \right)^n \sqrt{M(A)} \ \| X_0 - \overline{X} \|$$

ALGORITHME DE DESCENTE DE GRADIENT À PAS OPTIMALE

- **C'est quoi la théorie du chaos?:** Le fait d'étudier le comportement des systèmes sensibles aux conditions initiales.
- **Définition de l'application logistique:** Le cas le plus simple d'un système dynamique discret, c'est-à-dire dépendant du temps.
  - La fonction f est l'application logistique

$$f \begin{cases} \mathbb{R} \to \mathbb{R} \\ \mu x(1-x) \end{cases} \quad \text{avec } \mu > 0$$

- La loi d'évolution est donnée par :

$$x_{n+1} = f(x_n) \text{ pour } n \geqslant 1 \text{ et } x_0 \in \mathbb{R} \text{ comme condition intiale}$$

OBSERVATIONS

- **Allure de la fonction:**

- Travaillons dans l'intervalle $[0,1]$avec $0 < \mu < 4$.
- $f'(x) = \mu(1 - 2x)$, d'où f prend sa valeur maximale en $\frac{1}{-}$ qui est égale à $\frac{\mu}{4}$.

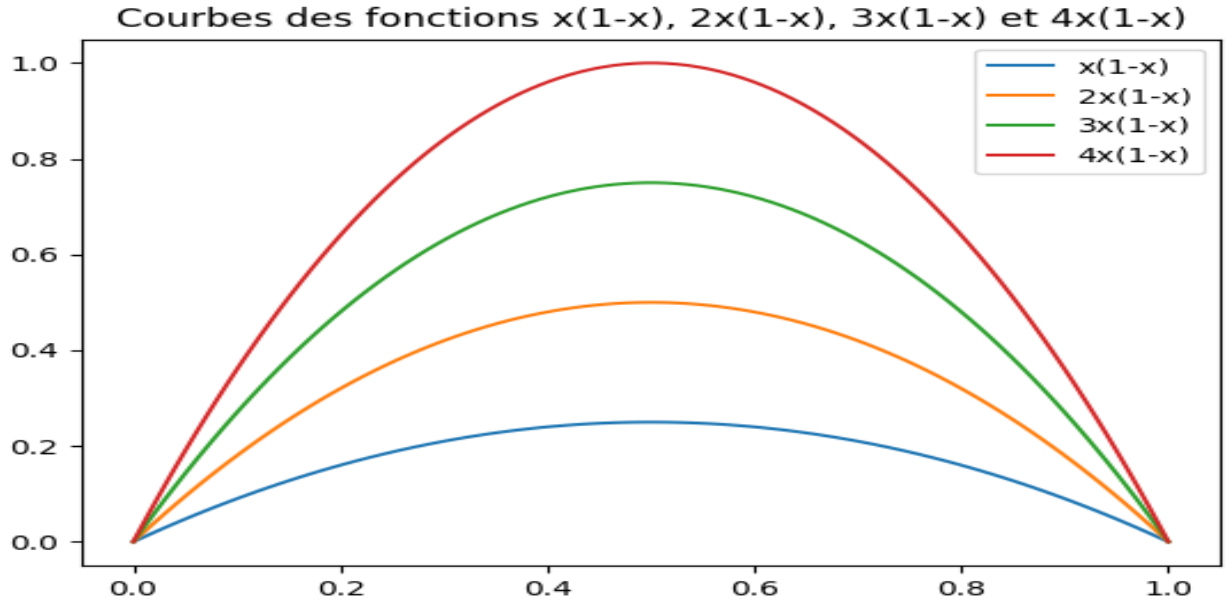


Figure 4.1 : Allure de l'application logistique pour différentes valeurs de μ

OBSERVATIONS

- **Orbite ou trajectoire d'un système:** L'ensemble de tous les points du système noté $O(x_0)$ où x_o le point initiale: $O(x_0) = \{x_0, x_1, \dots, x_n, \dots\}$
- **Point fixe un point est dit fixe:** Un point x est dit fixe ssi $f(x) = x$.

Pour notre cas $\mu x(1-x) = x \Leftrightarrow x_1 = 0$ ou $x_2 = 1 - \frac{1}{\mu}$

- Un point fixe x est dite attractive ssi : $\left|\frac{d}{dx} f(x)\right| < 1$
- Un point fixe x est dite attractive ssi : $\left|\frac{d}{dx} f(x)\right| > 1$

Si x_1 et x_2 sont attractifs alors:

- $\left|\frac{d}{dx} f(x_1)\right| = \mu(1-2x_1) = \mu < 1$
- $\left|\frac{d}{dx} f(x_2)\right| = \mu(1-2x_2) = |2-\mu| < 1$

OBSERVATIONS

- **Pour le système,** x_1 (resp x_2) est un point fixe attractif (resp répulsif) si $0 < \mu < 1$.
- **Pour le système,** x_1 (resp x_2) est un point fixe répulsif (resp attractif) si $1 < \mu < 3$.

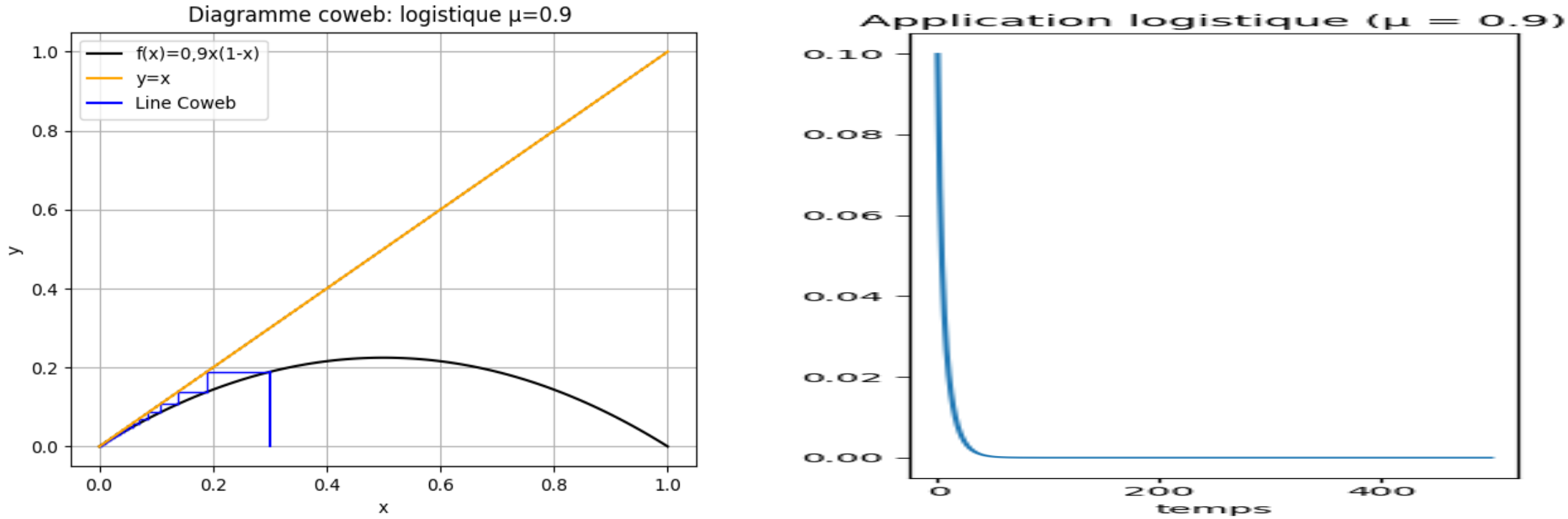
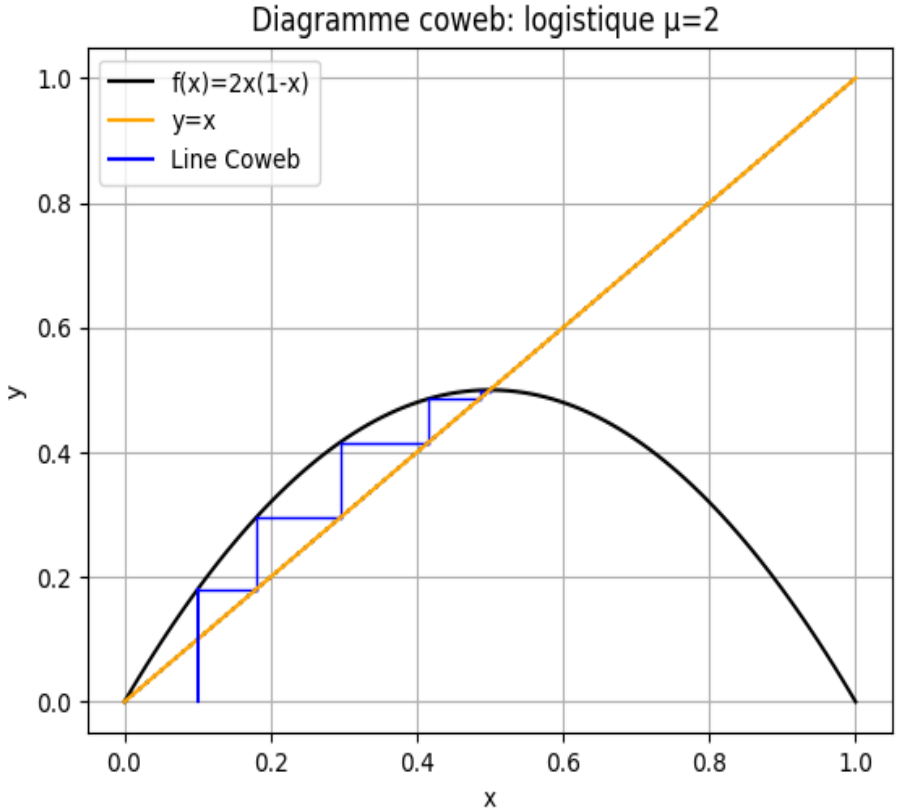


