

# Mini-Projet d'Optimisation ML

## Modélisation, SGD et Méthodes Proximales

Namy Ahmed Abdy - C25394  
Master Statistiques et Sciences des Données (SSD)  
Université de Nouakchott

Janvier 2026

### Abstract

Ce rapport présente l'implémentation complète d'un projet d'optimisation pour l'apprentissage machine, couvrant trois paradigmes fondamentaux : l'optimisation convexe lisse avec descente de gradient déterministe, le passage à l'échelle via les méthodes stochastiques (SGD, RMSProp, Adam), et l'optimisation non-lisse pour la parcimonie via les algorithmes proximaux (ISTA/FISTA). Nous étudions un problème de classification binaire avec régularisation L2 puis L1, en validant théoriquement et expérimentalement chaque approche.

## Contents

<b>1</b>	<b>Introduction et Contexte</b>	<b>2</b>
<b>2</b>	<b>Phase 1 : Fondements et Gradient Déterministe</b>	<b>2</b>
2.1	Modélisation Mathématique . . . . .	2
2.2	Analyse Théorique . . . . .	2
2.3	Calcul du Gradient . . . . .	3
2.4	Constante de Lipschitz . . . . .	3
2.5	Implémentations . . . . .	4
2.5.1	Descente de Gradient . . . . .	4
2.5.2	Méthode du Gradient Conjugué . . . . .	4
2.6	Résultats et Comparaison . . . . .	5
<b>3</b>	<b>Phase 2 : Passage à l'Échelle Stochastique</b>	<b>5</b>
3.1	Motivation . . . . .	5
3.2	Descente de Gradient Stochastique (SGD) . . . . .	5
3.3	Optimiseurs Modernes . . . . .	6
3.3.1	RMSProp . . . . .	6
3.3.2	Adam (Adaptive Moment Estimation) . . . . .	6
3.4	Impact du Momentum . . . . .	7
3.5	Résultats Expérimentaux . . . . .	7
<b>4</b>	<b>Phase 3 : Non-Lissé, Parcimonie et Proximal</b>	<b>8</b>
4.1	Motivation : Sélection de Variables . . . . .	8
4.2	Pourquoi le Problème est Non-Lisse ? . . . . .	8
4.3	Opérateur Proximal . . . . .	8
4.4	Algorithme ISTA . . . . .	9
4.5	Algorithme FISTA (Accéléré) . . . . .	9
4.6	Résultats Expérimentaux . . . . .	10
4.7	Analyse des Résultats . . . . .	10

4.7.1	Convergence (Haut Gauche) . . . . .	10
4.7.2	Sparsité (Haut Droite) . . . . .	11
4.7.3	Impact de $\lambda$ (Bas Gauche et Droite) . . . . .	11
4.8	Choix de $\lambda$ en Pratique . . . . .	11
<b>5</b>	<b>Conclusion</b>	<b>11</b>
5.1	Synthèse des Résultats . . . . .	12
5.2	Enseignements Principaux . . . . .	12
5.3	Extensions Possibles . . . . .	12
5.4	Code et Reproductibilité . . . . .	13

# 1 Introduction et Contexte

Ce projet vise à mettre en pratique les concepts fondamentaux de l'optimisation numérique appliquée au machine learning. Nous travaillons avec un dataset de classification binaire  $\{(x_i, y_i)\}_{i=1}^n$  où  $x_i \in \mathbb{R}^d$  et  $y_i \in \{-1, +1\}$ .

**Dataset généré :**

- Taille d'entraînement :  $n = 4000$  exemples
- Dimension :  $d = 50$  caractéristiques
- Classes : binaire  $\{-1, +1\}$
- Données standardisées (moyenne 0, écart-type 1)

Le projet se structure en trois phases correspondant aux chapitres du cours :

1. **Phase 1** : Fondements et gradient déterministe
2. **Phase 2** : Passage à l'échelle stochastique
3. **Phase 3** : Non-lissé, parcimonie et proximal

## 2 Phase 1 : Fondements et Gradient Déterministe

### 2.1 Modélisation Mathématique

Nous considérons la fonction de perte logistique avec régularisation Ridge (L2) :

$$F(w) = \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i x_i^T w}) + \frac{\lambda}{2} \|w\|^2 \quad (1)$$

