République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaïa

Faculté des Sciences Exactes

Département de Mathématiques

Mémoire présenté pour l'obtention du diplôme de Master en Mathématiques

Option : Mathématique de l'intelligence artificielle

Thème

Utilisation de l'IA pour l'optimisation des paramètres de régularisation dans les problèmes inverses liés aux équations de la chaleur

Présenté Par :

M^{lle} BENIDJER Macelia

Soutenu le : **29 Juin 2025**, devant le Jury composé de :

Nom	Fonction	Grade	Établissement
M^{r} KHELOUFI.A	Président	Professeur	Université de Bejaia
M^{me} DJENNADI.S	Promotrice	M.C.B	Université de Bejaia
M^{me} BECHIR.H	Examinatrice	Professeur	Université de Bejaia

Année universitaire : 2024 / 2025

Remerciements

Je tiens à exprimer ma profonde gratitude à ma promotrice, Madame DJENNADI SMINA, pour sa précieuse orientation, sa disponibilité constante, ses conseils avisés et son soutien tout au long de ce travail. Sa rigueur scientifique et son accompagnement bienveillant ont été essentiels à la réalisation de ce mémoire.

Je remercie également l'ensemble de mes enseignants et professeurs, qui m'ont transmis, au fil des années, un savoir solide et une passion pour la recherche scientifique. Leurs enseignements et leur encadrement ont grandement contribué à mon développement académique.

Enfin, je voudrais adresser mes remerciements les plus sincères à ma famille, pour son soutien moral indéfectible, sa patience et son encouragement permanent. Leur présence bienveillante m'a donné la force et la motivation nécessaires pour mener ce travail à terme.

Dédicace

Je dédie ce travail :

À mes parents, pour leur amour inconditionnel, leur soutien permanent et les innombrables sacrifices qu'ils ont consentis pour moi.

À mon fiancé, pour sa patience, son soutien fidèle, et pour avoir cru en moi avec douceur et constance.

À ma chère famille, pour leur encouragement et leur présence bienveillante tout au long de ce parcours.

À ma promotrice, pour sa disponibilité, ses conseils éclairés et son accompagnement attentif et rigoureux.

À tous mes enseignants, qui ont contribué à forger mon esprit et m'ont transmis la passion du savoir.

À toutes les personnes qui m'aiment, en témoignage de mon affection, de ma reconnaissance et de mon profond respect.

B.Macelia

Résumé

Les problèmes inverses jouent un rôle fondamental dans de nombreux domaines scientifiques et techniques, où il s'agit de reconstituer une information inconnue à partir de données observées indirectement. Toutefois, ces problèmes sont fréquemment mal posés au sens d'Hadamard, ce qui rend leur résolution instable et sensible au bruit. Pour surmonter cette difficulté, des techniques de régularisation sont couramment employées, parmi lesquelles la méthode de Tikhonov occupe une place centrale. Le choix du paramètre de régularisation constitue un enjeu crucial afin de garantir une solution stable et fidèle aux données.

Ce travail explore et compare deux approches pour la sélection optimale du paramètre de régularisation : d'une part, une approche classique basée sur le principe de Morozov ; d'autre part, une méthode moderne s'appuyant sur l'intelligence artificielle, via des réseaux de neurones conçus pour prédire automatiquement ce paramètre. Des simulations numériques montrent l'efficacité de ces méthodes dans le cadre d'un problème inverse associé à l'équation de la chaleur.

Abstract

Inverse problems play a fundamental role in many scientific and engineering fields, where the goal is to reconstruct unknown information from indirectly observed data. However, these problems are often ill-posed in the sense of Hadamard, which makes their solution unstable and sensitive to noise. To address this challenge, regularization techniques are employed, among which Tikhonov regularization holds a central position. Selecting an appropriate regularization parameter is crucial to ensure a stable and accurate solution.

This work aims to explore and compare two different methods for determining this parameter : a classical approach based on Morozov's discrepancy principle, and a modern approach depending on artificial intelligence, particularly neural networks. Numerical simulations illustrate the effectiveness of these methods in a representative context of an ill-posed inverse problem for diffusion equation.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

IA	Intelligence Artificielle
RN	Réseau de Neurones
EDP	Équation aux Dérivées Partielles
RMSE	Root Mean Square Error (Erreur quadratique moyenne)
FFT	Fast Fourier Transform
SVD	Singular Value Decomposition
GPU	Graphics Processing Unit
MLP	Multi-Layer Perceptron (Perceptron multicouche)
CNN	Convolutional Neural Network
CT	Tomodensitométrie
PET	Tomographie par émission de positons
IRM	Imagerie par résonance magnétique

Introduction

Les équations différentielles occupent une place centrale dans la modélisation des phénomènes physiques, biologiques, chimiques ou économiques. Elles permettent de décrire l'évolution de grandeurs dans le temps et/ou dans l'espace à partir de lois fondamentales formulées sous forme d'expressions mathématiques.

Le problème direct associé à ces équations consiste à déterminer l'état futur d'un système à partir de conditions initiales et de conditions aux limites connues. Cependant, dans de nombreuses situations concrètes, ces données ne sont pas directement accessibles. On est alors confronté à un problème inverse, qui vise à retrouver les causes (comme les conditions initiales ou des paramètres inconnus) à partir d'observations partielles ou bruitées. De tels problèmes apparaissent dans des domaines variés, tels que la géophysique, l'imagerie médicale, ou le traitement du signal.

Le problème inverse associé à l'équation de la chaleur modélise la diffusion thermique dans un milieu. Dans certaines applications industrielles ou scientifiques, il s'agit, par exemple, de reconstituer une distribution initiale de température à partir de mesures effectuées à un instant final ou sur la frontière du domaine. De telles situations illustrent un défi fondamental : la nature mal posée de nombreux problèmes inverses, caractérisée par une sensibilité excessive aux données et une instabilité des solutions.

Pour pallier ces difficultés, des techniques de régularisation sont introduites. Elles consistent à reformuler le problème de manière à en assurer la stabilité. Parmi elles, la méthode de Tikhonov est l'une des plus répandues : elle ajoute un terme de pénalisation

dans la formulation du problème, équilibrant la fidélité aux données et la régularité de la solution. Le choix du paramètre de régularisation, qui détermine ce compromis, est un enjeu central de la méthode.

En effet, selon la définition d'Hadamard, un problème est bien posé s'il admet une solution unique dépendant continûment des données. Or, les problèmes inverses violent souvent au moins une de ces conditions : la solution peut être inexistante, non unique, ou extrêmement sensible aux perturbations des données. Cette instabilité se manifeste de manière aiguë lorsqu'on travaille avec des observations bruitées, ce qui rend la résolution numérique particulièrement délicate.

Ce mémoire se propose d'explorer et de comparer deux approches pour la sélection optimale de ce paramètre : une méthode classique, fondée sur la règle de Morozov, et une approche moderne exploitant les outils de l'intelligence artificielle, en particulier les réseaux de neurones, pour prédire automatiquement un paramètre de régularisation adapté. Ces méthodes sont étudiées dans un cadre simulé basé sur l'équation de la chaleur, avec des observations bruitées, permettant une évaluation numérique.

Objectif du mémoire. :

Ce travail vise à mettre en évidence l'intérêt de combiner les approches analytiques traditionnelles avec les techniques modernes d'apprentissage automatique pour résoudre des problèmes inverses mal posés. Nous chercherons à comprendre dans quelles conditions un réseau de neurones peut surpasser une méthode classique en termes de qualité de reconstruction, robustesse au bruit et adaptabilité aux données.

CHAPITRE 1

Théorie des problèmes directs, inverses et la méthode de régularisation

Dans ce chapitre, on présente quelques définitions et résultats fondamentaux sur les problèmes directs et inverses, utiles pour la compréhension des méthodes de régularisation abordées par la suite. On insiste en particulier sur les difficultés propres aux problèmes inverses et sur l'importance des outils mathématiques permettant de les traiter de manière stable.

1.1 Notions fondamentales d'analyse fonctionnelle

Dans le but de simplifier la lecture de ce travail, nous avons consacré cette partie à quelques notions de base et résultats d'analyse fonctionnelle. Voir [1]-[2] pour les démonstrations.

Définition 1.1.1 *Soit H un espace vectoriel sur $\mathbb{R}$. Une norme sur H est une application*

$$\|\cdot\|_H : H \to \mathbb{R}$$

qui vérifie les trois propriétés suivantes :

Séparation : $\forall x \in H,\ \|x\|_H > 0$ *et* $\|x\|_H = 0 \Rightarrow x = 0$

Homogénéité : $\forall x \in H,\ \forall \alpha \in \mathbb{R},\ \|\alpha x\|_H = |\alpha| \cdot \|x\|_H$

Inégalité triangulaire : $\forall x, y \in H,\ \|x + y\|_H \leq \|x\|_H + \|y\|_H$

Exemple 1.1.1 *Dans le cas où H est de dimension n (c'est-à-dire $H = \mathbb{R}^n$), les normes suivantes sont parmi les plus utilisées :*

Norme ℓ^1 (ou norme de Manhattan) :

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

Norme ℓ^2 (ou norme euclidienne) :

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$$

Norme ℓ^∞ (ou norme du maximum) :

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

Définition 1.1.2 *Un **espace préhilbertien** est un espace vectoriel E (réel ou complexe) muni d'un produit scalaire*

$$\langle \cdot, \cdot \rangle : E \times E \to \mathbb{K}$$

où $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$, satisfaisant les propriétés suivantes, pour tous $x, y, z \in E$ et tout $\lambda \in \mathbb{K}$:

— ***Linéarité (en le premier argument)*** *:*

$$\langle \lambda x + y, z \rangle = \lambda \langle x, z \rangle + \langle y, z \rangle$$

— ***Symétrie hermitienne*** *:*

$$\langle x, y \rangle = \overline{\langle y, x \rangle}$$

— ***Positivité définie*** *:*

$$\langle x, x \rangle \geq 0 \quad et \quad \langle x, x \rangle = 0 \iff x = 0$$

La norme induite par ce produit scalaire est donnée par :

$$\|x\| = \sqrt{\langle x, x\rangle}$$

Un ***espace de Hilbert*** *est un espace préhilbertien H qui est* ***complet*** *pour la norme induite par le produit scalaire, c'est-à-dire que toute suite de Cauchy dans H converge dans H.*

Définition 1.1.3 *On appelle* ***base hilbertienne*** *d'un espace de Hilbert H toute suite $(e_n)_{n\in\mathbb{N}^*}$ telle que :*

1. $\|e_n\|_H = 1$, *pour tout* $n \in \mathbb{N}^*$, *(normalisation)*
2. $(e_n, e_m) = 0$, *pour tout* $n \neq m$, *(orthogonalité)*

et telle que l'espace vectoriel engendré par cette suite soit dense dans H.

Autrement dit, pour tout $x \in H$, on a :

$$x = \sum_{n=1}^{\infty} (x, e_n) e_n,$$

la série étant convergente dans H.

Remarque 1.1.1 *(Unicité du développement) Tout vecteur $x \in H$ s'écrit de façon unique sous la forme :*

$$x = \sum_{n=1}^{\infty} x_n e_n, \quad \text{avec} \quad \sum_{n=1}^{\infty} |x_n|^2 < \infty,$$

et les coefficients sont donnés par :

$$x_n = (x, e_n).$$

Théorème 1.1.1 *(Égalité de Bessel-Parseval) Soit $(e_n)_{n\in\mathbb{N}^*}$ une base hilbertienne d'un espace de Hilbert H, et $x \in H$. Alors on a l'égalité suivante :*

$$\|x\|_H^2 = \sum_{n=1}^{\infty} |(x, e_n)|^2.$$

Exemple 1.1.2 *Considérons l'espace de Hilbert $L^2(0, 2\pi)$, muni du produit scalaire :*

$$(f, g) = \int_0^{2\pi} f(x)\, \overline{g(x)}\, dx.$$

La famille suivante constitue une base hilbertienne orthonormée de $L^2(0, 2\pi)$ *:*

$$\left\{ \frac{1}{\sqrt{2\pi}},\ \frac{1}{\sqrt{\pi}}\cos(nx),\ \frac{1}{\sqrt{\pi}}\sin(nx) \right\}_{n\in\mathbb{N}^*}.$$

Tout élément $f \in L^2(0, 2\pi)$ *admet un développement en série de Fourier :*

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx) \right),$$

avec les coefficients de Fourier donnés par :

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(nx)\,dx, \quad b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(nx)\,dx.$$

Cette décomposition est unique, et la série converge vers f *en norme* L^2*, conformément au théorème de Bessel-Parseval.*

Définition 1.1.4 *Soient* H *et* G *deux espaces de Hilbert.*

1. *Un opérateur linéaire* $A : H \to G$ *est dit continu si :*
   - $\forall u \in H,\ Au \in G,$
   - $\exists M > 0, \forall u \in H,\ \|Au\|_G \leq M\|u\|_H.$

   Le plus petit réel M *satisfaisant cette inégalité est appelé la norme de l'opérateur* A *:*
   $$\|A\| = \sup_{u\in H,\, u\neq 0} \frac{\|Au\|_G}{\|u\|_H}$$
2. *Toute application linéaire continue* $A : H \to G$ *est appelée opérateur. L'ensemble des opérateurs continus de* H *dans* G *est noté* $\mathcal{L}(H, G)$*.*
3. *Un opérateur linéaire* $A : H \to G$*, défini sur tout* H*, est continu s'il est continu en* $0 \in H$*.*

Théorème 1.1.2 *Un opérateur linéaire* $A : H \to G$*, tel que* $D(A) = H$*, est borné si et seulement si :*

$$\exists c > 0 \text{ tel que } \|Au\| \leq c\|u\|, \quad \forall u \in H.$$

Un opérateur linéaire $A : H \to G$*, défini sur tout* H*, est continu si et seulement s'il est borné.*

Définition 1.1.5 *(Opérateurs adjoints) Soit A un opérateur linéaire défini sur un domaine $D(A) \subset H$, dense dans H. On appelle adjoint de A un opérateur A^* défini sur $D(A^*) \subset G$ à valeurs dans H, tel que :*

$$\forall u \in D(A),\, \forall v \in D(A^*) : \quad (Au, v)_G = (u, A^* v)_H.$$

De plus, l'adjoint vérifie :

$$(A^*)^* = A \quad et \quad \|A^*\| = \|A\|.$$

Proposition 1.1.1 (Unicité de l'adjoint) *Si l'adjoint A^* existe, alors il est unique.*

Soient B et C deux opérateurs qui satisfont la relation :

$$\forall u \in D(A),\, \forall v \in D(B) \cap D(C) : \quad (Au, v)_G = (u, Bv)_H = (u, Cv)_H.$$

Cela implique que $(u, Bv - Cv)_H = 0$ pour tout $u \in D(A)$. Or, comme $D(A)$ est dense dans H, il en résulte que $Bv - Cv = 0$, donc $Bv = Cv$ pour tout $v \in D(B) \cap D(C)$. Ainsi, $B = C$ sur leur domaine commun, et par extension $B = C$. L'adjoint est donc unique.

Proposition 1.1.2 *Soient A et B deux opérateurs linéaires, et $\alpha, \beta \in \mathbb{R}$, alors :*

- $(\alpha A + \beta B)^* = \alpha A^* + \beta B^*$,
- $(AB)^* = B^* A^*$.

Définition 1.1.6 (Opérateur compact) *Soient H_1 et H_2 deux espaces de Hilbert. Un opérateur linéaire $A : H_1 \to H_2$, défini sur un sous-ensemble $D(A) \subset H_1$, est dit* ***compact*** *si, pour toute suite bornée $(f_n)_{n\in\mathbb{N}} \subset D(A)$, il existe une sous-suite (f_{n_k}) telle que la suite (Af_{n_k}) converge fortement dans H_2.*

Autrement dit, l'image d'un ensemble borné par l'opérateur A est relativement compacte dans H_2.

On note $\mathcal{L}(H_1, H_2)$ l'ensemble des opérateurs linéaires continus de H_1 dans H_2, et $\mathcal{K}(H_1, H_2) \subset \mathcal{L}(H_1, H_2)$ l'ensemble des opérateurs compacts.