Figure 4.2 : Attracteur pour $x_1 = 0$

OBSERVATIONS



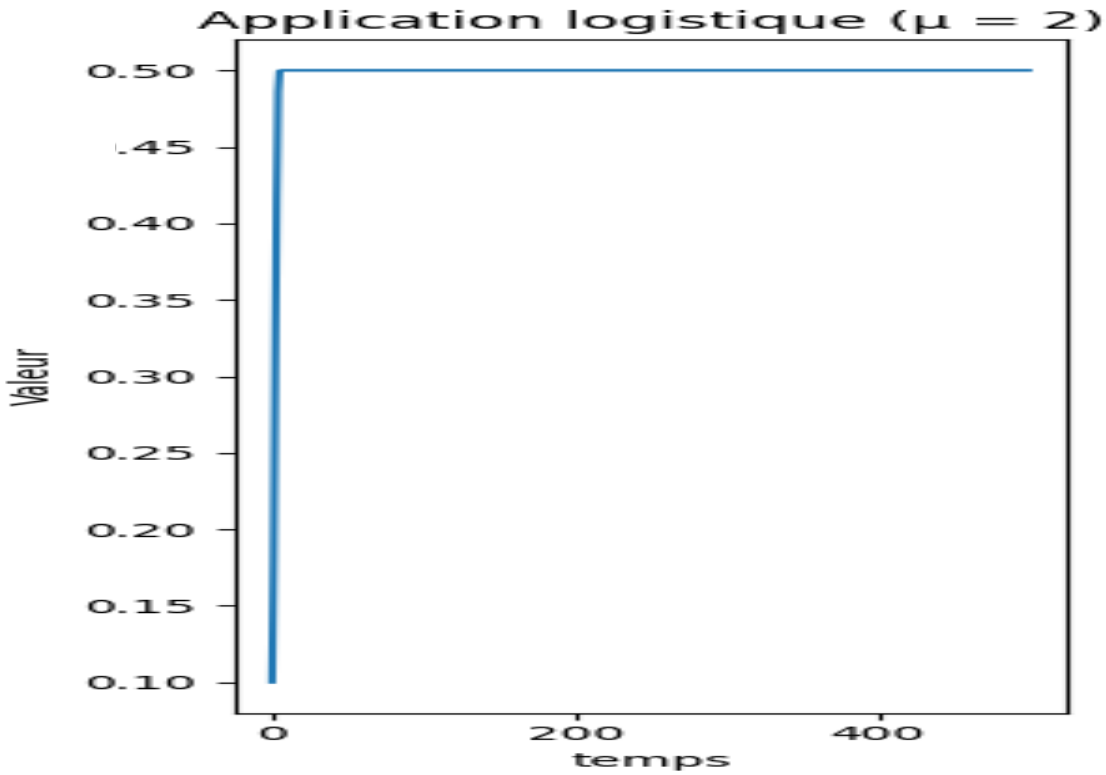


Figure 4.3 : $x_2 = 1 - \frac{1}{\mu}$ attracteur et répulsif pour $x_1 = 0$

OBSERVATIONS

- **Points et Orbites périodique:**

➢ Une orbite $O(x_0)$ est dite périodique s'il existe $p > 0$,tel que :
$\forall n \in \mathbb{N}, x(n+p) = x(n)$ .On parle d'une suite de période p.

➢ Tous les points de période p sont solution de l'équation $f^{(p)}(x) = x$ avec
$f^{(p)}(x) = f(f\left(f(\dots(f(x))\dots)\right))$

Pour le cas p=2

- $f\big(f(x)\big) = x \Rightarrow \mu^2 x(1-x)\big(1-\mu x(1-x)\big) = x$
$\Longrightarrow \mu^3 x^4 - 2\mu^3 x^3 + \mu^2(1+\mu)x^2 - (\mu^2-1)x = 0$

x_1 et x_2 sont déjà racines de ce polynôme. Après factorisation, les points périodiques sont solutions de l'équations:

$$\mu^2 x^2 - (\mu^2 - \mu)x + \mu + 1 = 0$$

Ce qui nous donne: $x_3 = \frac{\mu+1-\sqrt{(\mu-3)(\mu+1)}}{2\mu}$ et $x_4 = \frac{\mu+1+\sqrt{(\mu-3)(\mu+1)}}{2\mu}$

La périocidité existe si et seulement si $\mu \geqslant 3$ et il existe un seul $\mu = 3$ et deux distincts si $\mu > 3$

OBSERVATIONS

- **Orbite attractive et orbite répulsive:**

➢ Un orbite $O(x_0)$ est attractive $\left|\prod_{j=0}^{p-1} f'(x(j))\right| < 1$

➢ Un orbite $O(x_0)$ est attractive $\left|\prod_{j=0}^{p-1} f'(x(j))\right| > 1$