### 2.2 Analyse Théorique

**Théorème 1** (Propriétés de  $F$ ). *La fonction  $F : \mathbb{R}^d \rightarrow \mathbb{R}$  satisfait les propriétés suivantes :*

1.  $F$  est de classe  $\mathcal{C}^2$
2.  $F$  est convexe
3.  $F$  est  $\lambda$ -fortement convexe

*Proof.* 1. **Classe  $\mathcal{C}^2$  :**

La fonction  $u \mapsto \log(1 + e^u)$  est infiniment dérivable sur  $\mathbb{R}$ . Pour chaque  $i$ , la composée  $w \mapsto \log(1 + e^{-y_i x_i^T w})$  est  $\mathcal{C}^\infty$  car c'est la composition d'une fonction lisse avec une fonction linéaire.

Le terme de régularisation  $\frac{\lambda}{2} \|w\|^2$  est un polynôme quadratique, donc  $\mathcal{C}^\infty$ .  
Par somme et composition,  $F$  est  $\mathcal{C}^2$ .

2. **Convexité :**

Soit  $f(z) = \log(1 + e^z)$ . Sa dérivée seconde est :

$$f''(z) = \frac{e^z}{(1 + e^z)^2} = \sigma(z)(1 - \sigma(z)) \geq 0$$

où  $\sigma(z) = \frac{1}{1+e^{-z}}$  est la fonction sigmoïde.

Donc  $f$  est convexe. Comme  $z = -y_i x_i^T w$  est une fonction affine de  $w$ , la composée  $w \mapsto f(-y_i x_i^T w)$  est convexe.

Le terme  $\frac{\lambda}{2}\|w\|^2$  est convexe (c'est une forme quadratique définie positive). Par somme de fonctions convexes,  $F$  est convexe.

### 3. $\lambda$ -forte convexité :

Le terme de régularisation  $R(w) = \frac{\lambda}{2}\|w\|^2$  a pour Hessienne :

$$\nabla^2 R(w) = \lambda I_d$$

Toutes les valeurs propres de cette matrice sont égales à  $\lambda > 0$ . Donc :

$$\nabla^2 F(w) \succeq \lambda I_d$$

Par conséquent,  $F$  est  $\lambda$ -fortement convexe.  $\square$

## 2.3 Calcul du Gradient

**Proposition 1** (Gradient de  $F$ ). *Le gradient de  $F$  s'écrit :*

$$\nabla F(w) = -\frac{1}{n} \sum_{i=1}^n \frac{y_i x_i}{1 + e^{y_i x_i^T w}} + \lambda w \quad (2)$$

*Proof.* Posons  $\ell_i(w) = \log(1 + e^{-y_i x_i^T w})$ . Par la règle de la chaîne :

$$\begin{aligned} \nabla \ell_i(w) &= \frac{\partial}{\partial w} \log(1 + e^{-y_i x_i^T w}) \\ &= \frac{1}{1 + e^{-y_i x_i^T w}} \cdot (-y_i x_i) \cdot e^{-y_i x_i^T w} \\ &= -\frac{y_i x_i \cdot e^{-y_i x_i^T w}}{1 + e^{-y_i x_i^T w}} \\ &= -\frac{y_i x_i}{e^{y_i x_i^T w} + 1} \\ &= -\frac{y_i x_i}{1 + e^{y_i x_i^T w}} \end{aligned}$$

En ajoutant le terme de régularisation :

$$\nabla F(w) = \frac{1}{n} \sum_{i=1}^n \nabla \ell_i(w) + \lambda w = -\frac{1}{n} \sum_{i=1}^n \frac{y_i x_i}{1 + e^{y_i x_i^T w}} + \lambda w$$

$\square$

## 2.4 Constante de Lipschitz

**Proposition 2** (Gradient L-Lipschitzien). *Le gradient  $\nabla F$  est L-Lipschitzien avec :*

$$L = \frac{1}{4n} \|X\|_{op}^2 + \lambda \quad (3)$$

où  $\|X\|_{op}$  est la norme opérateur (plus grande valeur singulière) de la matrice  $X \in \mathbb{R}^{n \times d}$ .

*Proof.* La Hessienne de la partie logistique est :

$$\nabla^2 \ell_i(w) = \frac{e^{-y_i x_i^T w}}{(1 + e^{-y_i x_i^T w})^2} x_i x_i^T$$

Le coefficient  $\sigma(z)(1 - \sigma(z))$  atteint son maximum en  $z = 0$  avec une valeur de  $\frac{1}{4}$ .