Définition 1.1.7 *(Schauder) Soit $A \in \mathcal{L}(H, G)$, où G est un espace complet. L'opérateur A est compact si et seulement si son adjoint A^* est compact.*

Théorème 1.1.3 *(Riesz) L'opérateur identité I d'un espace vectoriel normé est compact si et seulement si cet espace est de dimension finie.*

Théorème 1.1.4 *Soit H un espace de Hilbert de dimension infinie, et $A : H \to H$ un opérateur compact. Alors, si A est inversible, son inverse A^{-1} n'est pas continu.*

Preuve : *Supposons par l'absurde que A^{-1} est continu. Alors l'identité $I = A \cdot A^{-1}$ est un opérateur compact. Cela impliquerait que la boule unité de H est compacte, ce qui est absurde sauf si H est de dimension finie (théorème de Riesz)*

Définition 1.1.8 *Soit A un opérateur linéaire défini sur tout H, tel que $A : H \to G$ avec $D(A) = H$. Un nombre complexe $\lambda \in \mathbb{C}$ est un* ***point régulier*** *de A si l'opérateur $(A - \lambda I)$ est inversible et borné.*

Définition 1.1.9 *L'ensemble des points réguliers est appelé* ***résolvante*** *de A :*

$$\rho(A) = \{\lambda \in \mathbb{C} \mid (A - \lambda I)^{-1} \text{ existe et est borné}\}.$$

Son complémentaire dans $\mathbb{C}$ est le ***spectre*** *de A, noté :*

$$\sigma(A) = \mathbb{C} \setminus \rho(A).$$

Définition 1.1.10 *Si $\lambda \in \rho(A)$, l'opérateur $R_\lambda(A) = (A - \lambda I)^{-1}$ est appelé la* ***résolvante*** *de A en λ.*

Définition 1.1.11 *(Valeur et vecteur propre) Un nombre $\lambda \in \mathbb{C}$ est une* ***valeur propre*** *de A s'il existe $u \in D(A) \setminus \{0\}$ tel que :*

$$Au = \lambda u.$$

Dans ce cas, u est un ***vecteur propre*** *associé à λ.*

Proposition 1.1.3 *Soit H un espace de Hilbert et A un opérateur linéaire borné, alors :*

$$\sigma(A^*) = \sigma(A).$$

1.2 Problèmes directs et inverses

La modélisation mathématique de phénomènes physiques, biologiques ou industriels conduit naturellement à l'étude de deux grandes catégories de problèmes : les problèmes directs et les problèmes inverses. Les problèmes directs consistent à prédire l'effet à partir d'une cause connue, en résolvant des équations modélisant le phénomène. À l'inverse, les problèmes inverses cherchent à déterminer les causes à partir de l'observation de leurs effets.

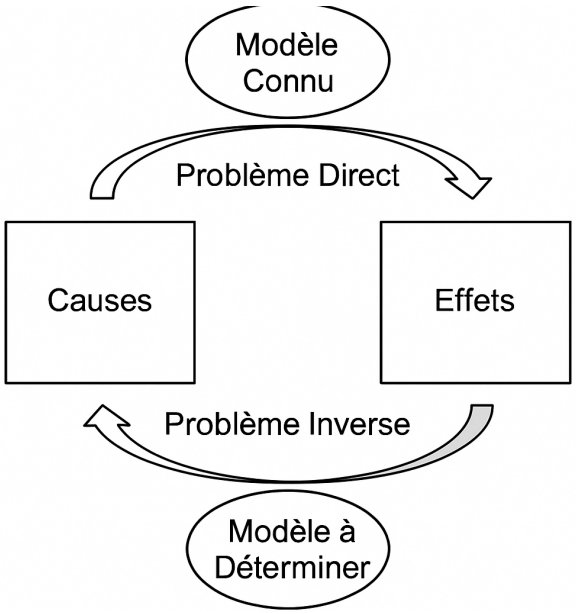


Figure 1.2.1 : Schéma du problème direct et inverse

Le problème inverse, par définition, présente plusieurs difficultés. Il est primordial que le problème direct associé soit bien posé, c'est-à-dire que « les mêmes causes produisent les mêmes effets ». En revanche, un même effet peut résulter de causes différentes, ce qui complique l'étude des problèmes inverses : il existe une forte possibilité d'avoir plusieurs solutions. Pour cette raison, il est nécessaire de s'appuyer sur des informations complémentaires afin de sélectionner la solution qui répondra adéquatement au problème posé.

Les problèmes inverses jouent un rôle important, comme en témoignent leurs nombreuses applications dans divers domaines scientifiques, tels que la mécanique quantique,

le radar, l'ingénierie pétrolière, la résolution de systèmes linéaires, ou encore l'imagerie médicale.

1.2.1 Définitions et principes fondamentaux

Considérons un système physique ou mathématique modélisé par un opérateur A, agissant sur un ensemble de données x pour produire une sortie y, selon la relation :

$$Ax = y \tag{1.1}$$

où :

1. *$A : X \to Y$ est un opérateur (intégral, différentiel, linéaire ou non linéaire),*
2. *$x \in X$ désigne les paramètres inconnus ou quantités d'entrée,*
3. *$y \in Y$ représente les données mesurées ou observées.*

Cette formulation englobe les problèmes directs et inverses.

Définition 1.2.1 *Le problème direct consiste à déterminer la sortie y à partir d'une entrée x connue, en appliquant un opérateur A.*

Le problème inverse, quant à lui, consiste à retrouver l'entrée x à partir de la connaissance de la sortie y.

Définition 1.2.2 *Un problème est dit bien posé s'il satisfait simultanément les trois critères suivants :*

1. ***Existence*** *: Le problème doit admettre au moins une solution pour les données considérées.*
2. ***Unicité*** *: Il ne doit exister qu'une seule solution pour les mêmes conditions initiales.*
3. ***Stabilité*** *(ou dépendance continue aux données) : La solution doit réagir de manière stable aux petites perturbations des données. Ce critère est crucial en calcul numérique.*

Si l'une de ces conditions n'est pas vérifiée, le problème est dit mal posé. Le fait que la solution d'un problème inverse puisse ne pas exister n'est pas une difficulté sérieuse. Il est habituellement possible de rétablir l'existence en relaxant la notion de solution. La non-unicité est un problème plus sérieux. S'il y a plusieurs solutions, il faut un moyen de choisir l'une d'entre elles (Disposer d'une information supplémentaire). L'absence de stabilité est sans doute la plus problématique, de petites perturbations sur les données peuvent engendrer de forts écarts sur la solution.

Exemple 1.2.1 *Considérons le système suivant $Ax = y$, où $A \in \mathbb{M}_{2,2}$, $x \in \mathbb{R}^2$, et $y \in \mathbb{R}^2$:*

$$A = \begin{pmatrix} 4{,}218613 & 6{,}327917 \\ 3{,}141592 & 4{,}712390 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad y = \begin{pmatrix} 10{,}54530 \\ 7{,}85982 \end{pmatrix}.$$

La solution exacte est :

$$x_1 = x_2 = 1.$$

Supposons maintenant que l'on introduise une petite perturbation dans la matrice A, et considérons la matrice perturbée suivante :

$$\tilde{A} = \begin{pmatrix} 4{,}218611 & 6{,}327917 \\ 3{,}141594 & 4{,}712390 \end{pmatrix}.$$

La solution du système perturbé $\tilde{A}\tilde{x} = y$ est alors :

$$\tilde{x}_1 = \tilde{x}_2 = 5.$$

*Autrement dit, de très petites variations dans les coefficients de la matrice A ont conduit à de **grandes variations dans la solution** x. Ce phénomène illustre clairement le **caractère instable** ou **mal posé** du problème inverse : une petite erreur dans les données peut engendrer une erreur importante dans la solution.*

Exemple 1.2.2 *Soit $A \in M_{n,n}(\mathbb{R})$ une matrice inversible et $y \in \mathbb{R}^n$ un vecteur colonne. On cherche à étudier l'influence des erreurs d'arrondi de la matrice A et du vecteur y sur la solution $x \in \mathbb{R}^n$ du système $Ax = y$.*

Le système linéaire $Ax = y$ est bien posé si

$$\begin{cases} A^{-1} \text{ existe} \\ \det A \neq 0 \\ Ax = 0 \text{ équivaut } x = 0 \end{cases}$$

Considérons le système suivant $Ax = y$

$$\begin{pmatrix} 23 & 9 & 12 \\ 12 & 10 & 1 \\ 14 & 12 & 25 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 44 \\ 23 \\ 27 \end{pmatrix}$$

qui admet comme solution

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Considérons le problème suivant dans lequel le vecteur y est légèrement perturbé

$$\begin{pmatrix} 23 & 9 & 12 \\ 12 & 10 & 1 \\ 14 & 12 & 25 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 44.44 \\ 22.77 \\ 26.73 \end{pmatrix}.$$

La solution du système perturbé est

$$\tilde{x} = \begin{pmatrix} 6.23 \\ 4.73 \\ 4.69 \end{pmatrix}.$$

On remarque que de petites variations sur y conduisent à de grandes variations sur x. Donc le problème est mal posé (condition de stabilité).

1.2.2 Applications en sciences et ingénierie

Pour mieux saisir l'importance des problèmes inverses dans la pratique, nous présentons quelques domaines d'application en sciences et en ingénierie.

1. ***Imagerie médicale***

- *Tomographie (CT, PET, IRM) : La reconstruction d'images internes à partir de mesures externes constitue un problème inverse typique. Dans ce contexte, les opérateurs linéaires, l'adjoint et la régularisation de Tikhonov sont utilisés pour reconstruire une image fiable à partir de données bruitées.*
- *Détection de tumeurs : Modèles pour localiser une tumeur dans des images bruitées, en utilisant des techniques d'imagerie inverse et de régularisation pour améliorer la précision.*

2. ***Géophysique et sismologie***
   - *Problème direct : La propagation des ondes sismiques à partir d'un épicentre connu est un problème direct, où l'on calcule l'effet des ondes sur des capteurs situés à la surface de la Terre.*
   - *Problème inverse : L'estimation des propriétés du sous-sol à partir des données mesurées par les ondes sismiques est un problème inverse où des techniques de régularisation et des espaces de Hilbert sont utilisés pour déterminer les caractéristiques géologiques.*
3. ***Thermique et transfert de chaleur***
   - *Systèmes linéaires : En mécanique et en électricité (par exemple, circuits RLC), les paramètres internes d'un système peuvent être estimés à partir des réponses mesurées, en utilisant des techniques de contrôle et d'identification des systèmes.*
4. ***Vision par ordinateur et traitement d'images***
   - *Reconstruction d'images floues ou bruitées (déconvolution, inpainting).*
   - *Interpolation de données manquantes via des modèles mathématiques inverses.*
   - *Application de l'analyse fonctionnelle : opérateurs, adjoints, régularisation.*
5. ***Modélisation des systèmes dynamiques***
   - *Systèmes linéaires : mécanique, électricité (circuits RLC).*
   - *Estimation des paramètres internes d'un système à partir des réponses mesurées.*
   - *Modèles de contrôle et identification.*

6. ***Applications industrielles***
   - *Détection de défauts dans les matériaux (NDT) :*
   - *Utilisation d'ondes ou de signaux pour diagnostiquer des défauts internes.*
   - *Optimisation de procédés industriels via des modèles inverses.*

1.3 Régularisation des problèmes inverses

Pour remédier à l'instabilité des problèmes inverses mal posés, on utilise des techniques de régularisation. Celles-ci consistent à modifier le problème initial de manière à le stabiliser, tout en conservant une approximation fidèle de la solution recherchée. Parmi les méthodes les plus répandues figure la régularisation de Tikhonov, qui introduit un terme de pénalisation afin de limiter la sensibilité de la solution aux perturbations affectant les données.

Dans notre travail, nous considérons la formulation suivante : soit la relation $Ax = y$ telle que $A : X \to Y$ un opérateur compact, linéaire et injectif entre deux espaces normés X (espace des données d'entrée) et Y (espace des observations). Le problème inverse associé consiste à déterminer l'inconnu $x \in X$ à partir de données mesurées, bruitées ou incomplètes $y \in Y$.

Remarque 1.3.1 *En pratique, cette inconnue peut représenter un paramètre du modèle, un terme source, une condition initiale ou une condition aux limites.*

Théorème 1.3.1 *Soit $A : X \to Y$ un opérateur compact entre deux espaces normés, avec $\dim(X) = \infty$. Alors, le problème linéaire $Ax = y$ est toujours mal posé au sens de Hadamard, c'est-à-dire qu'il ne satisfait pas en général la condition de stabilité.*

En général, les problèmes inverses sont résolus en minimisant l'erreur entre les données prédites et les données mesurées, ce qui conduit à rechercher une solution approchée de (1.1), c'est-à-dire à minimiser $\|Ax - y\|_Y$ par rapport à $x \in X$.

Lemme 1.1 *Soient X, Y des espaces de Hilbert tels que $A : X \to Y$ soit un opérateur linéaire et borné. Alors, il existe $\hat{x} \in X$ tel que*

$$\hat{x} = \arg\min_{x \in X} \|Ax - y\|_Y \tag{1.2}$$

si et seulement si $\hat{x}$ résout l'équation normale

$$A^*A\hat{x} = A^*y \tag{1.3}$$

où $A^* : Y \to X$ est l'opérateur adjoint de A.

D'après ce lemme, si $\dim X = \infty$ et que A est un opérateur compact, alors la minimisation ci-dessus est également un problème mal posé. Il faut donc modifier la fonctionnelle $\|Ax - y\|_Y$ de manière à ce que l'opérateur A^*A ne soit plus compact.

1.3.1 Principe de la regularisation

L'idée générale de la régularisation est de construire des solutions stables pour des problèmes inverses mal posés. Cela se fait en remplaçant le problème instable par une famille de problèmes bien posés qui dépendent d'un ou plusieurs paramètres de régularisation.

Définition 1.3.1 *La stratégie de régularisation est une famille d'opérateurs linéaires et bornés $R_\beta : Y \to X$, où $\beta > 0$ est appelé paramètre de régularisation, telle que*

$$\lim_{\beta \to 0} \|R_\beta Ax - x\|_X = 0, \text{ pour tout } x \in X$$

c'est-à-dire que les opérateurs $R_\beta A$ convergent ponctuellement vers l'identité.

Théorème 1.3.2 *Pour un opérateur linéaire compact $A : X \to Y$, et $\dim X = \infty$, la famille R_β n'est pas uniformément bornée, c'est-à-dire qu'il existe une suite $\{\beta_j\}$ telle que*

$$\lim_{j \to \infty} \beta_j = 0, \text{ avec } \|R_{\beta_j}\| \to \infty \text{ pour } j \to \infty.$$

i.e. R_β ne converge pas vers l'identité au sens de la norme d'opérateur.

La propriété de stabilité est la principale préoccupation lorsqu'on tente de résoudre (2.5), car, en pratique, y est souvent une quantité mesurée et est donc sujette à des erreurs d'observation. On suppose qu'une borne de l'erreur est donnée par :

$$\|y - y^\delta\|_Y \leq \delta \tag{1.4}$$

où y^δ est la donnée bruitée et δ représente le bruit. Ainsi, au lieu de résoudre (1.1), nous allons résoudre l'équation perturbée $Ax = y^\delta$. Soit $x_\beta^\delta = R_\beta y^\delta$ la solution régularisée de $Ax = y^\delta$. L'erreur fondamentale entre la solution exacte et la solution régularisée est donnée, en utilisant l'inégalité triangulaire sur $\|x_\beta, \delta - x\|$, par :

$$\begin{aligned}\|x_\beta^\delta - x\|_E &= \|R_\beta y^\delta - R_\beta y + R_\beta y - x\|_E \\ &\leq \|R_\beta y^\delta - R_\beta y\| + \|R_\beta y - x\| \\ &\leq \|R_\beta\| \cdot \|y^\delta - u\| + \|R_\beta Ax - x\|_E\end{aligned}$$

Ceci est notre estimation fondamentale.

$$\|x_\beta^\delta - x\|_X \leq \delta\|R_\beta\|_{\mathcal{L}(Y,X)} + \|R_\beta Ax - x\|_X \tag{1.5}$$

où $\delta\|R_\beta\|_{\mathcal{L}(Y,X)}$ représente l'erreur de propagation des données et $|R_\beta Ax - x\|_X$ est l'erreur d'approximation.