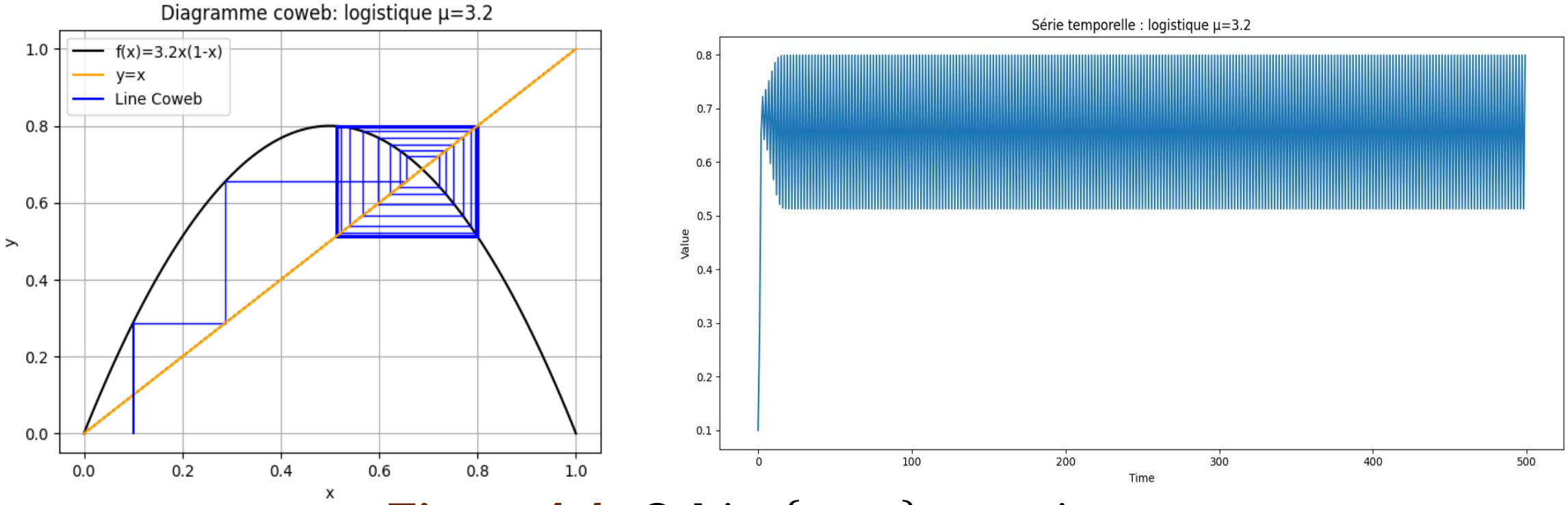
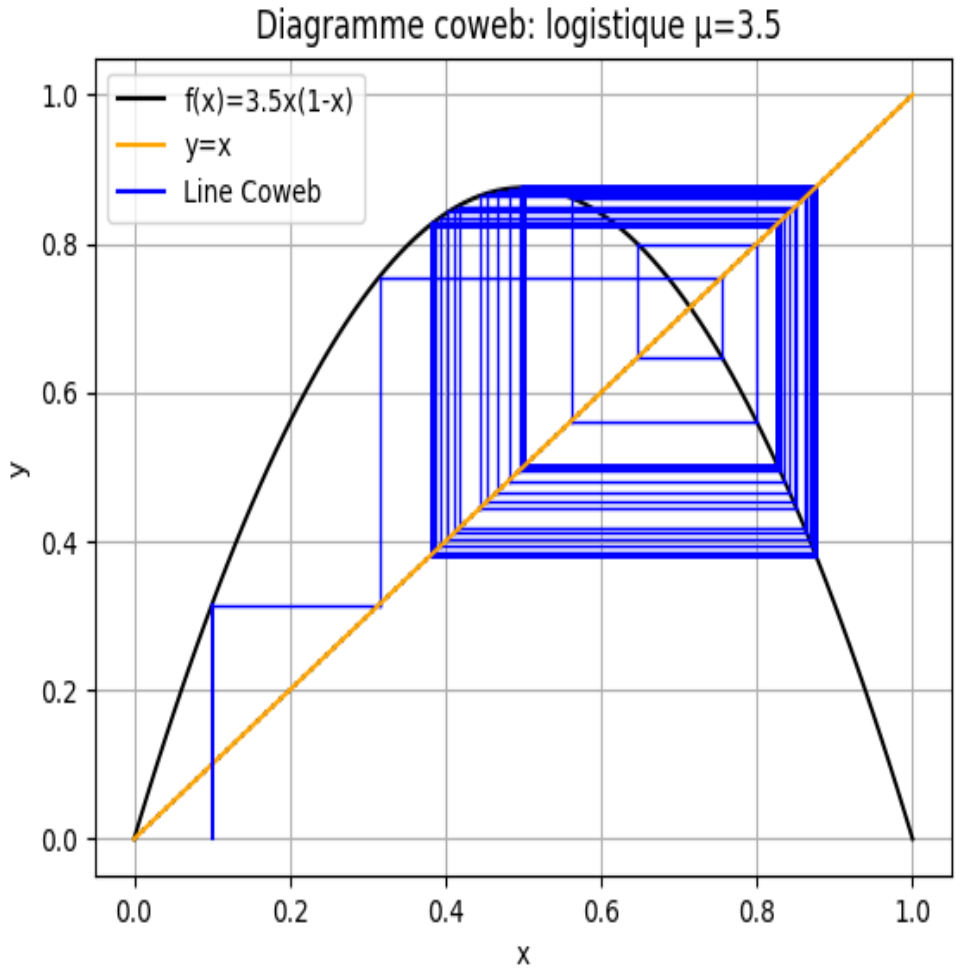


Figure 4.4 : Orbite $\{x_3, x_4\}$ attractive

OBSERVATIONS



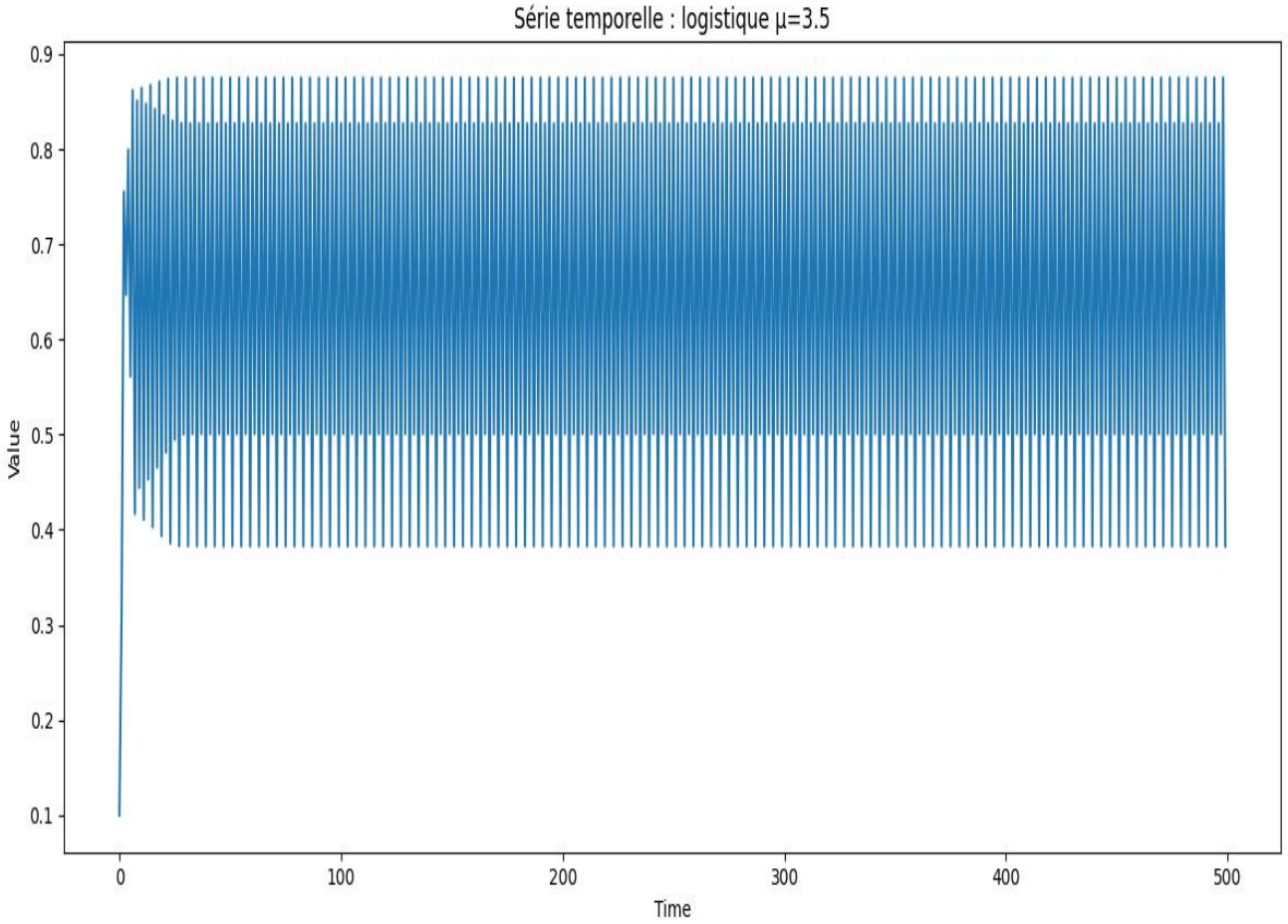


Figure 4.5 : Orbite $\{x_3, x_4\}$ répulsive

BIFURCATIONS

- **Définition:** Le diagramme de bifurcation représente graphiquement les valeurs d'équilibre en fonction d'un paramètre donné. Il met en évidence les changements qualitatifs du comportement du système lorsque le paramètre varie.