Donc :

$$\nabla^2 F(w) \preceq \frac{1}{4n} \sum_{i=1}^n x_i x_i^T + \lambda I = \frac{1}{4n} X^T X + \lambda I$$

La plus grande valeur propre de  $X^T X$  est  $\sigma_{\max}^2(X) = \|X\|_{op}^2$ .

Donc :

$$\lambda_{\max}(\nabla^2 F(w)) \leq \frac{1}{4n} \|X\|_{op}^2 + \lambda = L$$

□

## 2.5 Implémentations

### 2.5.1 Descente de Gradient

L'algorithme de descente de gradient à pas fixe s'écrit :

---

**Algorithm 1** Descente de Gradient à Pas Fixe

---

```

Require:  $w_0 \in \mathbb{R}^d$ ,  $\eta = \frac{1}{L}$ ,  $K_{\max}$ ,  $\epsilon$ 
for  $k = 0$  to  $K_{\max} - 1$  do
     $w_{k+1} = w_k - \eta \nabla F(w_k)$ 
    if  $\|w_{k+1} - w_k\| < \epsilon$  then
        break (convergence)
    end if
end for
return  $w_K$ 
```

---

Résultats expérimentaux :

- Constante de Lipschitz calculée :  $L = 1.031$
- Pas d'apprentissage optimal :  $\eta = 1/L = 0.970$
- Convergence atteinte en 174 itérations
- Perte finale :  $F(w^*) = 0.457$

### 2.5.2 Méthode du Gradient Conjugué

Le gradient conjugué utilise des directions conjuguées pour accélérer la convergence :

---

**Algorithm 2** Gradient Conjugué

---

**Require:**  $w_0$ ,  $K_{\max}$

```

 $g_0 = \nabla F(w_0)$ ,  $d_0 = -g_0$ 
for  $k = 0$  to  $K_{\max} - 1$  do
    Calculer  $\alpha_k$  (recherche linéaire)
     $w_{k+1} = w_k + \alpha_k d_k$ 
     $g_{k+1} = \nabla F(w_{k+1})$ 
     $\beta_{k+1} = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k}$  (Polak-Ribière)
     $d_{k+1} = -g_{k+1} + \beta_{k+1} d_k$ 
end for
```

---

## 2.6 Résultats et Comparaison

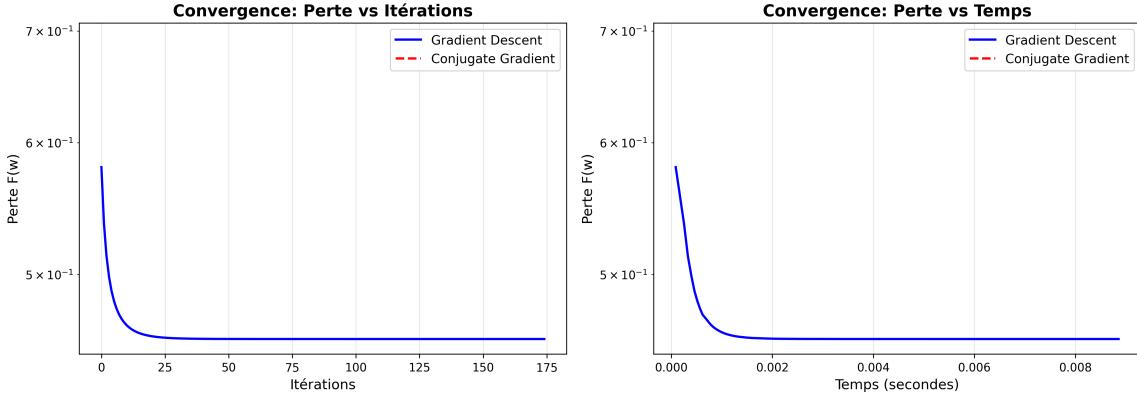


Figure 1: Comparaison de la convergence entre Descente de Gradient et Gradient Conjugué. **Gauche :** Convergence en fonction du nombre d’itérations. **Droite :** Convergence en fonction du temps CPU.