À ce stade, il devient clair que le choix de β est crucial, car l'estimation (1.4) montre que l'erreur se compose de deux termes :

***Le premier terme** dû aux erreurs sur les données, multiplié par un nombre de conditionnement, qui tend vers l'infini lorsque $\beta \to 0$ (i.e. $\|R_\beta\| \to \infty$ quand $\beta \to 0$).*
Il ne faut donc pas choisir β trop petit, sinon l'erreur peut devenir très grande.

***Le second terme** dû à l'approximation de la solution exacte, et qui tend vers zéro lorsque $\beta \to 0$ (i.e. $\|R_\beta Ax - x\|_X \to 0$ quand $\beta \to 0$).*

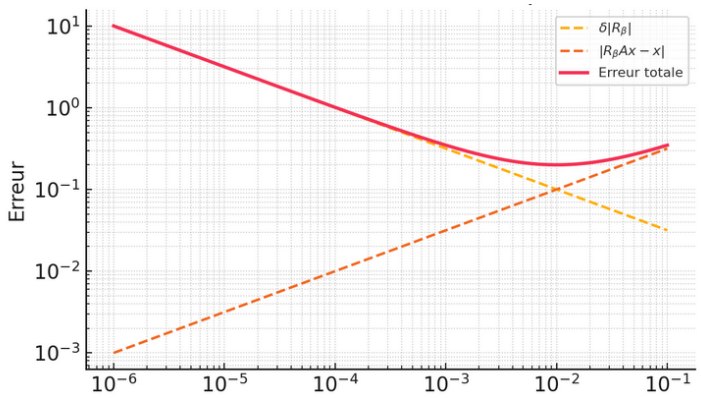


FIGURE 1.3.1 : Comportement de l'erreur total

La valeur optimale du paramètre de régularisation est le point où l'erreur totale atteint son minimum.

Il existe plusieurs techniques de régularisation développées pour traiter les problèmes inverses mal posés. Chaque méthode repose sur une stratégie spécifique pour stabiliser la solution. Ces techniques présentent chacune des avantages et des inconvénients, en fonction du type de problème considéré, de la nature des données et du niveau de bruit. Dans ce travail, nous nous concentrerons particulièrement sur la méthode de régularisation de Tikhonov, en raison de sa simplicité conceptuelle et de sa pertinence pour les problèmes inverses linéaires que nous traitons.

1.3.2 La Méthode de Tikhonov

La méthode de Tikhonov est la technique de régularisation la plus utilisée pour résoudre les problèmes inverses mal posés. Elle a été introduite par le mathématicien russe Andreï Nikolaïevitch Tikhonov.

Formulation mathématique du problème régularisé

Le principe de la méthode de Tikhonov repose sur la minimisation d'une fonctionnelle combinant l'ajustement aux données et une pénalisation destinée à stabiliser la solution. Cette fonctionnelle s'écrit :

$$J(x) = \|Ax - y\|_Y^2 + \beta\|L(x - x_0)\|_X^2 \tag{1.6}$$

où :

- $\beta > 0$ *est le paramètre de régularisation qui contrôle l'équilibre entre la fidélité aux données et la régularité de la solution,*
- L *est un opérateur linéaire (souvent la matrice identité* I*) imposant une contrainte sur la solution (norme, lissité, etc.),*
- x_0 *est une estimation a priori de la solution (souvent prise nulle).*

Dans la forme la plus courante, dite standard, on prend $L = I$ *et* $x_0 = 0$*, ce qui conduit à la formulation suivante :*

$$J(x) = \|Ax - y\|_Y^2 + \beta\|x\|_X^2 \tag{1.7}$$

Théorème 1.3.3 (3) *Soit* $A : X \to Y$ *un opérateur linéaire borné entre espaces de Hilbert. Alors,* $J(x)$ *admet un minimum unique* $x_\beta \in X$*. Ce minimum est l'unique solution de l'équation normale :*

$$(A^*A + \beta I)x_\beta = A^*y \tag{1.8}$$

Remarque 1.3.2 *Puisque (1.7) admet un minimum unique et satisfait (1.8), nous concluons que l'opérateur* $(\beta I + A^*A)^{-1}$ *est inversible, et que la solution unique est donnée par :*

$$x = (\beta I + A^*A)^{-1}A^*y := R_\beta y \tag{1.9}$$

où l'opérateur de régularisation $R_\beta : Y \to X$ *est défini par :*

$$R_\beta := (\beta I + A^*A)^{-1}A^* \tag{1.10}$$

Supposons maintenant que y *soit donné par son approximation* y^δ *(voir (1.4)), et que l'on résolve l'équation* $Ax = y^\delta$*, alors :*

$$x^\delta = (\beta I + A^*A)^{-1}A^*y^\delta := R_\beta y^\delta \tag{1.11}$$

Méthodes de choix du paramètre de régularisation

Plusieurs techniques ont été développées pour choisir le paramètre de régularisation β *de manière optimale. Chacune repose sur une philosophie différente et convient à divers contextes d'application. Nous présentons ici les méthodes les plus courantes.*

1. *Principe de Morozov (Critère de la Discrépance) repose sur l'idée que la solution régularisée doit reconstituer les données avec un écart compatible avec le niveau de bruit présent dans les observations. On cherche ainsi le plus petit α tel que :*

$$\|Ax_\beta - y^\delta\| \leq \tau\delta \tag{1.12}$$

où δ est le niveau de bruit supposé sur les données y^δ, et $\tau \geq 1$ est une constante fixée. Cette méthode suppose que l'on dispose d'une estimation fiable du bruit.

2. *Méthode de la courbe en L est une approche graphique basée sur la représentation, en échelle logarithmique, de la norme du résidu $\|Ax_\beta - y\|$ en fonction de la norme de régularisation $\|x_\beta\|$. Cette courbe a généralement la forme d'un « L » et le coin de cette courbe correspond à un bon compromis entre régularité et fidélité aux données. Le paramètre β associé à ce point est choisi comme optimal.*

3. *Validation croisée (Cross-validation) est une méthode statistique qui consiste à diviser les données en un ensemble d'apprentissage et un ensemble de validation. Pour différentes valeurs de β, on résout le problème sur l'ensemble d'apprentissage, puis on évalue l'erreur sur l'ensemble de validation. Le paramètre β qui minimise l'erreur de validation est retenu. Cette méthode est particulièrement utile lorsque le niveau de bruit n'est pas connu.*

4. *Critère de généralisation (GCV) est une version automatisée de la validation croisée. Il ne nécessite pas de partition explicite des données et repose sur une expression analytique de l'erreur moyenne de prédiction. Il est souvent utilisé pour son efficacité numérique et son caractère non supervisé.*

CHAPITRE 2

Problème inverse associé à l'équation de la chaleur

Dans ce chapitre, nous étudions un problème inverse associé à une équation de la chaleur en une dimension. Avant de formuler le problème inverse, nous rappelons d'abord le cadre du problème direct.

2.1 Formulation et résolution du problème

L'équation de la chaleur en une dimension est une équation aux dérivées partielles qui décrit l'évolution de la température $u(x,t)$ *dans un domaine spatial* $0 < x < L$ *au cours du temps* $t > 0$. *L'équation générale est donnée par :*

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2}, \quad 0 < x < L,\ t > 0$$

où :

- $u(x,t)$ *représente la température en un point* x *du domaine au temps* t,
- α *est la diffusivité thermique (constante positive)*
- L *est la longueur du domaine spatial,*

2.1.1 Conditions aux limites et condition initiale

Le problème direct est bien défini en ajoutant les conditions suivantes :

Conditions aux limites *: données sur les bords $x = a$ et $x = b$. On distingue plusieurs types :*

Condition de Dirichlet (température imposée) :

$$u(a,t) = \phi_1(t), \quad u(b,t) = \phi_2(t).$$

Condition de Neumann (flux imposé) :

$$\frac{\partial u}{\partial x}(a,t) = g_1(t), \quad \frac{\partial u}{\partial x}(b,t) = g_2(t).$$

Représente un flux de chaleur imposé ou une isolation thermique (si dérivée nulle)

Condition de Robin :

$$\frac{\partial u}{\partial x}(x,t) + \sigma u(x,t) = \gamma,$$

où σ est un coefficient positif d'échange, et γ un terme constant. Dans le cas homogène ($\gamma = 0$), cette condition devient :

$$\frac{\partial u}{\partial x}(x,t) + \beta u(x,t) = 0.$$

Utilisé pour modéliser un contact avec un fluide à température nulle ou négligée.

Condition initiale *:*

$$u(x,0) = f(x), \quad x \in [0,L],$$

où $f(x)$ est la distribution initiale de température.

2.1.2 Résolution du problème direct

Dans cette partie, nous nous intéressons à la résolution du ***problème direct*** *donné par le système suivant :*

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \dfrac{\partial^2 u}{\partial x^2}, & x \in (0,L),\ t \in (0,T], \\ u(0,t) = 0, \quad u(L,t) = 0, & t \in (0,T], \\ u(x,0) = f(x), & x \in (0,L). \end{cases} \tag{2.1}$$

Objectif *: Déterminer la solution $u(x,t)$ pour $t > 0$, connaissant la condition initiale $f(x)$.*

Pour résoudre ce problème, nous appliquons la méthode de séparation des variables, en cherchant une solution de l'équation de la chaleur sous la forme :

$$u(x,t) = X(x)T(t).$$

En substituant cette forme dans l'équation, on obtient :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \quad \Rightarrow \quad X(x)T'(t) = \alpha X''(x)T(t).$$

En divisant par $\alpha X(x)T(t)$ (en supposant $X \neq 0$ et $T \neq 0$) :

$$\frac{T'(t)}{\alpha T(t)} = \frac{X''(x)}{X(x)} = -\lambda,$$

où λ est une constante de séparation. On obtient alors deux équations différentielles ordinaires :

$$\begin{cases} X''(x) + \lambda X(x) = 0, \\ T'(t) + \alpha \lambda T(t) = 0. \end{cases}$$

Nous résolvons d'abord l'équation spatiale :

$$X''(x) + \lambda X(x) = 0, \quad X(0) = X(L) = 0.$$

1. *Si $\lambda = 0$: La solution générale est $X(x) = ax + b$. Les conditions aux limites imposent $X(0) = b = 0$ et $X(L) = aL + b = 0 \Rightarrow a = 0$. Donc $X(x) \equiv 0$: solution triviale.*
2. *Si $\lambda < 0$: Notons $\lambda = -\mu^2$ avec $\mu > 0$. On obtient $X(x) = Ae^{\mu x} + Be^{-\mu x}$, et en appliquant les conditions aux limites, on montre que $X(x) \equiv 0$.*
3. *Si $\lambda > 0$: Posons $\lambda = \omega^2$. L'équation devient :*

   $$X'' + \omega^2 X = 0 \Rightarrow X(x) = A\cos(\omega x) + B\sin(\omega x).$$

   En appliquant la condition $X(0) = 0$, on obtient :

   $$X(0) = 0 \Rightarrow A = 0$$

   donc $X(x) = B\sin(\omega x)$. En appliquant la condition $X(L) = 0$, on obtient :

   $$X(L) = 0 \Rightarrow \sin(\omega L) = 0 \Rightarrow \omega L = n\pi \Rightarrow \omega = \frac{n\pi}{L}$$

Ainsi, les valeurs propres sont :

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2, \quad n \in \mathbb{N}^*,$$

et les fonctions propres associées :

$$X_n(x) = \sin\left(\frac{n\pi x}{L}\right)$$

Remarque *Les fonctions propres $\{X_n\}_{n=1}^{\infty}$ forment une base hilbertienne orthogonale de $L^2(0,L)$, c'est-à-dire :*

1. $\langle X_n, X_m\rangle = 0$ *si* $n \neq m$,
2. $\|X_n\|^2_{L^2(0,L)} = \dfrac{L}{2}$,
3. *toute fonction $g \in L^2(0,L)$ s'écrit sous la forme suivante :*

$$g(x) = \sum_{n=1}^{\infty} g_n X_n(x), \quad \text{avec} \quad g_n = \frac{2}{L}\int_0^L g(x)\sin\left(\frac{n\pi x}{L}\right)dx.$$

L'équation différentielle associée à la variable temporelle est :

$$T'(t) + \alpha\lambda_n T(t) = 0,$$

il s'agit d'une équation différentielle linéaire du premier ordre. Sa solution générale est donnée par :

$$T_n(t) = C_n e^{-\alpha\lambda_n t} = C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t},$$

où C_n est une constante d'intégration déterminée par la condition initiale. La solution partielle associée à la n-ième fonction propre s'écrit :

$$u_n(x,t) = C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}\sin\left(\frac{n\pi x}{L}\right)$$

Par le principe de superposition (problème linéaire), la solution générale est donnée par :

$$u(x,t) = \sum_{n=1}^{\infty} C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}\sin\left(\frac{n\pi x}{L}\right) \tag{2.2}$$

Les coefficients C_n sont déterminés par la condition initiale $u(x,0) = f(x)$, soit :

$$f(x) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \quad \Rightarrow \quad C_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi x}{L}\right)dx$$

Dans la suite, nous allons détailler le problème inverse, mettre en évidence son caractère instable, puis introduire une méthode de régularisation appropriée pour le stabiliser.

2.1.3 Résolution du problème inverse

Dans cette partie, nous nous intéressons à la résolution du ***problème inverse*** *associé à l'équation de la chaleur unidimensionnelle donnée par le système suivant :*

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \dfrac{\partial^2 u}{\partial x^2}, & x \in (0, L),\ t \in (0, T], \\ u(0,t) = 0, \quad u(L,t) = 0, & t \in (0, T], \\ u(x,0) = f(x), & x \in (0, L), \quad \textit{(inconnue)}, \\ u(x,T) = h(x), & x \in (0, L) \end{cases} \tag{2.3}$$

Objectif *: Retrouver la condition initiale $f(x)$ à partir de la donnée finale $h(x)$. Pour cela, on procède en plusieurs étapes :*

On remplace dans la solution générale (2.2) du problème direct $t = T$, on obtient :

$$u(x,T) = h(x) = \sum_{n=1}^{\infty} C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} \sin\left(\frac{n\pi x}{L}\right) \tag{2.4}$$

On projette $h(x)$ sur la base orthogonale $\left\{\sin\left(\frac{n\pi x}{L}\right)\right\}_{n\in\mathbb{N}^*}$ en multipliant chaque côté par $\sin\left(\frac{m\pi x}{L}\right)$ et en intégrant sur $[0, L]$:

$$\int_0^L h(x) \sin\left(\frac{m\pi x}{L}\right) dx = \sum_{n=1}^{\infty} C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} \int_0^L \sin\left(\frac{n\pi x}{L}\right) \sin\left(\frac{m\pi x}{L}\right) dx.$$

Par orthogonalité :

$$\int_0^L \sin\left(\frac{n\pi x}{L}\right) \sin\left(\frac{m\pi x}{L}\right) dx = \begin{cases} \frac{L}{2}, & si\ n = m, \\ 0, & si\ n \neq m. \end{cases}$$

On obtient alors :

$$\int_0^L h(x) \sin\left(\frac{n\pi x}{L}\right) dx = C_n e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} \cdot \frac{L}{2}.$$

On en déduit :

$$C_n = \frac{2}{L} e^{\alpha\left(\frac{n\pi}{L}\right)^2 T} \int_0^L h(x) \sin\left(\frac{n\pi x}{L}\right) dx.$$

Une fois les C_n obtenus, on retrouve $f(x)$ en utilisant :

$$f(x) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \tag{2.5}$$

La solution du problème inverse est donc un ***ensemble de couples*** $(f(x), u(x,t))$ *donnée par (2.2) et (2.5).*

Nous démontrons ici que le problème inverse (2.3) est mal posé au sens d'Hadamard.