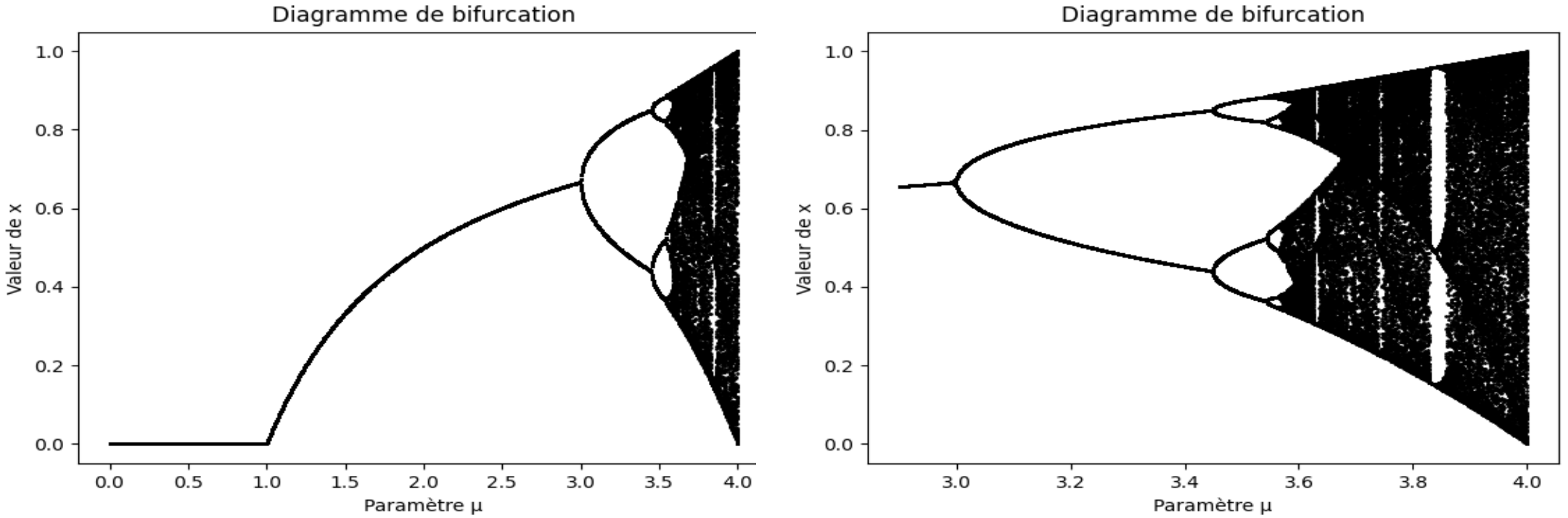


Figure 4.6 : Diagramme de bifurcation

LES EXPOSANTS DE LYAPOUNOV

- **Définition:** L'exposant de Lyapunov pour l'application logistique est une mesure de la sensibilité aux conditions initiales d'un système dynamique .
- La formule de l'exposant de Lyapunov pour l'application logistique est donnée par :

$$\lambda(x_0) = \lim_{n\to+\infty} \frac{1}{n} \sum_{i=1}^{n-1} \ln|f'(x(i))|$$

- Si $\lambda(x_0) < 0$, les orbites du système sont stables.
- Si $\lambda(x_0) > 0$, les orbites du système sont instables , cela une sensibilité aux conditions initiales et suggère un comportement chaotique du système.

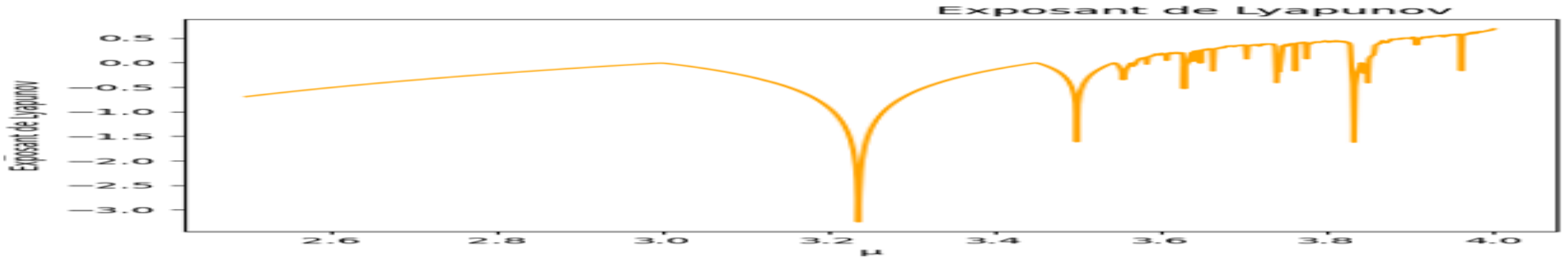


Figure 4.6 : Exposant de Lypanouv

CRÉATION D'UNE BASE DE DONNÉES

- Collecter le temps qu'il prend le bus de la ville d'Agadir chaque jour pour parcourir un trajet déterminé avec la vitesse avec laquelle il se déplace et le nombre des passagers.

- Ecrire un programme qui donne une prédiction avec ses deux types sur le temps d'arrivée de bus dans le cas chaotique ou non.

- Optimiser et analyser la qualité des données.

SÉRIE TEMPORELLE

- **Définition:** Les séries temporelles sont des ensembles de données qui sont collectées, dans un ordre chronologique. Il s'agit d'une suite finie indexée par le temps $(x_1, x_2, \dots, x_n)$.

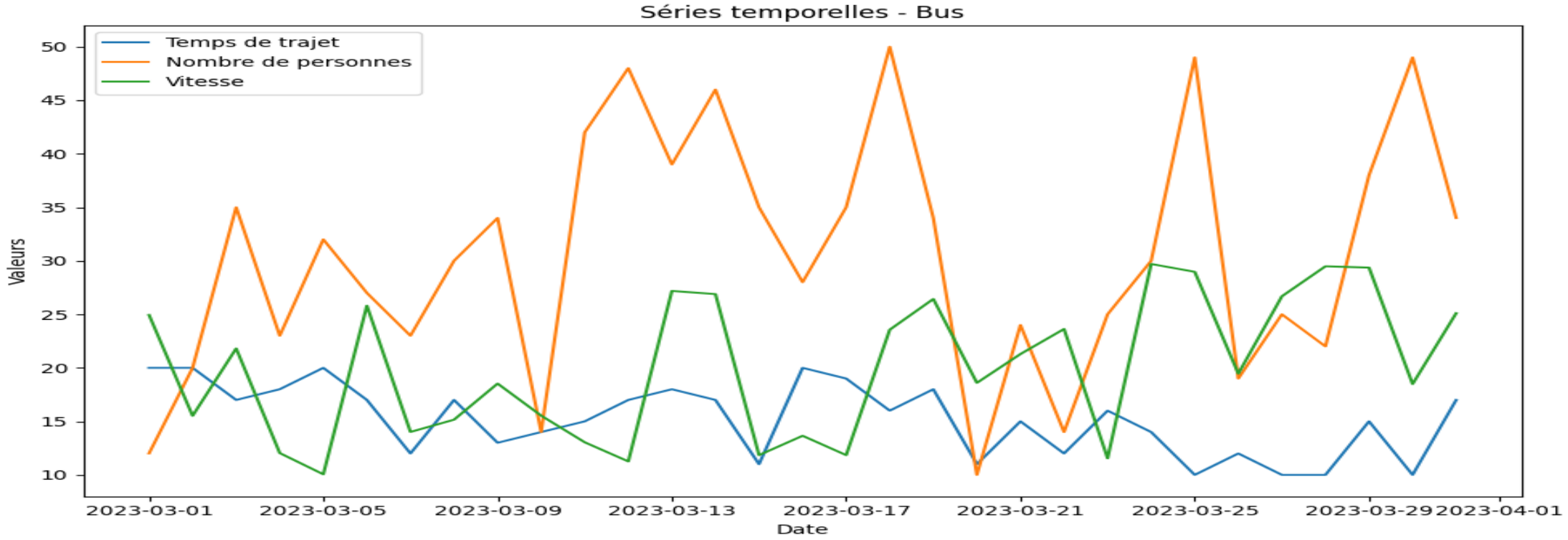


Figure 5.2 : Série temporelle-Bus

PRÉDICTION

- Il existe deux types de prédiction:

➢ **Prédiction à court terme:** Elle se concentre sur des événements ou des résultats qui se produiront dans un futur proche et utilise des prototypes différents de ceux qui sont utilisées pour l'apprentissage du réseau.

➢ **Prédiction à long terme:** Elle vise à estimer des événements ou des résultats qui se produiront dans un avenir plus éloigné et le réseau est itéré en boucle.