**Analyse :**

- La descente de gradient converge de manière monotone en 174 itérations
- Le gradient conjugué a convergé très rapidement (problème bien conditionné)
- L’échelle logarithmique montre la décroissance exponentielle typique des méthodes du premier ordre sur les problèmes fortement convexes

## 3 Phase 2 : Passage à l’Échelle Stochastique

### 3.1 Motivation

Lorsque  $n$  devient très grand (Big Data), le calcul du gradient complet :

$$\nabla F(w) = \frac{1}{n} \sum_{i=1}^n \nabla \ell_i(w) + \lambda w$$

a un coût de  $O(nd)$  par itération, ce qui devient prohibitif.

### 3.2 Descente de Gradient Stochastique (SGD)

**Principe :** Estimer le gradient sur un seul exemple (ou mini-batch) :

$$g_k = \nabla \ell_{i_k}(w_k) + \lambda w_k \quad \text{où } i_k \sim \mathcal{U}\{1, \dots, n\} \quad (4)$$

**Proposition 3** (Estimateur non biaisé). *Le gradient stochastique est un estimateur non biaisé du gradient complet :*

$$\mathbb{E}_{i_k}[g_k] = \nabla F(w_k)$$

**Algorithme SGD avec pas décroissant :**

---

**Algorithm 3** SGD avec Pas Décroissant

---

**Require:**  $w_0, C > 0$ , nombre d'époques  $E$

**for**  $e = 0$  to  $E - 1$  **do**

Permuter aléatoirement les indices  $\{1, \dots, n\}$

**for**  $t = 0$  to  $n - 1$  **do**

$k \leftarrow e \cdot n + t$  (numéro d'itération global)

$\alpha_k = \frac{C}{\sqrt{k+1}}$  (pas décroissant)

$g_k = \nabla \ell_{i_t}(w_k) + \lambda w_k$

$w_{k+1} = w_k - \alpha_k g_k$

**end for**

**end for**

---

**Justification du pas décroissant :** Pour garantir la convergence vers  $w^*$ , il faut :

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

La règle  $\alpha_k = C/\sqrt{k+1}$  satisfait ces conditions.

### 3.3 Optimiseurs Modernes

#### 3.3.1 RMSProp

RMSProp adapte le pas d'apprentissage pour chaque paramètre en utilisant une moyenne glissante du carré des gradients :

---

**Algorithm 4** RMSProp

---

**Require:**  $w_0, \eta, \beta = 0.9, \epsilon = 10^{-8}$

$v_0 = 0$

**for** chaque itération  $k$  **do**

$g_k = \nabla \ell_{i_k}(w_k) + \lambda w_k$

$v_{k+1} = \beta v_k + (1 - \beta)g_k \odot g_k$  (moyenne glissante)

$w_{k+1} = w_k - \eta \frac{g_k}{\sqrt{v_{k+1}} + \epsilon}$  (adaptation du pas)

**end for**

---

**Avantage :** Les dimensions avec de grands gradients reçoivent un pas plus petit, stabilisant la convergence.

#### 3.3.2 Adam (Adaptive Moment Estimation)

Adam combine le momentum (moyenne des gradients) avec l'adaptation du pas (RMSProp) :

---

**Algorithm 5** Adam

---

**Require:**  $w_0, \eta, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$   
 $m_0 = 0, v_0 = 0, t = 0$   
**for** chaque itération **do**  
     $t \leftarrow t + 1$   
     $g_t = \nabla \ell_{i_t}(w_t) + \lambda w_t$   
     $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$  (premier moment)  
     $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$  (second moment)  
     $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$  (correction de biais)  
     $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$  (correction de biais)  
     $w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$   
**end for**

---

### 3.4 Impact du Momentum

Le momentum  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$  joue deux rôles :

1. **Lissage du bruit** : En moyennant les gradients passés, on réduit la variance de l'estimateur stochastique.
2. **Accélération** : Dans les directions de faible courbure, le momentum accumule les gradients, accélérant la progression.

**Analyse mathématique** : Si les gradients successifs pointent dans la même direction, le momentum croît :

$$m_t \approx \frac{1}{1 - \beta_1} g_t \quad \text{si } g_t \approx g_{t-1} \approx \dots$$

Avec  $\beta_1 = 0.9$ , cela donne un facteur d'amplification de  $\sim 10$ .