Le lien entre la donnée finale $u(x,T)$ *et la condition initiale* $f(x)$ *peut s'écrire sous la forme opératorielle suivante :*

$$(Af)(x) = \sum_{n=1}^{\infty} e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} f_n X_n(x) \tag{2.5}$$

où $A : f \mapsto u(x,T)$ *est un opérateur linéaire de* $L^2(0,L)$ *dans lui-même.*

Pour montrer que A *est un opérateur compact, considérons l'opérateur de rang fini* A^m *défini par :*

$$A^m f(x) = \sum_{n=1}^{m} e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} f_n X_n(x)$$

On a alors la relation suivante :

$$\|A^m f - Af\|_{L^2}^2 = \sum_{n=m+1}^{\infty} e^{-2\alpha\left(\frac{n\pi}{L}\right)^2 T} |f_n|^2.$$

Comme les coefficients $e^{-2\alpha\left(\frac{n\pi}{L}\right)^2 T} \to 0$ *très rapidement lorsque* $n \to \infty$*, il en résulte que* $\|A^m f - Af\|_{L^2} \to 0$*. Ainsi,* A *est la limite d'une suite d'opérateurs compacts de rang fini, donc compact. Considérons maintenant l'opérateur* A^*A*, où* A^* *est l'adjoint de* A*. Comme les fonctions* $\{X_n\}_{n=1}^{\infty}$ *forment une base orthogonal de* $L^2(0,L)$*, on a :*

$$A^*AX_n = e^{-2\alpha\left(\frac{n\pi}{L}\right)^2 T} X_n,$$

ce qui montre que les valeurs propres de A^*A *sont données par :* $e^{-2\alpha\left(\frac{n\pi}{L}\right)^2 T}$*. Les valeurs singulières de* A *sont donc :*

$$\sigma_n = e^{-\alpha\left(\frac{n\pi}{L}\right)^2 T} \tag{2.6}$$

Ces valeurs singulières décroissent exponentiellement vers zéro, ce qui signifie que l'inversion de l'opérateur A *amplifie fortement les erreurs contenues dans les données finales* $u(x,T)$*.*

D'après les résultats classiques de la théorie des problèmes inverses (cf. théorie spectrale des opérateurs compacts), cela établit que le problème est ***mal posé*** *: la solution ne dépend pas continûment des données, rendant nécessaire l'emploi de techniques de régularisation pour en obtenir une solution stable.*

2.2 Régularisation par la méthode de Tikhonov

La méthode de Tikhonov consiste à résoudre un problème de minimisation de la forme :

$$\hat{f} = \arg\min_f \left(\|A(f) - h^\delta\|_2^2 + \beta \|f\|_2^2 \right),$$

où $\beta > 0$ est le paramètre de régularisation.

Le terme $\beta\|f\|_2^2$ agit comme une barrière qui empêche la solution de devenir trop instable, en limitant notamment l'influence des hautes fréquences dans la reconstruction de f.

La solution régularisée selon Tikhonov s'écrit alors :

$$f_\beta^\delta(x) = \sum_{n=1}^{\infty} \frac{e^{\frac{n\pi}{L}T}}{e^{2\frac{n\pi}{L}T} + \beta} h_n^\delta \sin\left(\frac{n\pi x}{L}\right) \tag{2.7}$$

où h_n^δ sont les coefficients de Fourier de la donnée finale bruitée h^δ.

Dans notre travail, le paramètre β est déterminé de manière optimale en utilisant le ***principe de Morozov****, qui impose la condition suivante :*

$$\|Af_\beta^\delta - h^\delta\|_{L^2} = \delta \tag{2.8}$$

où δ représente le niveau de bruit dans les données.

CHAPITRE 3

Optimisation du paramètre de régularisation par intelligence artificielle

La résolution des problèmes inverses mal posés nécessite toujours une régularisation pour stabiliser la solution. Le paramètre de régularisation noté dans le chapitre précédent β, joue un rôle important dans l'équilibre entre fidélité aux données et stabilité de la solution. Cependant, le choix de ce paramètre reste une difficulté majeure lorsque les données sont bruitées. Dans ce chapitre, nous présentons les idées principales de l'intelligence artificielle et comment elle peut aider à choisir automatiquement (les réseaux de neurones artificiels (ANN))ce paramètre.

3.1 Introduction à l'intelligence artificielle

L'intelligence artificielle (IA) désigne un ensemble de techniques visant à simuler l'intelligence humaine à l'aide d'algorithmes capables d'apprendre, de raisonner, de percevoir ou d'agir. Ces dernières années, l'IA a connu un développement spectaculaire, notamment grâce aux avancées en puissance de calcul, à la disponibilité massive de données et à l'amélioration des algorithmes d'apprentissage. Dans le contexte scientifique et technique, l'IA est utilisée pour résoudre des problèmes complexes où les approches analytiques ou

classiques atteignent leurs limites, notamment dans les domaines du traitement du signal, de l'imagerie médicale, du diagnostic, de l'optimisation et de la résolution de problèmes inverses.

3.2 Apprentissage automatique

L'apprentissage automatique (machine learning) est une sous-discipline de l'intelligence artificielle qui permet à un système d'apprendre à partir de données, sans avoir été explicitement programmé pour chaque tâche. Il s'appuie sur des algorithmes capables de détecter des régularités ou des motifs dans les données, puis de généraliser à de nouvelles situations. On distingue généralement trois grandes familles d'apprentissage :

1. ***L'apprentissage supervisé*** *: L'apprentissage supervisé consiste à entraîner un modèle à partir d'un ensemble de données étiquetées : à chaque observation x_i, on associe une sortie connue. L'objectif est de minimiser une fonction de coût qui mesure l'écart entre la prédiction du modèle et la vérité terrain.*
2. ***L'apprentissage non supervisé*** *: L'apprentissage non supervisé, quant à lui, est utilisé pour analyser la structure des données sans disposer de sorties cibles. Il permet, par exemple, de réaliser du regroupement (clustering), de la réduction de dimension ou de l'analyse exploratoire. Toutefois, ce type d'apprentissage n'est pas adapté à notre problématique, car nous cherchons à estimer de manière quantitative et précise le paramètre β.*
3. ***L'apprentissage par renforcement*** *: L'apprentissage par renforcement repose sur une interaction entre un agent et un environnement. Contrairement aux apprentissages supervisé et non supervisé, le système ne reçoit pas directement la réponse correcte, mais apprend par essais et erreurs, en fonction des récompenses ou pénalités reçues. Ce type d'apprentissage est moins pertinent dans notre cas, car il nécessite la définition explicite d'un environnement d'action et d'un mécanisme de rétroaction continue.*

Dans notre cas, nous utilisons l'apprentissage supervisé, où le but est d'apprendre une fonction de prédiction qui associe à chaque entrée (données bruitées) un paramètre optimal de régularisation β.

Parmi les algorithmes d'apprentissage supervisé les plus utilisés, on trouve la régression linéaire, les forêts aléatoires, les machines à vecteurs de support (SVM), ainsi que les réseaux de neurones.

3.3 Réseaux de neurones artificiels

Un réseau de neurones artificiels est un système composé d'un ensemble de processeurs élémentaires fonctionnant en parallèle, appelés neurones artificiels. Chaque neurone reçoit un ensemble d'entrées pondérées, effectue un traitement interne (souvent linéaire suivi d'une non-linéarité), puis transmet une sortie vers d'autres neurones du réseau.

L'origine conceptuelle de ces modèles provient du système nerveux biologique. En effet, un neurone biologique est une cellule nerveuse connectée à d'autres neurones par des synapses. En fonction des signaux reçus, le noyau du neurone décide ou non d'activer un signal électrique transmis le long de l'axone, vers d'autres cellules.

Cette structure biologique a inspiré la conception des réseaux de neurones artificiels, dans lesquels :

1. *les **synapses** sont modélisées par des poids associés aux connexions entre neurones,*
2. *le **noyau** correspond à la fonction d'activation (comme ReLU, sigmoid),*
3. *l'**axone** représente la sortie du neurone.*

La figure 3.1 illustre l'analogie entre un neurone biologique, entité élémentaire du système nerveux, et un neurone artificiel.

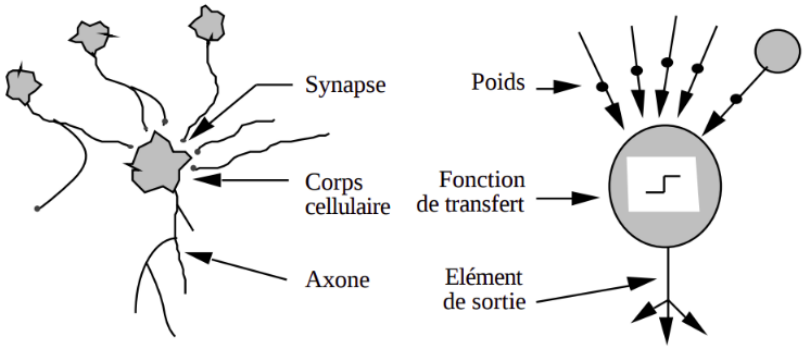


FIGURE 3.3.1 : Comparaison entre un neurone biologique et un neurone artificiel

Ainsi, un réseau de neurones artificiels constitue une modélisation computationnelle de certains mécanismes cognitifs, permettant notamment de résoudre des tâches complexes telles que la reconnaissance de formes, la classification, ou encore la résolution de problèmes inverses en physique.

3.3.1 Principe de fonctionnement

Un réseau de neurones est généralement composé de trois types de couches :

1. *Une couche d'entrée qui lit les données d'entrée : Dans notre situation elle reçoit les données d'observation g.*
2. *une couche (ou plusieurs) cachée qui effectue des modifications sur les entrées en leur appliquant une fonction de transfert*
3. *Une couche de sortie qui fournit la sortie : Dans notre cas, elle contient un seul neurone activé linéairement, prédisant le paramètre de régularisation β.*

Chaque connexion entre neurones est associée à un poids, ajusté lors de la phase d'apprentissage afin de minimiser l'erreur entre la sortie prédite et la sortie réelle.

3.3.2 Fonctions d'activation

Les fonctions d'activation jouent un rôle fondamental dans les réseaux de neurones artificiels. Elles introduisent une non-linéarité entre l'entrée et la sortie de chaque neurone, ce qui permet au réseau de modéliser des relations complexes dans les données.

Il existe des fonctions d'activation linéaires et non linéaires. Les fonctions linéaires, bien que simples, sont rarement utilisées car elles limitent la capacité du réseau à représenter uniquement des transformations linéaires. Or, dans la plupart des cas réels, les phénomènes à modéliser présentent des dépendances non linéaires entre les variables.

Les fonctions d'activation non linéaires permettent donc au réseau d'apprendre des comportements plus riches et adaptés à des tâches complexes telles que la classification, la régression ou la reconnaissance de motifs.

Les fonctions les plus utilisées dans la pratique sont résumées dans le tableau ci-dessous :

Nom	**Formule**	**Plage de sortie**
ReLU	$\mathrm{ReLU}(x) = \max(0, x)$	$[0, +\infty[$
Sigmoïde	$\sigma(x) = \dfrac{1}{1+e^{-x}}$	$]0, 1[$
Tanh	$\tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$[-1, 1]$
Leaky ReLU	$\max(\alpha x, x)$ avec $\alpha \ll 1$	$\mathbb{R}$

Table 3.3.1 : Fonctions d'activation couramment utilisées dans les réseaux de neurones

Le choix de la fonction d'activation dépend du problème traité, du type de réseau et du comportement attendu en sortie.

3.4 Application des réseaux de neurones à la résolution du problème inverse de l'équation de la chaleur

Dans cette section, nous explorons l'utilisation des réseaux de neurones profonds pour résoudre un problème inverse associé à l'équation de la chaleur en une dimension. L'ob-

jectif est de reconstruire la condition initiale d'un système à partir de mesures bruitées de sa solution à un instant final.

3.4.1 Formulation comme un problème d'apprentissage supervisé

Le problème inverse est formulé comme une tâche d'apprentissage supervisé où :

- ***Entrée du réseau** : une mesure bruitée de la solution finale, notée $h^\delta(x) = u^\delta(x, T)$, discrétisée en un vecteur de dimension n ;*
- ***Sortie attendue** : une estimation optimale du paramètre β, utilisé pour la régularisation de Tikhonov.*

3.4.2 Architecture du modèle proposé

L'architecture adoptée combine deux composantes complémentaires : un réseau convolutionnel unidimensionnel (CNN 1D) et un perceptron multicouche (MLP).

CNN 1D (Convolutional Neural Network)

Le CNN est utilisé pour extraire les caractéristiques locales du vecteur h_n :

- *Filtres convolutifs pour détecter des motifs ;*
- *Couches d'activation (ReLU, etc.) ;*
- *Couches de pooling et de normalisation.*

MLP (Perceptron multicouche)

Le MLP est conçu pour modéliser les paramètres globaux du problème (T, δ) :

- *Couches denses connectées ;*
- *Fonctions d'activation non linéaires (ReLU, tanh) ;*
- *Capacité à approximer des fonctions complexes.*

Fusion neuronale

Les sorties du CNN et du MLP sont concaténées, puis traitées par des couches supplémentaires pour produire une prédiction robuste du paramètre β.

3.4.3 Intérêt de l'approche hybride

Cette architecture hybride permet :

- *d'extraire la structure fine du signal final bruité via le CNN ;*
- *de modéliser les paramètres globaux via le MLP ;*
- *de combiner ces deux types d'information pour une estimation précise et stable de β.*

CHAPITRE 4

Expérimentations et résultats

4.1 Langage et bibliothèques utilisées

Python est un langage de programmation open source très utilisé dans les domaines scientifiques, techniques et industriels. Grâce à sa syntaxe simple et lisible, Python permet aux développeurs de se concentrer sur la logique de résolution plutôt que sur des détails syntaxiques complexes. Ce langage s'est largement imposé dans des domaines comme l'intelligence artificielle, l'analyse de données, la simulation numérique, ou encore le développement d'interfaces. Sa communauté active, sa documentation abondante et la richesse de ses bibliothèques font de Python un choix privilégié dans les projets informatiques modernes, notamment ceux liés à la recherche scientifique et à l'innovation. Dans le cadre de ce travail, le langage de programmation Python a été retenu pour sa lisibilité, sa polyvalence, ainsi que pour la richesse de son écosystème de bibliothèques scientifiques. Les principales bibliothèques utilisées sont les suivantes :

1. ***PyTorch*** *: Bibliothèque open source pour le calcul scientifique et l'apprentissage automatique, particulièrement adaptée aux réseaux de neurones grâce à son interface dynamique et intuitive. Utilisée ici pour définir, entraîner et évaluer des modèles en imagerie médicale.*

Figure 4.1.1 : Logo de Python

2. ***Streamlit*** *: Framework open source permettant de créer facilement des applications web interactives en Python, idéal pour intégrer des visualisations et des interfaces utilisateur. Employé pour développer une interface interactive de visualisation et d'ajustement des paramètres.*

3. ***NumPy*** *: Bibliothèque essentielle pour la manipulation de tableaux multidimensionnels et les opérations mathématiques vectorisées. Utilisée pour le prétraitement, la génération de bruit et les calculs matriciels liés aux problèmes inverses.*

4. ***Matplotlib*** *: Bibliothèque de visualisation 2D de haute qualité, souvent utilisée pour représenter graphiques d'erreur, comparaisons d'images, et courbes d'optimisation. Employée pour visualiser les résultats des méthodes de régularisation.*

5. ***Pandas*** *: Bibliothèque puissante pour la manipulation de données tabulaires via les DataFrames. Utilisée pour organiser, comparer et exporter les résultats sous forme de tableaux récapitulatifs.*

4.2 Résultats numériques

Ce chapitre présente une série d'expérimentations numériques visant à résoudre un problème inverse associé à l'équation de la chaleur. Plus précisément, il s'agit de reconstituer une condition initiale $f(x)$ à partir d'une donnée finale bruitée $h^\delta(x)$, mesurée à un

instant final T. Ce type de problème est fortement mal posé, et nécessite une régularisation efficace pour obtenir une solution stable et interprétable.