SIMULATION ET RÉSULTATS

- **Cas $\mu = 2$ (non chaotique) :**

Pas de l'apprent-issage	Momen-tum	Paramét-re de régulari-sation	Nombre de l'époque maximale
0.2	0.5	0.01	100

$$\text{MSE} = 0.008012$$

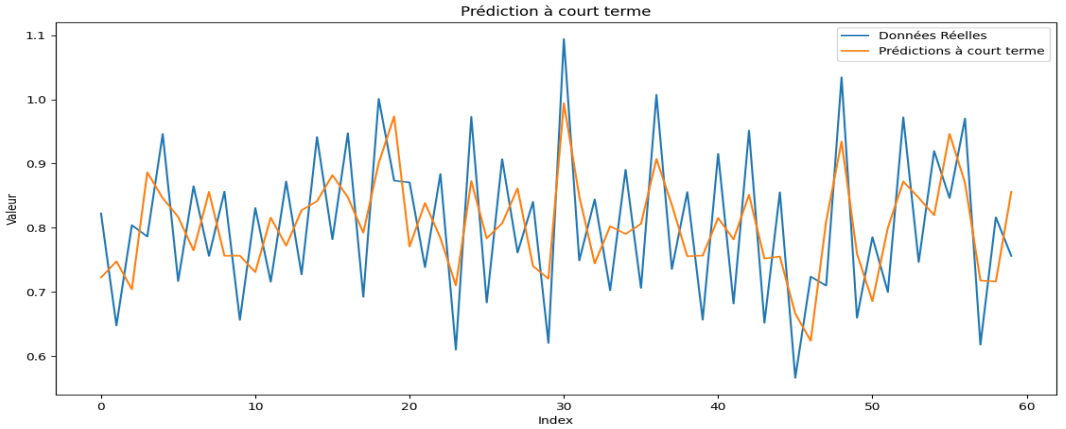


Figure 5.4: Prédiction à court terme pour $\mu = 2$

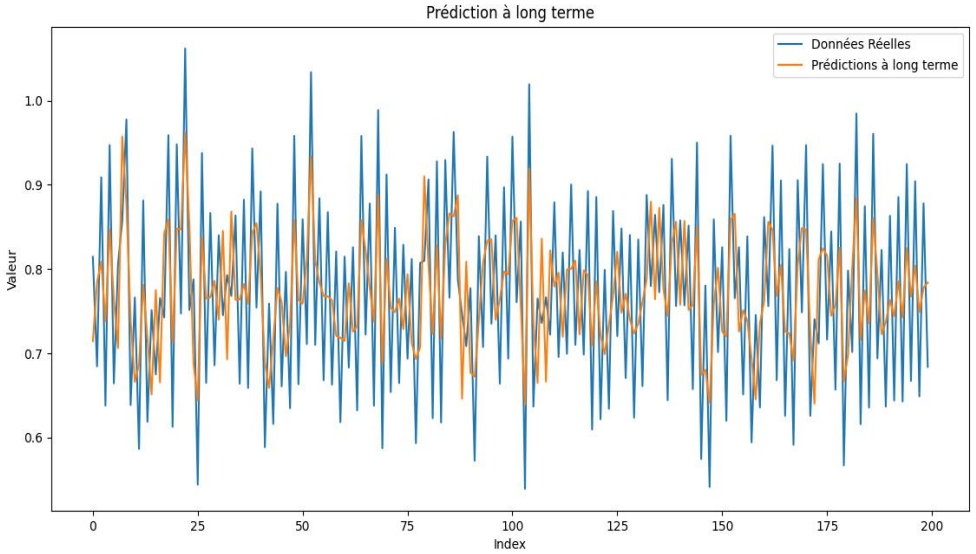


Figure 5.3: Prédiction à long terme pour $\mu = 2$

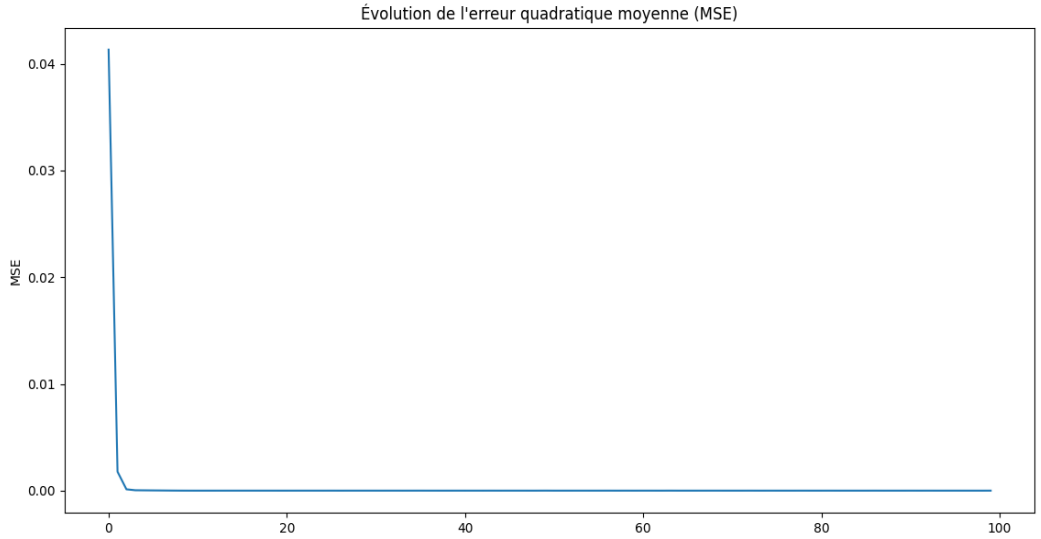


Figure 5.5: Evolution de l'erreur quadratique moyenne pour $\mu = 2$

SIMULATION ET RÉSULTATS

- **Cas $\mu = 4$ (chaotique) :**

Pas de l'apprent-issage	Momen-tum	Paramét-re de régulari-sation	Nombre de l'époque maximale
0.2	0.5	0.01	100