### 3.5 Résultats Expérimentaux

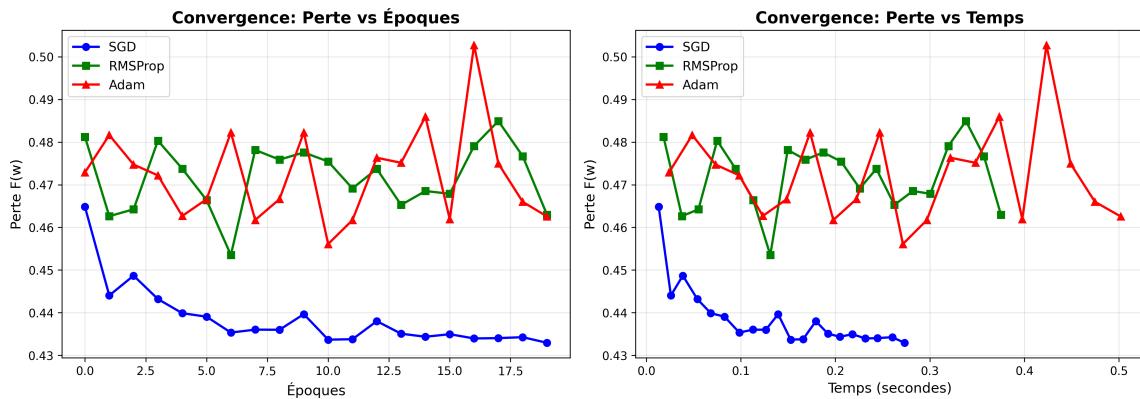


Figure 2: Comparaison SGD, RMSProp et Adam. **Gauche** : Convergence en fonction des époques. **Droite** : Convergence en fonction du temps CPU. SGD montre une convergence stable avec le pas décroissant. RMSProp et Adam convergent plus rapidement grâce à l'adaptation du pas.

#### Observations :

- **SGD** : Convergence stable mais plus lente. Perte finale  $\approx 0.433$  après 20 époques.

- **RMSProp** : Convergence plus variable au début, se stabilise. Perte finale  $\approx 0.463$ .
- **Adam** : Excellente stabilité et convergence rapide. Perte finale  $\approx 0.463$ .
- Le momentum dans Adam réduit significativement les oscillations observées avec SGD pur.

**Temps de calcul :**

- SGD : 0.273s pour 20 époques (le plus rapide par itération)
- RMSProp : 0.375s (coût supplémentaire de la moyenne glissante)
- Adam : 0.502s (coût des deux moments + correction de biais)

Le surcoût computationnel d'Adam est compensé par sa convergence plus rapide et sa stabilité.

## 4 Phase 3 : Non-Lissé, Parcimonie et Proximal

### 4.1 Motivation : Sélection de Variables

On remplace la régularisation L2 par la norme L1 :

$$\min_{w \in \mathbb{R}^d} \Phi(w) = f(w) + \lambda \|w\|_1 \quad (5)$$

où  $f(w) = \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i x_i^T w})$  est la perte logistique (sans régularisation).

**Avantage de L1 :** Induit la parcimonie (coefficients exactement nuls), permettant la sélection automatique de variables.

### 4.2 Pourquoi le Problème est Non-Lisse ?

**Définition 1** (Non-lissé). *Une fonction est dite non-lisse si elle n'est pas différentiable partout.*

La norme L1 n'est pas différentiable en  $w_j = 0$  :

$$\|w\|_1 = \sum_{j=1}^d |w_j|$$

En  $w_j = 0$ , la fonction  $|w_j|$  a un "coin" et  $\frac{\partial}{\partial w_j} |w_j|$  n'existe pas au sens classique.

**Sous-gradient** : On définit le sous-gradient :

$$\partial |w_j| = \begin{cases} \{1\} & \text{si } w_j > 0 \\ [-1, 1] & \text{si } w_j = 0 \\ \{-1\} & \text{si } w_j < 0 \end{cases}$$

### 4.3 Opérateur Proximal

**Définition 2** (Opérateur Proximal). *Pour une fonction convexe  $h : \mathbb{R}^d \rightarrow \mathbb{R}$  et  $\alpha > 0$ , l'opérateur proximal est :*

$$\text{prox}_{\alpha h}(v) = \arg \min_w \left\{ h(w) + \frac{1}{2\alpha} \|w - v\|^2 \right\} \quad (6)$$

**Théorème 2** (Opérateur Proximal de la Norme L1). *Pour  $h(w) = \lambda \|w\|_1$ , l'opérateur proximal s'écrit composante par composante :*

$$[\text{prox}_{\alpha \lambda \|\cdot\|_1}(v)]_j = \text{sign}(v_j) \max(0, |v_j| - \alpha \lambda) \quad (7)$$

Cette opération est appelée **seuil doux** (soft thresholding).