Pour cela, nous comparons deux approches complémentaires permettant de déterminer le paramètre de régularisation β dans le cadre de la méthode de Tikhonov :

1. ***Méthode classique** : cette approche repose sur la régularisation de Tikhonov, avec un choix automatique du paramètre β par la **règle de Morozov**. Cette règle ajuste β de manière à ce que le résidu entre la solution régularisée et les données soit proportionnel au bruit. Elle est robuste, mais nécessite une procédure d'optimisation numérique (recherche de β par essais).*
2. ***Méthode par réseau de neurones (IA)** : dans cette approche innovante, un modèle de réseau de neurones est entraîné à prédire directement une bonne valeur de β, à partir des données bruitées h^δ, du temps final T, et du niveau de bruit δ. Le modèle est conçu pour apprendre à minimiser l'erreur de reconstruction, et ainsi automatiser le choix de β de manière rapide et efficace.*

*Pour enrichir la comparaison, nous introduisons également une **troisième approche dite hybride**, qui combine les deux méthodes :*

- *on commence par estimer un premier β via le réseau de neurones (β_{NN}) ;*
- *puis on applique localement la règle de Morozov autour de cette valeur pour obtenir une estimation raffinée (β_{Fusion}).*

Ces différentes approches ont été intégrées dans une application interactive développée avec deux bibliothèques Python :

1. ***Streamlit**, pour construire une interface conviviale permettant de visualiser les résultats en temps réel, ajuster les paramètres (T, δ), et tester les différentes méthodes de sélection de β ;*
2. ***PyTorch**, utilisée pour définir, entraîner et déployer le réseau de neurones, basé sur une architecture hybride combinant des couches convolutives (pour analyser les données spectrales h^n) et des couches entièrement connectées (pour les paramètres globaux).*

Les résultats présentés dans les sections suivantes permettent d'analyser en détail les performances respectives de ces méthodes, en termes :

- *de qualité de reconstruction de la solution $f(x)$,*
- *de précision sur le choix de β,*
- *et de robustesse face au bruit et à la variation du paramètre temporel T.*

Exemple 4.2.1 *Considérons le problème inverse de la chaleur dans un domaine unidimensionnel $x \in (0,1)$ et pour un temps final fixé $T > 0$. L'objectif est de reconstruire la condition initiale $f(x)$ à partir d'une mesure bruitée $u^\delta(x,T)$ de la solution $u(x,t)$ à l'instant $t = T$ de l'équation de la chaleur suivante :*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad u(x,0) = f(x), \quad u(0,t) = u(1,t) = 0.$$

Le but est de retrouver $f(x)$ à partir de la solution $u(x,T)$ observée à un instant final $T > 0$, en présence de bruit. Il s'agit donc d'un problème inverse de la chaleur, typiquement mal posé.

Dans le cadre de cette expérimentation, la solution exacte utilisée comme référence est :

$$f(x) = \sin(\pi x),$$

pour laquelle la solution analytique directe est :

$$u(x,t) = \sin(\pi x) e^{-\pi^2 t}.$$

Pour mettre en évidence l'instabilité de ce problème inverse, nous comparons ci-dessous :

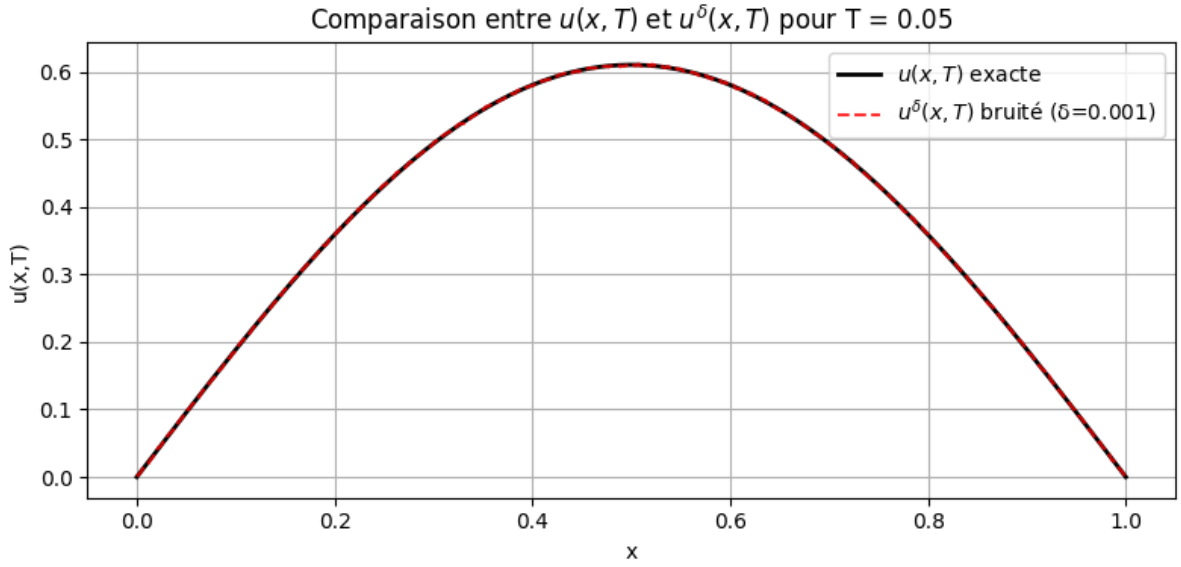


FIGURE 4.2.1 : Comparaison entre $u(x,T)$ et sa version bruitée $u^\delta(x,T)$.

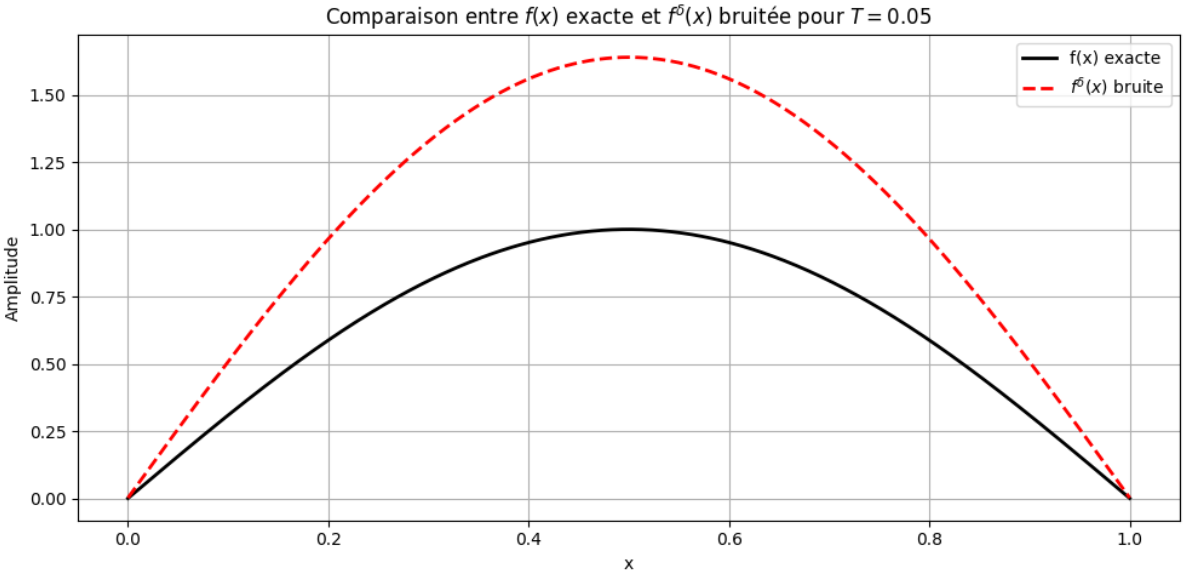


FIGURE 4.2.2 : Comparaison entre la fonction exacte f et sa version bruitée f^δ.

Les figures 4.2.1 et 4.2.2 montrent que la reconstruction directe de la condition initiale à partir de données bruitées conduit à des solutions instables et fortement dégradées.

Dans ce travail, nous présentons et comparons deux approches permettant de stabiliser numériquement la solution. Chacune de ces méthodes sera étudiée en détail, puis

testée numériquement sur le problème inverse associé à l'équation de la chaleur présentée précédemment.

4.2.1 Mise en œuvre de la régularisation de Tikhonov avec choix du paramètre par la règle de Morozov

Cette section présente la mise en œuvre de la régularisation de Tikhonov pour résoudre le problème inverse associé à l'équation de la chaleur. Le paramètre de régularisation β est automatiquement ajusté selon le principe de Morozov, qui impose que l'écart entre la solution reconstruite et les données bruitées corresponde au niveau de bruit supposé.

L'implémentation numérique a été réalisée à l'aide de l'interface ***Streamlit****, permettant une visualisation interactive et une manipulation intuitive des paramètres. Trois fonctions mathématiques principales sont utilisées dans le code pour encadrer la démarche de résolution :*

1. Projection sur la base de Fourier-sinus : `compute_hn`

Listing 4.1 : Calcul des coefficients h_n

```
def compute_hn(h, M, x, L):
    return np.array([2/L * np.trapz(h * np.sin(n*np.pi*x/L), x) for n in
        range(1, M+1)])
```

Cette fonction calcule les coefficients de Fourier-sinus de la fonction bruitée $h^\delta(x)$ par projection sur la base $\{\sin(n\pi x/L)\}_{n=1}^{M}$. Ces coefficients sont définis par :

$$h_n = \frac{2}{L}\int_0^L h^\delta(x)\sin\left(\frac{n\pi x}{L}\right)\,dx, \quad \textit{pour } n = 1,\ldots,M.$$

L'intégration est réalisée numériquement à l'aide de la méthode des trapèzes. Ces coefficients représentent les données transformées dans la base propre à l'opérateur de diffusion.

2. Reconstruction régularisée : `compute_reconstruction`

Listing 4.2 : Reconstruction régularisée à partir des h_n

```
def compute_reconstruction(beta, h_n, x, L, T):
    recon = np.zeros_like(x)
    for n in range(1, len(h_n)+1):
        lambda_n = (n*np.pi/L)**2
        sigma_n = np.exp(-lambda_n*T)
        recon += (sigma_n / (sigma_n**2 + beta)) * h_n[n-1] * np.sin(n*np.
            pi*x/L)
    return recon
```

Cette fonction utilise les coefficients h_n calculés précédemment pour reconstruire une approximation stable de la solution $f(x)$, selon la régularisation de Tikhonov spectrale :

$$f_\beta(x) = \sum_{n=1}^{M} \frac{\sigma_n}{\sigma_n^2 + \beta} \cdot h_n \cdot \sin\left(\frac{n\pi x}{L}\right), \quad \text{avec } \sigma_n = e^{-\lambda_n T},\ \lambda_n = \left(\frac{n\pi}{L}\right)^2.$$

3. Choix automatique de l'intervalle de recherche

Le principe de Morozov permet de déterminer la valeur optimale du paramètre de régularisation. L'algorithme implémente ce principe en deux étapes successives :

— ***Estimation initiale par grille logarithmique :*** *La fonction résiduelle*

$$r(\beta) = \left| \|A_\beta f - h^\delta\| - \delta \right|$$

est évaluée pour $\beta \in [10^{-10}, 1]$ sur une grille logarithmique. Cette exploration permet de repérer une zone où la fonction résiduelle atteint un minimum.

— ***Affinage par minimisation locale :*** *Une fois l'intervalle resserré autour du minimum, une minimisation locale de la fonction $r(\beta)$ est effectuée à l'aide de la méthode de Brent (via* `scipy.optimize.minimize_scalar`*), avec une tolérance numérique élevée (*`xatol = 1e-12`*) pour assurer une grande précision.*

L'implémentation Python correspondante est donnée ci-dessous :

Listing 4.3 : Optimisation de β par la règle de Morozov

```
def refine_beta_with_morozov(h_n, h_noisy, x, L, T, delta):
    def residu(beta):
        Af_val = np.zeros_like(x)
        for n in range(1, len(h_n) + 1):
            lambda_n = (n * np.pi / L) ** 2
            sigma_n = np.exp(-lambda_n * T)
            coeff = (sigma_n / (sigma_n ** 2 + beta)) * sigma_n
            Af_val += coeff * h_n[n - 1] * np.sin(n * np.pi * x / L)
        return abs(np.sqrt(np.trapz((Af_val - h_noisy) ** 2, x)) - delta)

    beta_candidates = np.logspace(-10, 0, 100)
    residuals = [residu(beta) for beta in beta_candidates]
    min_idx = np.argmin(residuals)
    i_low = max(min_idx - 5, 0)
    i_high = min(min_idx + 5, len(beta_candidates) - 1)
    lower = beta_candidates[i_low]
    upper = beta_candidates[i_high]

    result = minimize_scalar(residu, bounds=(lower, upper), method='bounded
        ', options={'xatol': 1e-12})
    return result.x
```

Cette stratégie automatisée permet de déterminer efficacement un β optimal sans intervention manuelle, tout en respectant la contrainte imposée par le bruit. Elle est particulièrement utile pour intégrer la sélection de β dans une interface dynamique ou un processus d'apprentissage automatique.

Visualisation et résultats numériques

Les figures suivantes illustrent la qualité de la reconstruction obtenue par la méthode de Morozov pour différentes valeurs du temps final T et différents niveaux de bruit ajoutés. Pour chaque configuration, nous comparons visuellement la solution exacte f à la solution régularisée f_β.

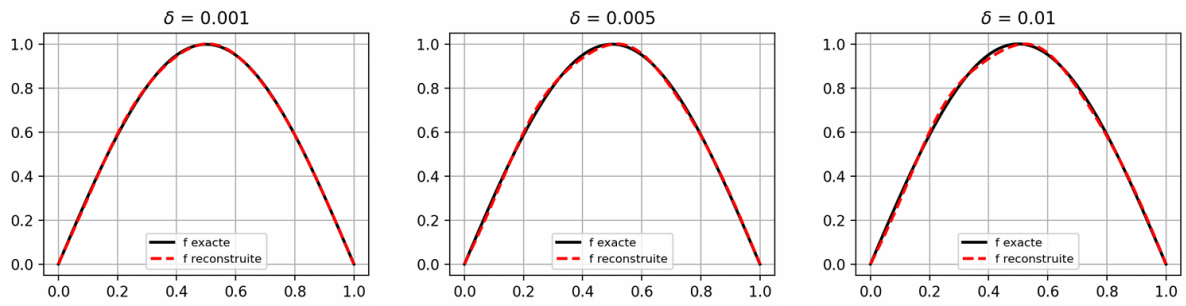


Figure 4.2.3 : Comparaison entre f exacte et f_β reconstruite (méthode de Morozov) avec $T = 0{,}01$.

	T	δ	β choisi	Erreur relative
0	0,010	0,00100	0,00019	0,00449
1	0,010	0,00500	0,00088	0,01141
2	0,010	0,01000	0,00169	0,01553

Figure 4.2.4 : Tableau des Résultats $T = 0{,}01$.

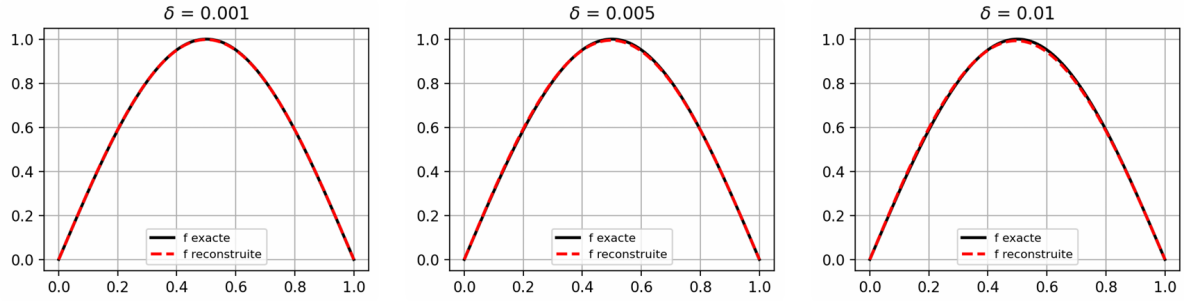


Figure 4.2.5 : Comparaison entre f exacte et f_β reconstruite (méthode de Morozov) avec $T = 0{,}05$.

	T	δ	β choisi	Erreur relative
0	0,050	0,00100	0,00010	0,00198
1	0,050	0,00500	0,00047	0,00521
2	0,050	0,01000	0,00094	0,00856

Figure 4.2.6 : Tableau des Résultats $T = 0{,}05$.