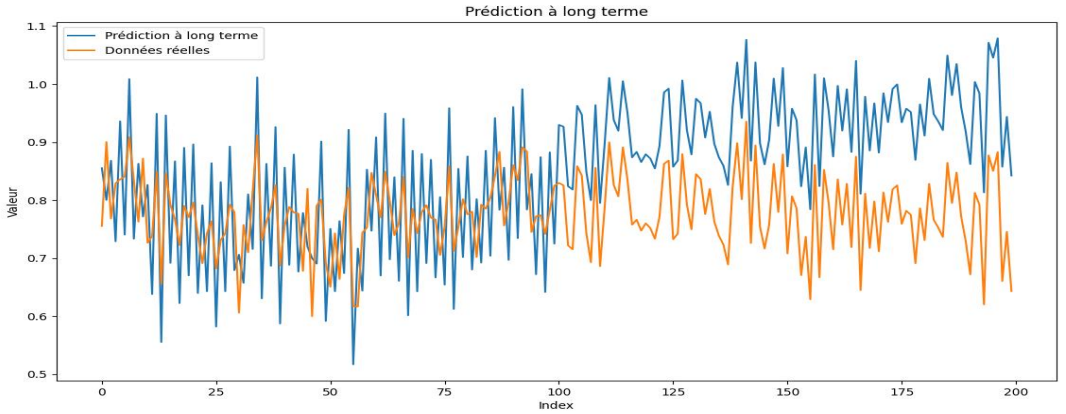


Figure 5.3: Prédiction à long terme pour $\mu = 4$

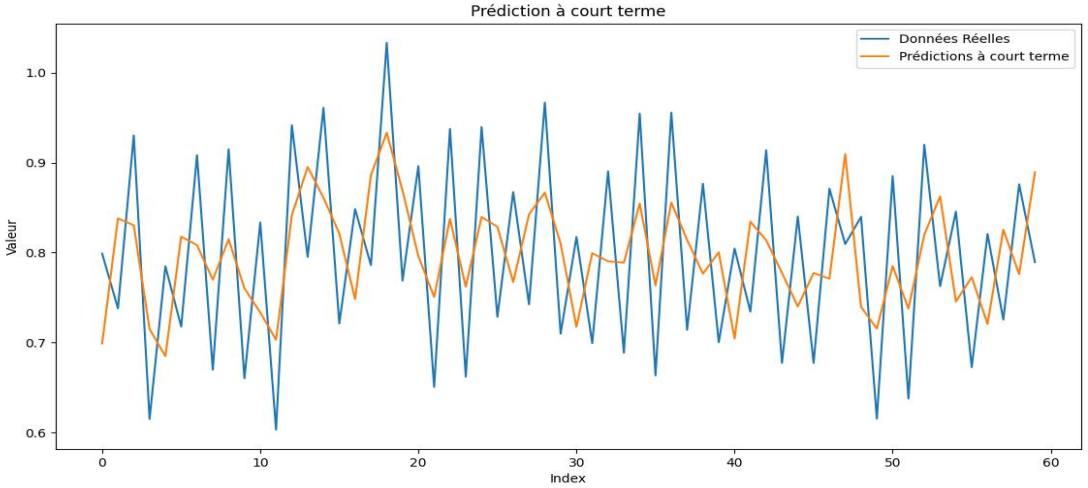


Figure 5.3: Prédiction à court terme pour $\mu = 4$

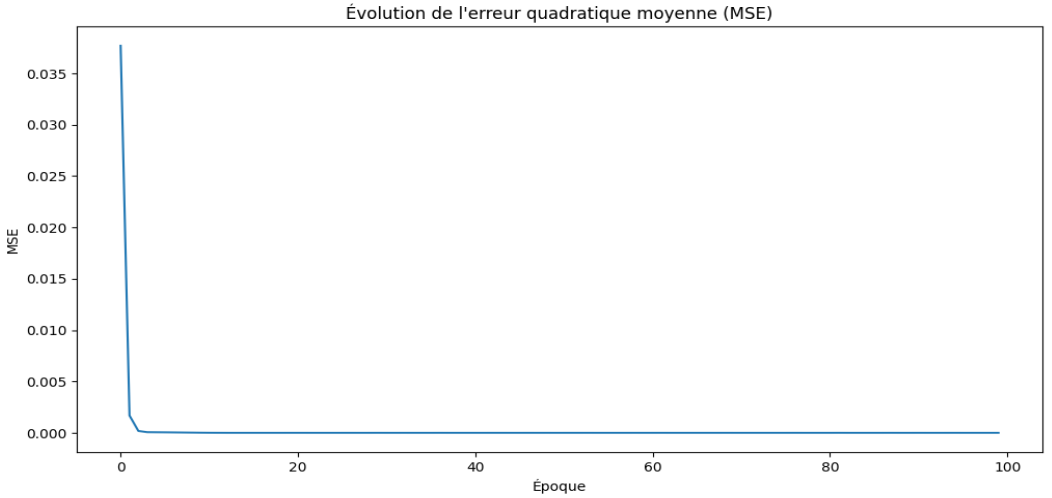


Figure 5.3: Evolution de l'erreur quadratique moyenne pour $\mu = 4$

CONCLUSION

- Les réseaux neurones artificielles sont un outil de prédiction puissant.

- Le comportement de l'application logistique dépend de μ . En variant ce paramètre, les séries temporelles peuvent être chaotiques ou non. La bifurcation met en évidence les points critiques et les différents comportements du système. La stabilité de l'application est étudiée à partir de l'exposant de Lypanouv.

- L'appréhension du problème des séries temporelles est possible même dans le cas chaotique .En utilisant la prédiction à court terme, les valeurs estimées par le perceptron sont très proches de la réalité. Par contre, la prédiction à long terme, l'erreur de la prédiction augmente avec le nombre d'itérations dans le cas chaotique.

ANNEXE

- **<u>Preuve de l'application aux cas de la résolution d'un système linéaire:</u>**

D'après le théoreme spectrale, il existe une base orthonormé $(X_1, \dots, X_n)$ de diagonolisation associée aux valeurs propres strictement positives $(\lambda_1, \dots, \lambda_n)$.

Soit $X \in \mathbb{R}^n$, d'où $\exists(\alpha_1, b_1, \dots, \alpha_n, b_n) \in \mathbb{R}^{2n}, X = \sum_{i=1}^{n} \alpha_i X_i$, $B = \sum_{i=1}^{n} b_i X_i$

donc $f(X) = \sum_{i=1}^{n} (\frac{1}{2}\lambda_i \alpha_i^2 - b_i \alpha_i)$

$\forall i \in [\![1, n]\!], \frac{1}{2}\lambda_i \alpha_i^2 - b_i \alpha_i \geq -\frac{b_i^2}{2\lambda_i}$ avec égalité ssi $\alpha_i = \frac{b_i}{\lambda_i}$ d'où $X_o = \sum_{i=1}^{n} \frac{b_i}{\lambda_i} X_i = A^{-1}B$

ANNEXE

- **Preuve du chois de la direction de la descendante:**

En effet, $< d_k, \nabla f(x_k) >=< -\nabla f(x_k), \nabla f(x_k) >= -\| \nabla f(x_k) \|^2 < 0$

Si par l'absurde $\min_{t\in[0,b]} g_k(t) \geq f(t)$. Alors, $\forall t \in [0,b]\ g_k(t) = f(x_k + td_k) \geq f(x_k)$

Ce qui implique, $< d_k, \nabla f(x_k) >= \lim_{t\to 0, t>0} \frac{f(x_k + td_k) - f(x_k)}{t} \Rightarrow < d_k, \nabla f(x_k) > \geq 0$

≥ 0

ANNEXE

- **Démonstration de la proposition 1:**

Soit $(x_k)_k$ une suite stationnaire pour f . Comme f supposé convexe, on a Alors :

$\forall x \in \mathbb{R}^n, f(x) \geq f(x_k) < \nabla f(x_k), x - x_k >$. Donc :

$\inf_{x\in\mathbb{R}^n} f \leq \sup_{K\in\mathbb{N}} \inf_{k\geq K} f(x_k) = \lim_{k\to+\infty} f(x_k) \leq \limsup_{k\to+\infty} f(x_k)$

$\leq \limsup_{k\to+\infty} f(x_k) +$

$\lim_{k\to+\infty} < \nabla f(x_k), x - x_k >$

$\leq \limsup_{k\to+\infty} (f(x_k) +$

$< \nabla f(x_k), x - x_k >) \leq f(x)$

D'où $\inf_{\mathbb{R}^n} f \leq \lim_{k\to+\infty} \inf f(x_k) \leq \lim_{k\to+\infty} \sup f(x_k) \leq \inf_{\mathbb{R}^n} f$

Alors : $\lim_{k\to+\infty} \inf f(x_k) = \lim_{k\to+\infty} \sup f(x_k) = \inf_{\mathbb{R}^n} f$. Finalement $\lim_{k\to+\infty} f(x_k) = \inf_{\mathbb{R}^n} f$

ANNEXE

- **Démonstration de la proposition 2:**

Commençons par montrer l'existence d'une sous-suite convergente .$\forall k \in \mathbb{N}, g_k(t) \leq f(x_k)$.

Donc $f(x_{k+1}) < f(x_k)$. D'où la suite $(f(x_k))_k$ est décroissante, d'où $f(x_k) \leq f(x_0)$, donc $(x_k)_k$ est contenu dans le sous-niveau $s_{f(x_0)} = \{x \in \mathbb{R}^n, f(x) \leq f(x_0)\}$.

Par la coerisivité de f, $s_{f(x_0)}$ est un compact . Alors la suite $(x_k)_k$ est bornée. Donc l'existence d'une sous-suite convergente d'après le théorème de Bolzano-Weistrass dans un espace vectoriel de dimension finie.

ANNEXE

Montrons les valeurs d'adhérence de $(x_k)_k$ sont des points stationnaires

$$\bar{x} = \lim_{k \to +\infty} x_{\varphi(n)}$$

D'après ce qui précède

$$\forall t \in]0, b] \text{ et } \forall n \in \mathbb{N}, f\left(x_{\varphi(n+1)}\right) \leq f\left(x_{\varphi(n)+1}\right) \leq f\left(x_{\varphi(n)} - t\nabla f\left(x_{\varphi(n)}\right)\right)$$

Par la continuité dérivabilité de f, $\forall t \in]0, b]$,

$$\frac{f\left(\bar{x} - t\nabla f(\bar{x})\right) - f(\bar{x})}{t} = \lim_{k \to +\infty} \frac{f\left(x_{\varphi(n)} - t\nabla f\left(x_{\varphi(n)}\right)\right) - f\left(x_{\varphi(n)}\right)}{t} \leq 0$$

D'où $0 \geq -\| \nabla f(\bar{x}) \|^2 = < \nabla f(\bar{x}), -\nabla f(\bar{x}) > = \lim_{t \to 0} \frac{f\left(\bar{x} - t\nabla f(\bar{x})\right) - f(\bar{x})}{t} \geq 0$ .Donc : $\| \nabla f(\bar{x}) \| = 0$

ANNEXE

- **Preuve du théorème:**

1. D'après la proposition 1 et 2.

2. D'après le point du théorème 1, Soit $(x_{\varphi(n)})_n$ une de ces suites de limite $\bar{x}$.

Par la continuité de f et la minimaliste de $(x_{\varphi(n)})_n$: $f(\bar{x}) = \inf_{x\in\mathbb{R}^n} f$. On en déduit que $\lim_{k\to+\infty} f(x_{\varphi(n)}) = f(\bar{x}) = \inf_{x\in\mathbb{R}^n} f$. Puisque f strictement convexe, $Argmin_{\mathbb{R}^n} f = \{\bar{x}\}$. Donc toute suite $(x_n)_n$ converge vers $\bar{x}$. Par la compacité de $s_{f(x_0)}$, $(x_n)_n$ converge vers $\bar{x}$.