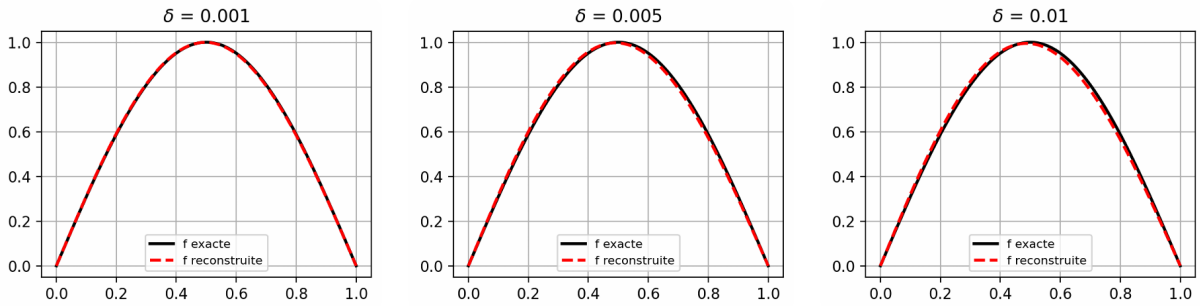


Figure 4.2.7 : Comparaison entre f exacte et f_β reconstruite (méthode de Morozov) avec $T = 0{,}1$.

	T	δ	β choisi	Erreur relative
0	0,100	0,00100	0,00006	0,00439
1	0,100	0,00500	0,00027	0,01472
2	0,100	0,01000	0,00053	0,02147

Figure 4.2.8 : Tableau des Résultats $T = 0{,}1$.

4.2.2 Résolution du problème inverse par réseaux de neurones et règle de Morozov

Dans ce qui suit, nous présentons une approche complète basée sur l'intelligence artificielle pour le choix automatique du paramètre de régularisation β. Cette approche repose sur une architecture neuronale hybride, appelée ***HybridNet****, conçue pour exploiter à la fois :*

- *les données spectrales du problème inverse (les coefficients h^n) ;*
- *les paramètres globaux, à savoir le temps final T et le niveau de bruit δ.*

L'objectif est de prédire une première estimation de β, notée β_{NN}, que nous affinerons ensuite à l'aide de la règle de Morozov, dans un intervalle restreint centré autour de cette prédiction initiale. Cette stratégie permet de limiter les coûts de calcul tout en garantissant une régularisation adaptée au niveau de bruit.

Le code que nous allons détailler met en œuvre successivement les trois composantes suivantes :

1. ***Prédiction initiale de*** *β par le réseau* `HybridNet`*, à partir des entrées du problème inverse ;*
2. ***Recherche locale*** *autour de β_{NN} pour déterminer un intervalle de régularisation pertinent ;*
3. ***Affinement final*** *par la règle de Morozov restreinte à cet intervalle, pour obtenir un paramètre optimisé β_{Fusion}.*

Les étapes principales mises en œuvre dans le code pour encadrer la démarche de résolution sont les suivantes :

1. Architecture du réseau `HybridNet`

Le modèle `HybridNet` *est un réseau de neurones hybride conçu pour prédire automatiquement un paramètre de régularisation β optimal, en exploitant à la fois les caractéristiques spectrales de la donnée bruitée et des informations globales du problème.*

Listing 4.4 : Architecture du réseau HybridNet

```
class HybridNet(nn.Module):
    def __init__(self, M):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(1, 64, kernel_size=5, padding=2),
            nn.BatchNorm1d(64),
            nn.GELU(),
            nn.MaxPool1d(2),
            nn.Conv1d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.AdaptiveAvgPool1d(1)
        )
        self.mlp = nn.Sequential(
            nn.Linear(2, 64),
            nn.LayerNorm(64),
            nn.GELU(),
            nn.Dropout(0.3)
        )
        self.head = nn.Sequential(
            nn.Linear(128 + 64, 256),
            nn.LayerNorm(256),
            nn.GELU(),
            nn.Dropout(0.4),
            nn.Linear(256, 1),
```

```
            nn.Softplus()  # garantit que beta > 0
        )

    def forward(self, params, signal):
        signal = signal.unsqueeze(1)  # (batch_size, 1, M)
        cnn_features = self.cnn(signal).squeeze(-1)  # (batch_size, 128)
        mlp_features = self.mlp(params)              # (batch_size, 64)
        combined = torch.cat([cnn_features, mlp_features], dim=1)
        return self.head(combined)                   # sortie : beta pré
           dite
```

2. Projection de la donnée bruitée en base de Fourier

Avant d'être envoyée au réseau, la donnée bruitée $h^\delta(x)$ est projetée sur une base de fonctions propres (ici les sinus) pour obtenir les coefficients h^n :

Listing 4.5 : Projection sur les bases sinus

```
def compute_hn(h, M, x):
    return np.array([
        2 / L * np.trapz(h * np.sin(n * np.pi * x / L), x)
        for n in range(1, M + 1)
    ])
```

3. Génération des données pour l'apprentissage

Pour entraîner le réseau, on génère des triplets (T, δ, h^n) associés à la cible $\beta^\star$, qui minimise l'erreur de reconstruction :

Listing 4.6 : Génération des entrées, signaux et cibles pour l'entraînement

```
def generate_data(T_list, delta_list, N_samples=200):
    inputs, signals, targets = [], [], []
    for T in T_list:
        for delta in delta_list:
            for _ in range(N_samples):
                # Donnée exacte
                h = np.exp(-np.pi ** 2 * T) * np.sin(np.pi * x)
                # Ajout de bruit gaussien
                noise = delta * np.random.randn(len(x))
```

```
                h_noisy = h + noise
                # Projection en base de Fourier
                h_n = compute_hn(h_noisy, M, x)
                # Calcul du beta optimal par minimisation directe
                res = minimize_scalar(
                    lambda b: np.linalg.norm(reconstruction(b, h_n, x, L, T
                        ) - f_exact),
                    bounds=(1e-10, 1), method='bounded', tol=1e-12
                )
                inputs.append([T, delta])
                signals.append(h_n)
                targets.append([res.x])  # beta optimal
    return np.array(inputs), np.array(signals), np.array(targets)
```

4. Entraînement du réseau de neurones

Listing 4.7 : Boucle d'entraînement du modèle

```
def train_model(model, inputs, signals, targets, epochs=500):
    input_scaler = StandardScaler()
    signal_scaler = StandardScaler()

    inputs_norm = input_scaler.fit_transform(inputs)
    signals_norm = signal_scaler.fit_transform(signals)

    inputs_t = torch.tensor(inputs_norm, dtype=torch.float32)
    signals_t = torch.tensor(signals_norm, dtype=torch.float32)
    targets_t = torch.tensor(targets, dtype=torch.float32)

    dataset = TensorDataset(inputs_t, signals_t, targets_t)
    loader = DataLoader(dataset, batch_size=64, shuffle=True)

    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-5)
    scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=1e-3,
                                              steps_per_epoch=len(loader),
                                              epochs=epochs)
    loss_fn = nn.HuberLoss()
```

```
    for epoch in range(epochs):
        model.train()
        for batch_inputs, batch_signals, batch_targets in loader:
            optimizer.zero_grad()
            preds = model(batch_inputs, batch_signals)
            loss = loss_fn(preds, batch_targets)
            loss.backward()
            optimizer.step()
            scheduler.step()
```

5.Méthode de fusion autour de la prédiction β_{NN}

La prédiction du paramètre de régularisation β_{NN} par réseau de neurones peut être affinée localement à l'aide de la règle de Morozov. L'objectif est d'ajuster précisément ce β dans un intervalle restreint centré autour de la valeur prédite.

Listing 4.8 : Détection automatique de l'intervalle autour de β_{NN}

```
def find_local_range(beta_nn, h_n, h_noisy, x, L, T, delta):
    erreurs = []
    for alpha in [0.25, 0.5, 1, 2, 4]:
        beta_test = alpha * beta_nn
        Af_val = np.zeros_like(x)
        for n in range(1, len(h_n) + 1):
            lambda_n = (n * np.pi / L) ** 2
            sigma_n = np.exp(-lambda_n * T)
            coeff = (sigma_n / (sigma_n ** 2 + beta_test)) * sigma_n
            Af_val += coeff * h_n[n - 1] * np.sin(n * np.pi * x / L)
        residu = np.sqrt(np.trapz((Af_val - h_noisy) ** 2, x))
        erreurs.append((beta_test, abs(residu - delta)))
    erreurs.sort(key=lambda x: x[1])
    return min(erreurs[0][0], erreurs[1][0]), max(erreurs[0][0], erreurs
       [1][0])
```

6. Affinement final *L'intervalle déterminé est ensuite exploité dans une optimisation de type Morozov restreinte :*

Listing 4.9 : Affinement final du paramètre β

```
def refine_beta_fusion(beta_nn, h_n, h_noisy, x, L, T, delta):
    def residu(beta):
        Af_val = np.zeros_like(x)
        for n in range(1, len(h_n) + 1):
            lambda_n = (n * np.pi / L) ** 2
            sigma_n = np.exp(-lambda_n * T)
            coeff = (sigma_n / (sigma_n ** 2 + beta)) * sigma_n
            Af_val += coeff * h_n[n - 1] * np.sin(n * np.pi * x / L)
        return abs(np.sqrt(np.trapz((Af_val - h_noisy) ** 2, x)) - delta)

    lower, upper = find_local_range(beta_nn, h_n, h_noisy, x, L, T, delta)
    result = minimize_scalar(residu, bounds=(lower, upper), method='bounded
        ', options={'xatol': 1e-12})
    return result.x
```

Visualisation et résultats numériques

Les figures suivantes illustrent la qualité de la reconstruction obtenue par la méthode de Morozov pour différentes valeurs du temps final T et différents niveaux de bruit ajoutés. Pour chaque configuration, nous comparons visuellement la solution exacte f à la solution régularisée f_β.

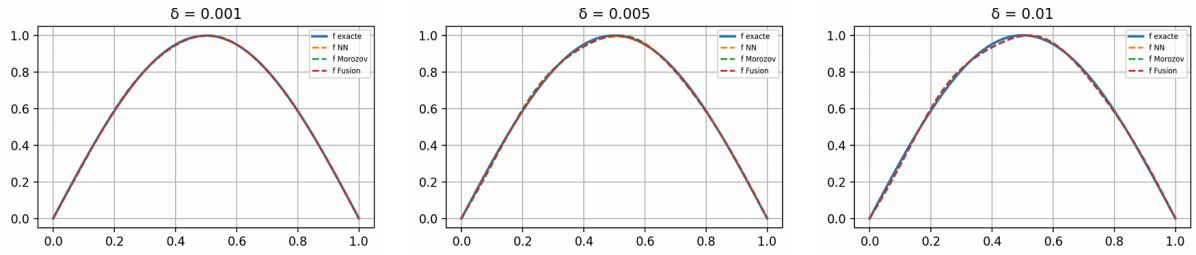


FIGURE 4.2.9 : Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}01$ et différents niveaux de bruit δ.

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.01	0.001	0.001464	0.000190	0.000827	0.00250	0.00449	0.00259
1	0.01	0.005	0.003540	0.000884	0.002212	0.00650	0.01141	0.00706
2	0.01	0.010	0.002298	0.001690	0.001994	0.01286	0.01553	0.01402

FIGURE 4.2.10 : Tableau des Résultats $T = 0{,}1$.

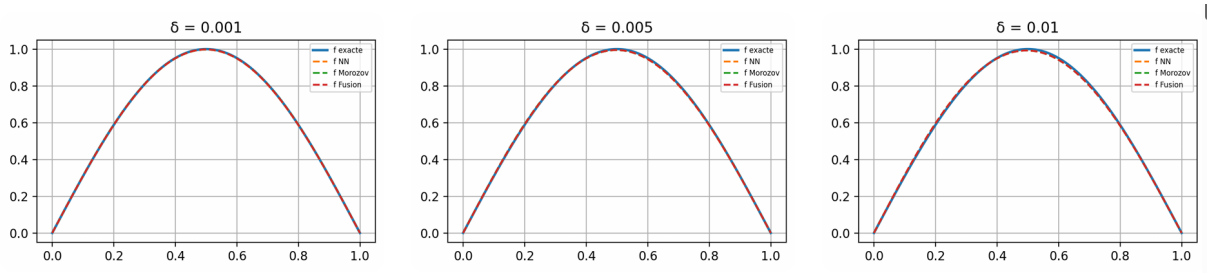


FIGURE 4.2.11 : Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}05$ et différents niveaux de bruit δ.

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.05	0.001	0.000785	0.000095	0.000440	0.00236	0.00198	0.00164
1	0.05	0.005	0.001237	0.000470	0.000854	0.00532	0.00521	0.00490
2	0.05	0.010	0.000873	0.000938	0.000905	0.00862	0.00856	0.00859

FIGURE 4.2.12 : Tableau des Résultats $T = 0{,}05$.

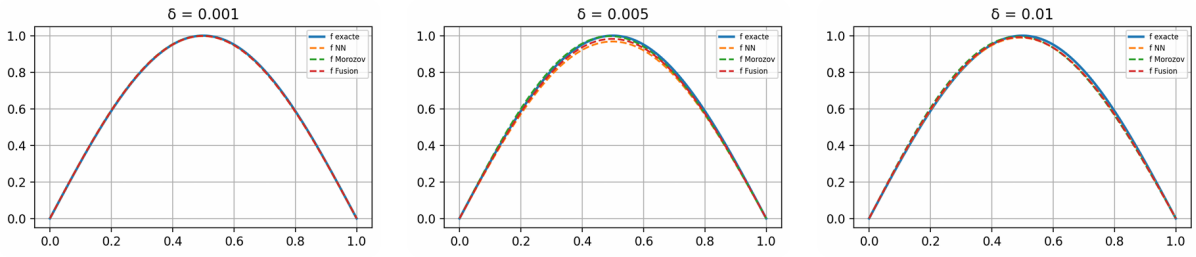


FIGURE 4.2.13 : Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}1$ et différents niveaux de bruit δ.

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.10	0.001	0.000300	0.000057	0.000178	0.00364	0.00439	0.00369
1	0.10	0.005	0.004460	0.000274	0.002367	0.03212	0.01472	0.01805
2	0.10	0.010	0.001309	0.000529	0.000919	0.01583	0.02147	0.01677

FIGURE 4.2.14 : Tableau des Résultats $T = 0{,}1$.

Remarque. *Cette méthode de fusion permet une optimisation plus rapide que la règle de Morozov classique, car elle restreint l'espace de recherche à une bande étroite centrée sur la prédiction* β_{NN}.

4.3 Discussion et analyse des performances

Le tableau suivant résume les résultats de reconstruction obtenus avec trois méthodes : prédiction du paramètre β *par réseau de neurones (*β_{NN}*), sélection via la règle de Morozov (*β_{Morozov}*) et raffinement hybride (*β_{Fusion}*). Les erreurs relatives associées sont présentées pour différentes valeurs du temps final* T *et du niveau de bruit* δ.

T	δ	$\beta_{\mathbf{NN}}$	$\beta_{\mathbf{Morozov}}$	$\beta_{\mathbf{Fusion}}$	**Erreur NN**	**Erreur Morozov**	**Erreur Fusion**
0,01	0,001	0,00100	0,001464	0,000190	0,00250	0,00449	0,00259
	0,005	0,00500	0,003540	0,000884	0,00650	0,01141	0,00706
	0,010	0,01000	0,002298	0,001690	0,01286	0,01553	0,01402
0,05	0,001	0,00100	0,000785	0,000095	0,00236	0,00198	0,00164
	0,005	0,00500	0,001237	0,000470	0,00532	0,00521	0,00490
	0,010	0,01000	0,000873	0,000938	0,00862	0,00856	0,00859
0,1	0,001	0,00100	0,000300	0,000057	0,00364	0,00439	0,00369
	0,005	0,00500	0,004460	0,000274	0,03212	0,01472	0,01805
	0,010	0,01000	0,001309	0,000529	0,01583	0,02147	0,01677

TABLE 4.3.1 : Comparaison des valeurs de β et des erreurs relatives pour différentes valeurs de T et δ

Cas $T = 0{,}01$ ***:***

- *Pour* $\delta = 0{,}001$*, la prédiction du réseau donne la plus faible erreur (0,00250), devant la méthode hybride (0,00259), tandis que Morozov est ici moins performant (0,00449).*
- *À* $\delta = 0{,}005$*, le réseau surestime* β*, conduisant à une erreur plus faible (0,00650) que Morozov seul (0,01141), mais l'hybride améliore encore (0,00706).*
- *Pour* $\delta = 0{,}010$*, le réseau reste performant (0,01286), bien que les trois méthodes donnent des erreurs supérieures à 0,01.*

Cas $T = 0{,}05$ ***:***

- *Pour* $\delta = 0{,}001$*, la méthode hybride donne l'erreur la plus faible (0,00164), devant Morozov (0,00198), le réseau étant légèrement en retrait (0,00236).*
- *À* $\delta = 0{,}005$*, les trois méthodes donnent des résultats proches, mais la méthode hybride offre l'erreur la plus basse (0,00490).*
- *À* $\delta = 0{,}010$*, les erreurs sont quasi équivalentes pour toutes les méthodes (0,0086), témoignant d'un bon accord.*

Cas $T = 0{,}10$ ***:***

- *Pour* $\delta = 0{,}001$*, la méthode hybride est la plus précise (0,00369), suivie du réseau (0,00364), puis Morozov (0,00439).*
- *À* $\delta = 0{,}005$*, le réseau surestime fortement* β*, ce qui entraîne une erreur élevée (0,03212). Morozov réduit significativement cette erreur (0,01472), mais la méthode hybride reste compétitive (0,01805).*
- *Pour* $\delta = 0{,}010$*, le réseau reste efficace (0,01583), tandis que Morozov et la méthode hybride donnent des erreurs légèrement supérieures (0,02147 et 0,01677).*

Ces résultats montrent que :

- *Le réseau* `HybridNet` *fournit généralement une estimation rapide et raisonnablement précise du paramètre* β*, notamment pour les faibles niveaux de bruit.*
- *La méthode de Morozov peut parfois produire de meilleures reconstructions, mais au prix d'un calcul plus coûteux.*

— *La méthode hybride β_{Fusion} combine les forces des deux approches, permettant d'améliorer les erreurs dans la majorité des cas.*

Conclusion générale

Dans ce travail, nous avons étudié les problèmes inverses mal posés associés à l'équation de la chaleur, en mettant l'accent sur la méthode de régularisation de Tikhonov et sur l'importance cruciale du choix du paramètre de régularisation. Deux approches ont été comparées : une méthode classique basée sur la règle de Morozov, et une approche moderne exploitant l'intelligence artificielle à l'aide de réseaux de neurones capables de prédire automatiquement un paramètre optimal.

Les problèmes inverses interviennent dans de nombreux domaines scientifiques et techniques — tels que la médecine, la géophysique ou l'imagerie industrielle — où l'on cherche à reconstituer une information inaccessible à partir de données incomplètes et bruitées. Leur étude est donc indispensable pour garantir des reconstructions fiables et exploitables.

Ce travail met en lumière l'intérêt de combiner des méthodes analytiques rigoureuses avec des techniques d'apprentissage automatique, afin de concevoir des stratégies de régularisation plus adaptatives, robustes et efficaces. Les résultats obtenus montrent que l'IA peut offrir une alternative performante, notamment dans des situations fortement bruitées où les approches classiques atteignent leurs limites.

Annexe 1 : Interface Morozov

Annexe – Implémentation Streamlit : Sélection du paramètre β par la règle de Morozov

Le code ci-dessous implémente une application Streamlit permettant de reconstruire la condition initiale d'un problème inverse de la chaleur à partir de données bruitées, en sélectionnant automatiquement le paramètre de régularisation β selon la règle de Morozov. Cette interface permet de visualiser la solution exacte, la reconstruction obtenue, et l'erreur relative associée, pour différents niveaux de bruit δ et un temps final T choisi.

Listing 4.10 : Interface Streamlit pour la méthode de Morozov

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st
import pandas as pd
from scipy.optimize import minimize_scalar

def compute_hn(h, M, x, L):
    return np.array([2 / L * np.trapz(h * np.sin(n * np.pi * x / L), x) for
        n in range(1, M + 1)])
```

```
9
10 def compute_reconstruction(beta, h_n, x, L, T):
11     recon = np.zeros_like(x)
12     for n in range(1, len(h_n) + 1):
13         lambda_n = (n * np.pi / L) ** 2
14         sigma_n = np.exp(-lambda_n * T)
15         recon += (sigma_n / (sigma_n ** 2 + beta)) * h_n[n - 1] * np.sin(n
               * np.pi * x / L)
16     return recon
17
18 def auto_find_beta_morozov(h_n, h_noisy, x, L, T, delta):
19     def residu(beta):
20         Af_val = np.zeros_like(x)
21         for n in range(1, len(h_n) + 1):
22             lambda_n = (n * np.pi / L) ** 2
23             sigma_n = np.exp(-lambda_n * T)
24             coeff = (sigma_n / (sigma_n ** 2 + beta)) * sigma_n
25             Af_val += coeff * h_n[n - 1] * np.sin(n * np.pi * x / L)
26         return abs(np.sqrt(np.trapz((Af_val - h_noisy) ** 2, x)) - delta)
27
28     beta_candidates = np.logspace(-10, 0, 100)
29     residuals = [residu(beta) for beta in beta_candidates]
30     min_idx = np.argmin(residuals)
31     i_low = max(min_idx - 5, 0)
32     i_high = min(min_idx + 5, len(beta_candidates) - 1)
33     lower = beta_candidates[i_low]
34     upper = beta_candidates[i_high]
35     result = minimize_scalar(residu, bounds=(lower, upper), method='bounded
           ', options={'xatol': 1e-12})
36     return result.x
37
38 st.title("Choix du Paramètre de Régularisation β par la Règle de Morozov")
39
40 L = 1.0
41 M = 50
```

```
x = np.linspace(0, L, 500)
f_exact = np.sin(np.pi * x)
T = st.sidebar.selectbox("Choisir une valeur de temps (T)", [0.01, 0.05,
    0.1])
deltas = [0.001, 0.005, 0.01]

st.subheader(fr"Comparaison des reconstructions pour $T = {T}$")

cols = st.columns(len(deltas))
h_exact = np.exp(-np.pi ** 2 * T) * np.sin(np.pi * x)

for i, delta in enumerate(deltas):
    np.random.seed(0)
    noise = delta * np.random.randn(len(x))
    h_noisy = h_exact + noise
    h_n = compute_hn(h_noisy, M, x, L)
    beta_opt = auto_find_beta_morozov(h_n, h_noisy, x, L, T, delta)
    f_reg_opt = compute_reconstruction(beta_opt, h_n, x, L, T)

    with cols[i]:
        fig, ax = plt.subplots(figsize=(4, 3))
        ax.plot(x, f_exact, 'k-', linewidth=2, label='f exacte')
        ax.plot(x, f_reg_opt, 'r--', linewidth=2, label='f reconstruite')
        ax.set_title(fr"$\delta$ = {delta}")
        ax.legend(fontsize=8)
        ax.grid(True)
        st.pyplot(fig)

st.subheader("Erreur relative |f - f_reg|/|f|")

cols_error = st.columns(3)

for i, delta in enumerate(deltas):
    h_exact = np.exp(-np.pi ** 2 * T) * np.sin(np.pi * x)
    np.random.seed(0)
```

```
    noise = delta * np.random.randn(len(x))
    h_noisy = h_exact + noise
    h_n = compute_hn(h_noisy, M, x, L)
    beta_opt = auto_find_beta_morozov(h_n, h_noisy, x, L, T, delta)
    f_reg_opt = compute_reconstruction(beta_opt, h_n, x, L, T)

    erreur_relative = np.abs(f_exact - f_reg_opt) / np.abs(f_exact)
    erreur_relative[np.isinf(erreur_relative)] = 0

    with cols_error[i]:
        fig_err, ax_err = plt.subplots(figsize=(4, 3))
        ax_err.plot(x, erreur_relative, 'b-', linewidth=2)
        ax_err.set_title(f"    = {delta}")
        ax_err.set_ylim(0, 1)
        ax_err.grid()
        st.pyplot(fig_err)

st.subheader(f"Tableau des résultats ")

results = []
h_exact_T = np.exp(-np.pi ** 2 * T) * np.sin(np.pi * x)

for delta in deltas:
    np.random.seed(0)
    noise = delta * np.random.randn(len(x))
    h_noisy = h_exact_T + noise
    h_n = compute_hn(h_noisy, M, x, L)

    beta_opt = auto_find_beta_morozov(h_n, h_noisy, x, L, T, delta)

    f_reg_opt = compute_reconstruction(beta_opt, h_n, x, L, T)
    erreur_relative = (
        np.sqrt(np.trapz((f_exact - f_reg_opt) ** 2, x))
        / np.sqrt(np.trapz(f_exact ** 2, x))
    )
```