ANNEXE

- **Montrons tout d'abord que: $< R_{k+1}, R_k >= 0 \ , \forall k \in \mathbb{N}$**

$$< R_{k+1}, R_k >=<AX_{k+1} - B, R_k >=< AX_k - B - \alpha_{k+1}AR_k >$$
$$=<AX_k - B, R_k > -\alpha_{k+1} < AR_k, R_k >$$
$$=<R_k, R_k > -\alpha_{k+1} < AR_k, R_k >=\| R_k \|^2 - \frac{\|R_k\|^2}{\|R_k\|_A{}^2} \| R_k \|_A = 0$$

- **Montrons la convergence de la suite X_n vers $\bar{X}$:**

➢ Commençons par montrer que $\| X_{n+1} - \bar{X} \|_A{}^2 = -\alpha_{n+1} \| R_{n+1} \|^2 + \| X_n - \bar{X} \|_A{}^2$:

$$\| X_{n+1} - \bar{X} \|_A{}^2 =<A(X_{n+1}-\bar{X}), (X_{n+1}-\bar{X}) >= <A(X_{n+1}-\bar{X}), X_{n+1} - X_n + X_n - \bar{X}>$$
$$=< A(X_{n+1}-\bar{X}), X_{n+1} - X_n>+< A(X_{n+1}-\bar{X}), X_n - \bar{X}>$$
$$=< A(X_{n+1}-\bar{X}), -\alpha_{n+1}R_n >+< A(X_{n+1}-\bar{X}), X_n - \bar{X}>$$
$$=< AX_{n+1} - B, -\alpha_{n+1}R_n >+< A(X_{n+1}-\bar{X}), X_n - \bar{X}>$$
$$= -\alpha_{n+1}< R_{n+1}, R_n >+< A(X_{n+1}-\bar{X}), X_n - \bar{X} >=< A(X_{n+1}-\bar{X}), X_n - \bar{X} >$$
$$=< A(X_n-\alpha_{n+1}R_n - \bar{X}), X_n - \bar{X} >$$
$$=< A(X_n-\bar{X}), X_n - \bar{X} > -\alpha_{n+1}< AR_n, X_n - \bar{X} >$$
$$= \| X_n - \bar{X} \|_A{}^2 - \alpha_{n+1} < R_n, A(X_n - \bar{X}) >=\| X_n - \bar{X} \|_A{}^2 - \alpha_{n+1} < R_n, A(X_n - \bar{X}) >$$
$$= \| X_n - \bar{X} \|_A{}^2 - \alpha_{n+1} < R_n, AX_n - B >= -\alpha_{n+1} \| R_{n+1} \|^2 + \| X_n - \bar{X} \|_A{}^2$$

ANNEXE

➢ **Montrons que** $\| X_n - \bar{X} \|_A \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^n \| X_0 - \bar{X} \|_A$:

$$\| X_n - \bar{X} \|_A^{\,2} = <A(X_n - \bar{X}), (X_n - \bar{X})> = <A(X_n - \bar{X}), A^{-1}A(X_n - \bar{X})>$$

$$= <AX_n - B, A^{-1}(AX_{n+1} - B)> = <R_n, A^{-1}R_n> = \| R_n \|_{A^{-1}}$$

Alors , $\| X_{n+1} - \bar{X} \|_A^{\,2} = -\alpha_{n+1} \| R_{n+1} \|^2 + \| R_n \|_{A^{-1}}^{\,2} = -\alpha_{n+1} \| R_{n+1} \|^2 + \| R_n \|_{A^{-1}}^{\,2}$

$$= -\frac{\|R_{n+1}\|^4}{\|R_n\|_A^{\,2}} + \| R_n \|_{A^{-1}}^{\,2} = \| R_n \|_{A^{-1}}^{\,2} \left(1 - \frac{\|R_{n+1}\|^4}{\|R_n\|_A^{\,2}\|R_n\|_{A^{-1}}^{\,2}}\right)$$

$$= \| X_n - \bar{X} \|_{A^{-1}}^{\,2} \left(1 - \frac{\|R_{n+1}\|^4}{\|R_n\|_A^{\,2}\|R_n\|_{A^{-1}}^{\,2}}\right) \leq \| X_n - \bar{X} \|_{A^{-1}}^{\,2} \left(1 - \frac{4\lambda_n\lambda_1}{(\lambda_n+\lambda_1)^2}\right)$$

$$\leq \| X_n - \bar{X} \|_{A^{-1}}^{\,2} \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 = \| X_n - \bar{X} \|_{A^{-1}}^{\,2} \left(\frac{M(A)-1}{M(A)+1}\right)^2$$

D'où on obtient, $\| X_n - X \|_A \leq \left(\frac{M(A)-1}{M(A)+1}\right)^n \| X_0 - \bar{X} \|_A$

Enfin, $\lambda_1 \| X \|^2 = \sum_i \lambda_1 x_i^2 \leq \sum_i \lambda_i x_i^2 = \| X \|_A^2 \leq \sum_i \lambda_n x_i^2 = \lambda_n \| X \|_A^2$

Donc , $\| X_n - X \| \leq \left(\frac{M(A)-1}{M(A)+1}\right)^n \sqrt{M(A)} \| X_0 - \bar{X} \|$

ANNEXE

Programme pour tracer le diagramme de birufication

```python
import numpy as np
import matplotlib.pyplot as plt
def logistic_map(x, μ):
    return μ * x * (1 - x)
def bifurcation_diagram(μ_vals, x_vals):
    bifurcation_points = []
    for μ in μ_vals:
        x = 0.5  # Condition initiale
        # Ignorer les premières itérations pour atteindre l'état stable
        for _ in range(500):
            x = logistic_map(x, μ)
        # Collecter les valeurs stables de x pour la valeur de r actuelle
        for _ in range(x_vals):
            x = logistic_map(x, μ)
            bifurcation_points.append([μ, x])
    bifurcation_points = np.array(bifurcation_points)
```

ANNEXE

```
17      # Tracer le diagramme de bifurcation
18      plt.scatter(bifurcation_points[:, 0], bifurcation_points[:, 1], s=1,
    c='black')
19      plt.xlabel('Paramètre μ')
20      plt.ylabel('Valeur de x')
21      plt.title('Diagramme de bifurcation')
22      plt.show()
23
24  # Paramètres du diagramme de bifurcation
25  μ_vals = np.linspace(2.9, 4.0, 1000)  # Intervalles et nombre de valeurs à
    ajuster selon vos besoins
26  x_vals = 100  # Nombre de valeurs stables de x à collecter pour chaque valeur de
    r
27  # Appel de la fonction pour tracer le diagramme de bifurcation
28  bifurcation_diagram(μ_vals, x_vals)
```

ANNEXE

- **Programme pour tracer le diagramme de birufication et exposant de Lypanouv:**

```python
import numpy as np
import matplotlib.pyplot as plt
def logistic_map(x, μ):
    return μ * x * (1 - x)
def lyapunov_exponent(x0, μ, n):
    x = x0
    sum_log = 0.0

    for i in range(n):
        derivative = μ - 2 * μ * x
        sum_log += np.log(np.abs(derivative))
        x = logistic_map(x, μ)

    return sum_log / n
# Paramètres du système
x0 = 0.2  # Condition initiale
μ_values = np.linspace(2.5, 4.0, 5000)  # Valeurs de μ
n = 1000  # Nombre d'itérations
# Calcul des exposants de Lyapunov
exponents = []
for μ in μ_values:
    exponent = lyapunov_exponent(x0, μ, n)
    exponents.append(exponent)
# Tracé du diagramme de bifurcation
plt.plot(μ_values, exponents, c='orange')
plt.xlabel('μ')
plt.ylabel('Exposant de Lyapunov')
plt.title("Diagramme de bifurcation - Exposant de Lyapunov")
plt.show()
```

ANNEXE

- **Programme pour tracer le diagramme coweb:**

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 3.5*x*(1-x)
def cobweb(f, x0, N, a=0, b=1):
    t = np.linspace(a, b, N)
    z = np.linspace(0,f(x0), N)
    L=[x0 for j in range(N)]
    plt.plot(t, f(t), 'k')
    plt.plot(t, t, "k:")
    x, y = x0, f(x0)
    for _ in range(N):
        fy = f(y)
        plt.plot([x, y], [y, y], 'b', linewidth=1)
        plt.plot([y, y], [y, fy], 'b', linewidth=1)
        x, y = y, fy
    plt.plot(0,0,c='black',label='f(x)=3.5x(1-x)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Diagramme coweb: logistique μ=3.5 ')
    plt.plot(t, t, c='orange', label='y=x')
    plt.plot(L, z, c='b',label='Line Coweb')
    plt.legend()
    plt.grid(True)
    plt.show()
    plt.close()

cobweb(f, 0.1, 100, 0, 1)
```

ANNEXE

- **Programme pour tracer série temporelle pour l'application logistique:**