```
111
112     results.append({
113         "T": f"{T:.3f}".replace('.', ','),
114         "  ": f"{delta:.5f}".replace('.', ','),
115         "β choisi": f"{beta_opt:.5f}".replace('.', ','),
116         "Erreur relative": f"{erreur_relative:.5f}".replace('.', ',')
117     })
118
119 df_results = pd.DataFrame(results)
120
121 st.table(df_results.reset_index(drop=True))
122
123 csv = df_results.to_csv(index=False, sep=';').encode('utf-8')
124 st.download_button(
125     label="Télécharger le tableau en CSV",
126     data=csv,
127     file_name=f'resultats_T_{T}.csv',
128     mime='text/csv',
129 )
```

Résultats visuels de l'interface

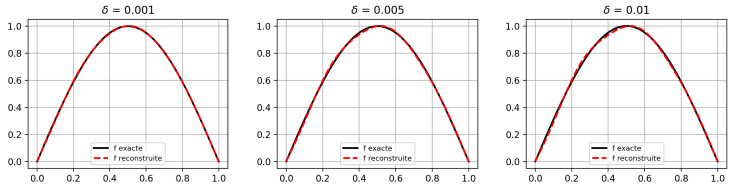
Les figures suivantes sont générées à partir de l'application interactive développée avec `Streamlit`. *Elles illustrent la qualité de la reconstruction obtenue par la régularisation de Tikhonov, avec un paramètre β sélectionné selon la règle de Morozov. Pour chaque valeur du paramètre temporel T considérée, on visualise l'effet de différents niveaux de bruit δ sur l'écart entre la fonction exacte f et la solution reconstruite f_β. Ces résultats permettent d'évaluer empiriquement la stabilité et l'efficacité de la méthode utilisée.*

Choisir une valeur de temps (T)

0.01

Choix du Paramètre de Régularisation β par la Règle de Morozov

Comparaison des reconstructions pour $T = 0.01$



Erreur relative |f - f_reg|/|f|

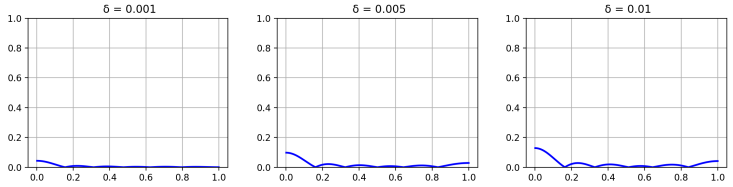


Tableau des résultats

	T	δ	β choisi	Erreur relative
0	0,010	0,00100	0,00019	0,00449
1	0,010	0,00500	0,00088	0,01141
2	0,010	0,01000	0,00169	0,01553

Télécharger le tableau en CSV

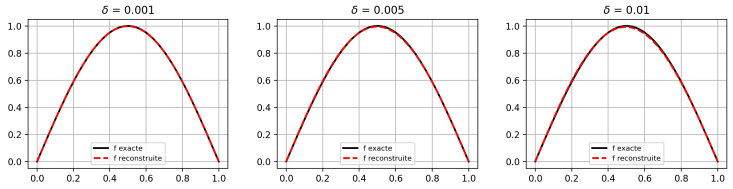
Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}01$

Choisir une valeur de temps (T)

0.05

Choix du Paramètre de Régularisation β par la Règle de Morozov

Comparaison des reconstructions pour $T = 0.05$



Erreur relative |f - f_reg|/|f|

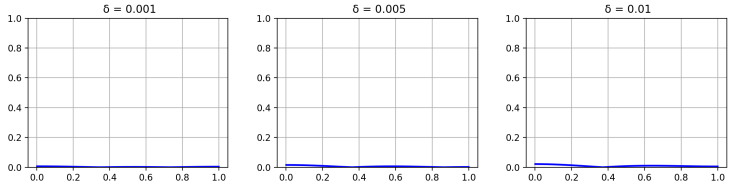


Tableau des résultats

	T	δ	β choisi	Erreur relative
0	0,050	0,00100	0,00010	0,00198
1	0,050	0,00500	0,00047	0,00521
2	0,050	0,01000	0,00094	0,00856

Télécharger le tableau en CSV

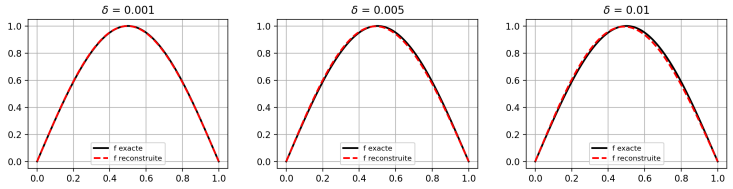
Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}05$

Choisir une valeur de temps (T)

0.1

Choix du Paramètre de Régularisation β par la Règle de Morozov

Comparaison des reconstructions pour $T = 0.1$



Erreur relative |f - f_reg|/|f|

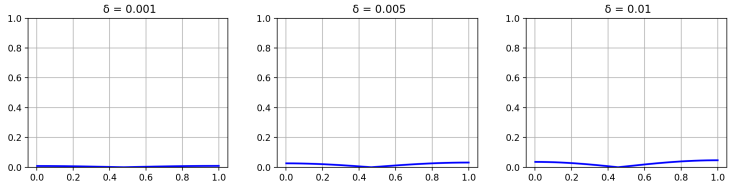


Tableau des résultats

	T	δ	β choisi	Erreur relative
0	0,100	0,00100	0,00006	0,00439
1	0,100	0,00500	0,00027	0,01472
2	0,100	0,01000	0,00053	0,02147

Télécharger le tableau en CSV

Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}1$

Annexe 2 : Interface Réseau de neurones et Morozov

Annexe – Implémentation Streamlit : Sélection du paramètre β par `HybridNet` et la méthode de Morozov

Ce code constitue le cœur de notre application développée en `Streamlit`*. Il implémente une approche de régularisation pour les problèmes inverses, en combinant l'apprentissage automatique (via un réseau de neurones hybride nommé* `HybridNet`*) et une correction analytique par la règle de Morozov. La prédiction initiale du paramètre β par le réseau est affinée par une méthode de fusion pondérée avec $\beta_{Morozov}$, permettant une reconstruction stable et précise. L'interface permet de visualiser les reconstructions obtenues et de comparer les performances des différentes méthodes.*

Listing 4.11 : Code Python complet pour l'application Streamlit combinant HybridNet et Morozov

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
os.environ["TORCH_ALLOW_TF32_CUBLAS_OVERRIDE"] = "0"

import numpy as np
import torch
```

```
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from scipy.optimize import minimize_scalar
import streamlit as st
import matplotlib.pyplot as plt
import pandas as pd
L = 1.0
M = 50
x = np.linspace(0, L, 500)
f_exact = np.sin(np.pi * x)

class HybridNet(nn.Module):
    def __init__(self, M):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(1, 64, kernel_size=5, padding=2),
            nn.BatchNorm1d(64),
            nn.GELU(),
            nn.MaxPool1d(2),
            nn.Conv1d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.AdaptiveAvgPool1d(1)
        )
        self.mlp = nn.Sequential(
            nn.Linear(2, 64),
            nn.LayerNorm(64),
            nn.GELU(),
            nn.Dropout(0.3)
        )
        self.head = nn.Sequential(
            nn.Linear(128 + 64, 256),
            nn.LayerNorm(256),
```

```
            nn.GELU(),
            nn.Dropout(0.4),
            nn.Linear(256, 1),
            nn.Softplus()
        )

    def forward(self, params, signal):
        signal = signal.unsqueeze(1)
        cnn_features = self.cnn(signal).squeeze(-1)
        mlp_features = self.mlp(params)
        combined = torch.cat([cnn_features, mlp_features], dim=1)
        return self.head(combined)

def compute_hn(h, M, x):
    return np.array([2 / L * np.trapz(h * np.sin(n * np.pi * x / L), x) for
        n in range(1, M + 1)])

def reconstruction(beta, h_n, x, L, T):
    recon = np.zeros_like(x)
    for n in range(1, len(h_n) + 1):
        lambda_n = (n * np.pi / L) ** 2
        sigma_n = np.exp(-lambda_n * T)
        recon += (sigma_n / (sigma_n ** 2 + beta + 1e-12)) * h_n[n - 1] *
            np.sin(n * np.pi * x / L)
    return recon

def auto_find_beta_morozov(h_n, h_noisy, x, L, T, delta):
    def residu(beta):
        Af_val = np.zeros_like(x)
        for n in range(1, len(h_n) + 1):
            lambda_n = (n * np.pi / L) ** 2
            sigma_n = np.exp(-lambda_n * T)
            coeff = (sigma_n / (sigma_n ** 2 + beta)) * sigma_n
            Af_val += coeff * h_n[n - 1] * np.sin(n * np.pi * x / L)
        return abs(np.sqrt(np.trapz((Af_val - h_noisy) ** 2, x)) - delta)
```

```
75
76      beta_candidates = np.logspace(-10, 0, 100)
77      residuals = [residu(beta) for beta in beta_candidates]
78      min_idx = np.argmin(residuals)
79      i_low = max(min_idx - 5, 0)
80      i_high = min(min_idx + 5, len(beta_candidates) - 1)
81      lower = beta_candidates[i_low]
82      upper = beta_candidates[i_high]
83      result = minimize_scalar(residu, bounds=(lower, upper), method='bounded
            ', options={'xatol': 1e-12})
84      return result.x
85
86  def generate_data(T_list, delta_list, N_samples=200):
87      inputs, signals, targets = [], [], []
88      for T in T_list:
89          for delta in delta_list:
90              for _ in range(N_samples):
91                  h = np.exp(-np.pi ** 2 * T) * np.sin(np.pi * x)
92                  noise = delta * np.random.randn(len(x))
93                  h_noisy = h + noise
94                  h_n = compute_hn(h_noisy, M, x)
95                  res = minimize_scalar(
96                      lambda b: np.linalg.norm(reconstruction(b, h_n, x, L, T
                            ) - f_exact),
97                      bounds=(1e-10, 1), method='bounded', tol=1e-12
98                  )
99                  inputs.append([T, delta])
100                 signals.append(h_n)
101                 targets.append([res.x])
102     return np.array(inputs), np.array(signals), np.array(targets)
103
104 def train_model(model, inputs, signals, targets, epochs=500):
105     input_scaler = StandardScaler()
106     signal_scaler = StandardScaler()
107     inputs_norm = input_scaler.fit_transform(inputs)
```

```
    signals_norm = signal_scaler.fit_transform(signals)
    inputs_t = torch.tensor(inputs_norm, dtype=torch.float32)
    signals_t = torch.tensor(signals_norm, dtype=torch.float32)
    targets_t = torch.tensor(targets, dtype=torch.float32)
    dataset = TensorDataset(inputs_t, signals_t, targets_t)
    loader = DataLoader(dataset, batch_size=64, shuffle=True)
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-5)
    scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=1e-3,
        steps_per_epoch=len(loader), epochs=epochs)
    loss_fn = nn.HuberLoss()
    for epoch in range(epochs):
        model.train()
        for batch_inputs, batch_signals, batch_targets in loader:
            optimizer.zero_grad()
            preds = model(batch_inputs, batch_signals)
            loss = loss_fn(preds, batch_targets)
            loss.backward()
            optimizer.step()
            scheduler.step()
    return input_scaler, signal_scaler

def main():
    st.title("Fusion Réseau de Neurones + Règle de Morozov pour un β
        Optimal")
    T_list = [0.01, 0.05, 0.1]
    delta_list = [0.001, 0.005, 0.01]
    inputs, signals, targets = generate_data(T_list, delta_list, N_samples
        =30)
    model = HybridNet(M)
    input_scaler, signal_scaler = train_model(model, inputs, signals,
        targets, epochs=1000)
    T_user = st.sidebar.selectbox("Choisir T", T_list)
    alpha = st.sidebar.slider("Poids du réseau neuronal (  )", 0.0, 1.0,
        0.5, 0.05)
    h_exact = np.exp(-np.pi**2 * T_user) * np.sin(np.pi * x)
```

```
    results = []
    st.subheader(f"Comparaison des reconstructions pour T = {T_user}")
    cols = st.columns(len(delta_list))
    for i, delta in enumerate(delta_list):
        np.random.seed(0)
        noise = delta * np.random.randn(len(x))
        h_noisy = h_exact + noise
        h_n_user = compute_hn(h_noisy, M, x)
        input_norm = input_scaler.transform([[T_user, delta]])
        signal_norm = signal_scaler.transform([h_n_user])
        with torch.no_grad():
            beta_nn = model(torch.tensor(input_norm, dtype=torch.float32),
                torch.tensor(signal_norm, dtype=torch.float32)).item()
        beta_morozov = auto_find_beta_morozov(h_n_user, h_noisy, x, L,
            T_user, delta)
        beta_fus = alpha * beta_nn + (1 - alpha) * beta_morozov
        f_nn = reconstruction(beta_nn, h_n_user, x, L, T_user)
        f_mor = reconstruction(beta_morozov, h_n_user, x, L, T_user)
        f_fus = reconstruction(beta_fus, h_n_user, x, L, T_user)
        err_nn = np.linalg.norm(f_exact - f_nn) / np.linalg.norm(f_exact)
        err_mor = np.linalg.norm(f_exact - f_mor) / np.linalg.norm(f_exact)
        err_fus = np.linalg.norm(f_exact - f_fus) / np.linalg.norm(f_exact)
        results.append([f"{T_user:.2f}", f"{delta:.3f}", f"{beta_nn:.6f}",
            f"{beta_morozov:.6f}", f"{beta_fus:.6f}", f"{err_nn:.5f}", f"{
            err_mor:.5f}", f"{err_fus:.5f}"])
        with cols[i]:
            fig, ax = plt.subplots(figsize=(5, 3))
            ax.plot(x, f_exact, label="f exacte", lw=2)
            ax.plot(x, f_nn, '--', label="f NN")
            ax.plot(x, f_mor, '--', label="f Morozov")
            ax.plot(x, f_fus, '--', label="f Fusion")
            ax.set_title(f"   = {delta}")
            ax.legend(fontsize=6)
            ax.grid(True)
            st.pyplot(fig)
```

```
    st.subheader("Erreur relative |f - f_reg| / |f|")
    cols_error = st.columns(len(delta_list))
    for i, delta in enumerate(delta_list):
        np.random.seed(0)
        noise = delta * np.random.randn(len(x))
        h_noisy = h_exact + noise
        h_n_user = compute_hn(h_noisy, M, x)
        input_norm = input_scaler.transform([[T_user, delta]])
        signal_norm = signal_scaler.transform([h_n_user])
        with torch.no_grad():
            beta_nn = model(torch.tensor(input_norm, dtype=torch.float32),
                torch.tensor(signal_norm, dtype=torch.float32)).item()
        beta_morozov = auto_find_beta_morozov(h_n_user, h_noisy, x, L,
            T_user, delta)
        beta_fus = alpha * beta_nn + (1 - alpha) * beta_morozov
        f_fus = reconstruction(beta_fus, h_n_user, x, L, T_user)
        erreur_relative = np.abs(f_exact - f_fus) / (np.abs(f_exact) + 1e
            -12)
        erreur_relative[np.isinf(erreur_relative)] = 0
        with cols_error[i]:
            fig_err, ax_err = plt.subplots(figsize=(5, 3))
            ax_err.plot(x, erreur_relative, 'b-', linewidth=2)
            ax_err.set_title(f"    = {delta}")
            ax_err.set_ylabel("|f - f_reg| / |f|")
            ax_err.set_ylim(0, 1)
            ax_err.grid(True)
            st.pyplot(fig_err)
    st.subheader("Tableau des résultats")
    df_results = pd.DataFrame(results, columns=["T", "  ", "β_NN", "β
        _Morozov", "β_Fusion", "Erreur NN", "Erreur Morozov", "Erreur Fusion
        "])
    st.table(df_results)
    csv = df_results.to_csv(index=False, sep=';').encode('utf-8')
    st.download_button(label="Télécharger le tableau en CSV", data=csv,
        file_name=f'resultats_T_{T_user}.csv', mime='text/csv')
```

```
198
199 if __name__ == "__main__":
200     main()
```

Résultats visuels de l'interface

Les visualisations suivantes sont issues du tableau de bord interactif construit avec `Streamlit`. *Elles comparent la fonction exacte f et les reconstructions f_β obtenues pour différentes configurations de (T, δ), en utilisant la prédiction du paramètre β par le réseau de neurones.*

Choisir T

0.01

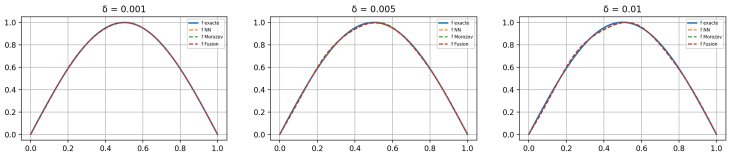
Poids du réseau neuronal (α)

0.50

0.00 1.00

Fusion Réseau de Neurones + Règle de Morozov pour un β Optimal

Comparaison des reconstructions pour T = 0.01



Erreur relative |f - f_reg| / |f|

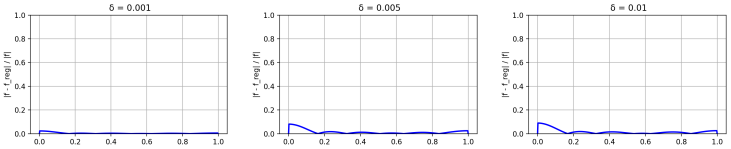


Tableau des résultats

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.01	0.001	0.001464	0.000190	0.000827	0.00250	0.00449	0.00259
1	0.01	0.005	0.003540	0.000884	0.002212	0.00650	0.01141	0.00706
2	0.01	0.010	0.002298	0.001690	0.001994	0.01286	0.01553	0.01402

Télécharger le tableau en CSV

Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}01$

Choisir T

0.05

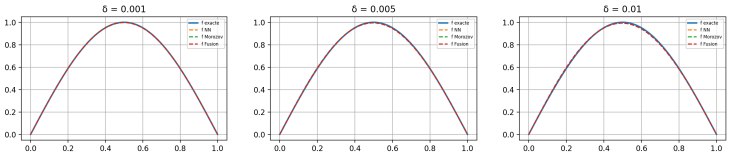
Poids du réseau neuronal (α)

0.50

0.00 1.00

Fusion Réseau de Neurones + Règle de Morozov pour un β Optimal

Comparaison des reconstructions pour T = 0.05



Erreur relative |f - f_reg| / |f|

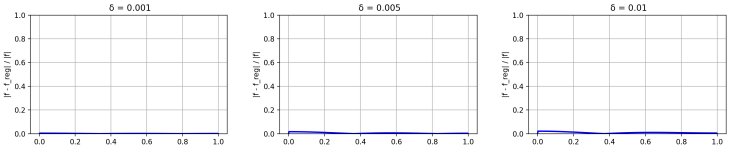


Tableau des résultats

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.05	0.001	0.000785	0.000095	0.000440	0.00236	0.00198	0.00164
1	0.05	0.005	0.001237	0.000470	0.000854	0.00532	0.00521	0.00490
2	0.05	0.010	0.000873	0.000938	0.000905	0.00862	0.00856	0.00859

Télécharger le tableau en CSV

Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}05$.

Choisir T

0.1

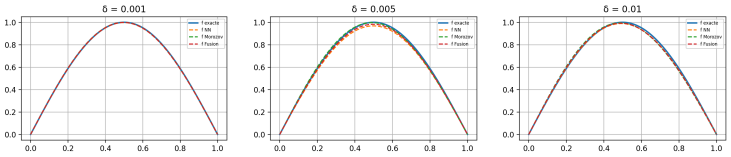
Poids du réseau neuronal (α)

0.50

0.00 1.00

Fusion Réseau de Neurones + Règle de Morozov pour un β Optimal

Comparaison des reconstructions pour T = 0.1



Erreur relative |f - f_reg| / |f|

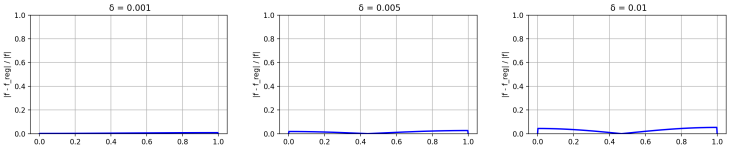


Tableau des résultats

	T	δ	β_NN	β_Morozov	β_Fusion	Erreur NN	Erreur Morozov	Erreur Fusion
0	0.10	0.001	0.000300	0.000057	0.000178	0.00364	0.00439	0.00369
1	0.10	0.005	0.004460	0.000274	0.002367	0.03212	0.01472	0.01805
2	0.10	0.010	0.001309	0.000529	0.000919	0.01583	0.02147	0.01677

Télécharger le tableau en CSV

Comparaison entre la fonction exacte f et la reconstruction f_β pour $T = 0{,}1$.

Bibliographie

[1] *Bertero, M., & Boccacci, P. (1998). Introduction to Inverse Problems in Imaging. IOP Publishing.*

[2] *Bécache, E., & Bonnet, M. (2010). Problèmes inverses en diffusion thermique. Techniques de l'Ingénieur.* https://www.techniques-ingenieur.fr

[3] *Benabdellah, A. (2021). Problème inverse pour une équation de chaleur non locale avec conditions aux limites de type périodique. Mémoire de Master, Université Mohamed Boudiaf - M'Sila.* http://repository.univ-msila.dz

[4] *Brézis, H. (2011). Analyse Fonctionnelle, Théorie et Applications. Masson, Paris. (Traduction anglaise : Functional Analysis)*

[5] *Chikhi, L. (2017). Problèmes inverses de l'équation de transfert de la chaleur. Mémoire de Magister, USTHB.* https://www.theses-algerie.com

[6] *Engl, H. W., Hanke, M., & Neubauer, A. (1996). Regularization of Inverse Problems. Springer.*

[7] *Groetsch, C. W. (2007). Integral Equations of the First Kind, Inverse Problems and Regularization. Chapman and Hall/CRC.*

[8] *Hansen, P. C. (1998). Rank-Deficient and Discrete Ill-Posed Problems : Numerical Aspects of Linear Inversion. SIAM.*

[9] *Hansen, P. C. (2010). Discrete Inverse Problems : Insight and Algorithms. SIAM.*

[10] Jin, B., & Rundell, W. (2015). A tutorial on inverse problems for anomalous diffusion processes. Inverse Problems.

[11] Kabanikhin, S. I. (2008). Definitions and examples of inverse and ill-posed problems. Journal of Inverse and Ill-Posed Problems.

[12] Kebaili, N. (2020). Problème inverse et méthodes de régularisation. Mémoire de Master, Université Abderrahmane Mira – Béjaïa. http://univ-bejaia.dz

[13] Kirsch, A. (2011). An Introduction to the Mathematical Theory of Inverse Problems (Vol. 120). Springer.

[14] Lattès, R., & Lions, J.-L. (1969). Méthode de quasi-réversibilité et applications. Dunod.

[15] Lions, J.-L., & Magenes, E. (1972). Problèmes aux limites non homogènes et applications (Vols. 1 et 2). Dunod.

[16] Natterer, F., & Wübbeling, F. (2001). Mathematical Methods in Image Reconstruction. SIAM.

[17] Renardy, M., & Rogers, R. C. (2006). An Introduction to Partial Differential Equations (Vol. 13). Springer.

[18] Tarantola, A. (2005). Inverse Problem Theory and Methods for Model Parameter Estimation. SIAM.

[19] Tikhonov, A. N., & Arsenin, V. Y. (1977). Solutions of Ill-Posed Problems. Winston & Sons.

[20] Trenogin, V. A., Pissarevski, B., & Soboleva, T. S. (1987). Problèmes et exercices d'analyse fonctionnelle.