```
33  import numpy as np
34  import matplotlib.pyplot as plt
35  def logistic_map(x0, μ, n):
36      series = [x0]
37      for _ in range(n-1):
38          x = μ * series[-1] * (1 - series[-1])
39          series.append(x)
40      return series
41  n = 500
42  x0 = 0.1
43  μ=3.5
44  plt.figure(figsize=(12, 6))
45  series = logistic_map(x0, μ, n)
46  plt.plot(series)
47  plt.xlabel('Time')
48  plt.ylabel('Value')
49  plt.title(f'Série temporelle : logistique μ=3.5')
50  plt.tight_layout()
51  plt.show()
```

ANNEXE

- **Programme pour tracer la frontière de classification:**

```python
import numpy as np
import matplotlib.pyplot as plt
# Données d'entrée et étiquettes de classe
x = np.array([[10, 15], [20, 25], [15, 10], [30, 20], [25, 25], [20, 10]])
y = np.array([0, 0, 0, 1, 1, 1])
# Ajouter une colonne de 1 pour représenter le terme de biais
x = np.concatenate((np.ones((x.shape[0], 1)), x), axis=1)
# Initialisation des paramètres
theta = np.zeros(x.shape[1])
# Définition de la fonction sigmoïde
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
# Entraînement du modèle de classification (logistic regression)
learning_rate = 0.1
num_iterations = 1000
for _ in range(num_iterations):
    # Calcul de la prédiction
    z = np.dot(x, theta)
    y_pred = sigmoid(z)
    # Calcul du gradient et mise à jour des paramètres
    gradient = np.dot(x.T, (y_pred - y)) / len(y)
    theta -= learning_rate * gradient
# Création de la grille pour la visualisation de la frontière
x_min, x_max = x[:, 1].min() - 1, x[:, 1].max() + 1
y_min, y_max = x[:, 2].min() - 1, x[:, 2].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
grid_points = np.c_[np.ones(len(xx.ravel())), xx.ravel(), yy.ravel()]
# Prédiction des classes pour les points de la grille
Z = sigmoid(np.dot(grid_points, theta))
Z = Z.reshape(xx.shape)
# Tracé des données de chaque classe et de la frontière
plt.contourf(xx, yy, Z, alpha=0.5, cmap=plt.cm.Paired)
plt.scatter(x[:, 1], x[:, 2], c=y, cmap=plt.cm.Paired)
plt.xlabel('la vitesse de bus')
plt.ylabel('le temps pris par le bus pour parcourir le trajet')
plt.title('Problème de classification avec frontière')
plt.grid(True)
plt.show()
```

ANNEXE

- **Programme pour tracer série temporelle pour la prédiction à long terme:**

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
# Génération de données de série temporelle à partir de l'application
logistique
def generate_time_series(num_points, logistic_param):
    X = np.arange(0, num_points)
    y = 10 / (1 + np.exp(-logistic_param * (X - 50))) + []
# Les Données réelles
    return X, y
# Division des données en ensembles d'entraînement et de test
def split_train_test(data, train_ratio):
    train_size = int(len(data) * train_ratio)
    train_data = data[:train_size]
    test_data = data[train_size:]
    return train_data, test_data
# Prétraitement des données
def preprocess_data(data):
    data_min = np.min(data)
    data_max = np.max(data)
    scaled_data = (data - data_min) / (data_max - data_min)
    return scaled_data, data_min, data_max
# Création du modèle LSTM
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(32, input_shape=input_shape))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
# Entraînement du modèle
def train_model(model, X_train, y_train, epochs):
    model.fit(X_train, y_train, epochs=epochs, batch_size=1, verbose=2)
# Prédiction à partir du modèle entraîné
def predict(model, X):
```

ANNEXE

```
34  def predict(model, X):
35      return model.predict(X)
36  # Paramètres du modèle
37  num_points = 1000  # Nombre de points de données à générer
38  train_ratio = 0.8  # Ratio des données d'entraînement
39  logistic_param = 4  # Paramètre logistique de l'application logistique
40  # Génération des données de série temporelle à partir de l'application
    logistique
41  X, y = generate_time_series(num_points, logistic_param)
42  # Division des données en ensembles d'entraînement et de test
43  train_data, test_data = split_train_test(y, train_ratio)
44  # Prétraitement des données
45  X_train, data_min, data_max = preprocess_data(train_data)
46  X_test = (test_data - data_min) / (data_max - data_min)
47  # Vérification de la forme des données d'entraînement
48  if len(X_train.shape) == 1:
49      X_train = np.reshape(X_train, (X_train.shape[0], 1))
50      X_test = np.reshape(X_test, (X_test.shape[0], 1))
51  # Création et entraînement du modèle LSTM à long terme
52  model_long_term = create_lstm_model((X_train.shape[1], 1))
53  train_model(model_long_term, X_train, X_train, epochs=100)
54  # Prédiction à long terme
55  y_pred_long_term = predict(model_long_term, X_test)
56  # Tracé du graphe de prédiction à long terme
57  plt.figure(figsize=(12, 6))
58  plt.plot(test_data, label='Données réelles')
59  plt.plot(y_pred_long_term, label='Prédictions à long terme')
60  plt.legend()
61  plt.xlabel('Index')
62  plt.ylabel('Valeur')
63  plt.title('Prédiction à long terme')
64  plt.show()
```

ANNEXE

- **Programme pour tracer série temporelle pour la prédiction à court terme:**

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
# Génération de données de série temporelle à partir de l'application logistique
def generate_time_series(num_points, logistic_param):
    X = np.arange(0, num_points)
    y = 10 / (1 + np.exp(-logistic_param * (X - 50))) + []# Données réelles
    return X, y
# Division des données en ensembles d'entraînement et de test
def split_train_test(data, train_ratio):
    train_size = int(len(data) * train_ratio)
    train_data = data[:train_size]
    test_data = data[train_size:]
    return train_data, test_data
# Prétraitement des données
def preprocess_data(data):
    data_min = np.min(data)
    data_max = np.max(data)
    scaled_data = (data - data_min) / (data_max - data_min)
    return scaled_data, data_min, data_max
# Création du modèle LSTM
def create_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(32, input_shape=input_shape))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
# Entraînement du modèle
def train_model(model, X_train, y_train, epochs):
    model.fit(X_train, y_train, epochs=epochs, batch_size=1, verbose=2)
# Prédiction à partir du modèle entraîné
def predict(model, X):
    return model.predict(X)
```

ANNEXE

```
34      return model.predict(X)
35  # Paramètres du modèle
36  num_points = 300  # Nombre de points de données à générer
37  train_ratio = 0.8  # Ratio des données d'entraînement
38  logistic_param = 2  # Paramètre logistique de l'application logistique
39  # Génération des données de série temporelle à partir de l'application
    logistique
40  X, y = generate_time_series(num_points, logistic_param)
41  # Division des données en ensembles d'entraînement et de test
42  train_data, test_data = split_train_test(y, train_ratio)
43  # Prétraitement des données
44  X_train, data_min, data_max = preprocess_data(train_data)
45  X_test = (test_data - data_min) / (data_max - data_min)
46  # Remodelage des données pour l'entrée du LSTM (3D array)
47  X_train = np.reshape(X_train, (X_train.shape[0], 1, 1))
48  X_test = np.reshape(X_test, (X_test.shape[0], 1, 1))
49  # Création et entraînement du modèle LSTM à court terme
50  model_short_term = create_lstm_model((1, X_train.shape[2]))
51  train_model(model_short_term, X_train, X_train, epochs=100)
52  # Prédiction à court terme
53  y_pred_short_term = predict(model_short_term, X_test)
54  # Tracé du graphe de prédiction à court terme
55  plt.figure(figsize=(12, 6))
56  plt.plot(test_data, label='Données réelles')
57  plt.plot(y_pred_short_term, label='Prédictions à court terme')
58  plt.legend()
59  plt.xlabel('Index')
60  plt.ylabel('Valeur')
61  plt.title('Prédiction à court terme')
62  plt.show()
```

MERCI POUR VOTRE ATTENTION