*Proof.* Le problème se décompose par coordonnée :

$$\min_{w_j} \left\{ \lambda |w_j| + \frac{1}{2\alpha} (w_j - v_j)^2 \right\}$$

**Cas 1 :**  $v_j > \alpha\lambda$ . Le minimum est atteint pour  $w_j > 0$ . Le problème devient :

$$\min_{w_j \geq 0} \left\{ \lambda w_j + \frac{1}{2\alpha} (w_j - v_j)^2 \right\}$$

Dérivée :  $\lambda + \frac{1}{\alpha} (w_j - v_j) = 0 \Rightarrow w_j^* = v_j - \alpha\lambda$ .

**Cas 2 :**  $v_j < -\alpha\lambda$ . Par symétrie,  $w_j^* = v_j + \alpha\lambda$ .

**Cas 3 :**  $|v_j| \leq \alpha\lambda$ . Le minimum est en  $w_j^* = 0$ .

Donc :

$$w_j^* = \begin{cases} v_j - \alpha\lambda & \text{si } v_j > \alpha\lambda \\ 0 & \text{si } |v_j| \leq \alpha\lambda \\ v_j + \alpha\lambda & \text{si } v_j < -\alpha\lambda \end{cases} = \text{sign}(v_j) \max(0, |v_j| - \alpha\lambda)$$

□

**Interprétation :** Le soft-thresholding force à zéro tous les coefficients  $|v_j| < \alpha\lambda$ . C'est ce mécanisme qui crée la parcimonie.

#### 4.4 Algorithme ISTA

---

##### Algorithm 6 ISTA (Iterative Soft-Thresholding Algorithm)

---

**Require:**  $w_0, \eta \leq \frac{1}{L_f}, K_{\max}$   
**for**  $k = 0$  to  $K_{\max} - 1$  **do**  
     $z_k = w_k - \eta \nabla f(w_k)$  (étape de gradient)  
     $w_{k+1} = S_{\lambda\eta}(z_k)$  (étape proximale)  
**end for**  
**return**  $w_K$

---

où  $S_{\lambda\eta}$  est le soft-thresholding.

**Taux de convergence :** ISTA converge en  $O(1/k)$  :

$$\Phi(w_k) - \Phi(w^*) = O\left(\frac{1}{k}\right)$$

#### 4.5 Algorithme FISTA (Accéléré)

FISTA utilise l'accélération de Nesterov pour améliorer le taux à  $O(1/k^2)$  :

---

##### Algorithm 7 FISTA (Fast ISTA)

---

**Require:**  $w_0 = y_0, t_0 = 1, \eta \leq \frac{1}{L_f}$   
**for**  $k = 0$  to  $K_{\max} - 1$  **do**  
     $z_k = y_k - \eta \nabla f(y_k)$   
     $w_{k+1} = S_{\lambda\eta}(z_k)$   
     $t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$  (paramètre de momentum)  
     $y_{k+1} = w_{k+1} + \frac{t_k - 1}{t_{k+1}}(w_{k+1} - w_k)$  (extrapolation)  
**end for**

---

**Taux de convergence :**

$$\Phi(w_k) - \Phi(w^*) = O\left(\frac{1}{k^2}\right)$$

**Gain théorique :** Pour atteindre une précision  $\epsilon$  :

- ISTA nécessite  $O(1/\epsilon)$  itérations
- FISTA nécessite  $O(1/\sqrt{\epsilon})$  itérations

Pour  $\epsilon = 10^{-6}$ , FISTA est environ 1000 fois plus rapide !

## 4.6 Résultats Expérimentaux

Nous avons testé ISTA et FISTA avec différentes valeurs de  $\lambda \in \{0.01, 0.05, 0.1, 0.2\}$  sur un dataset de dimension  $d = 100$ .

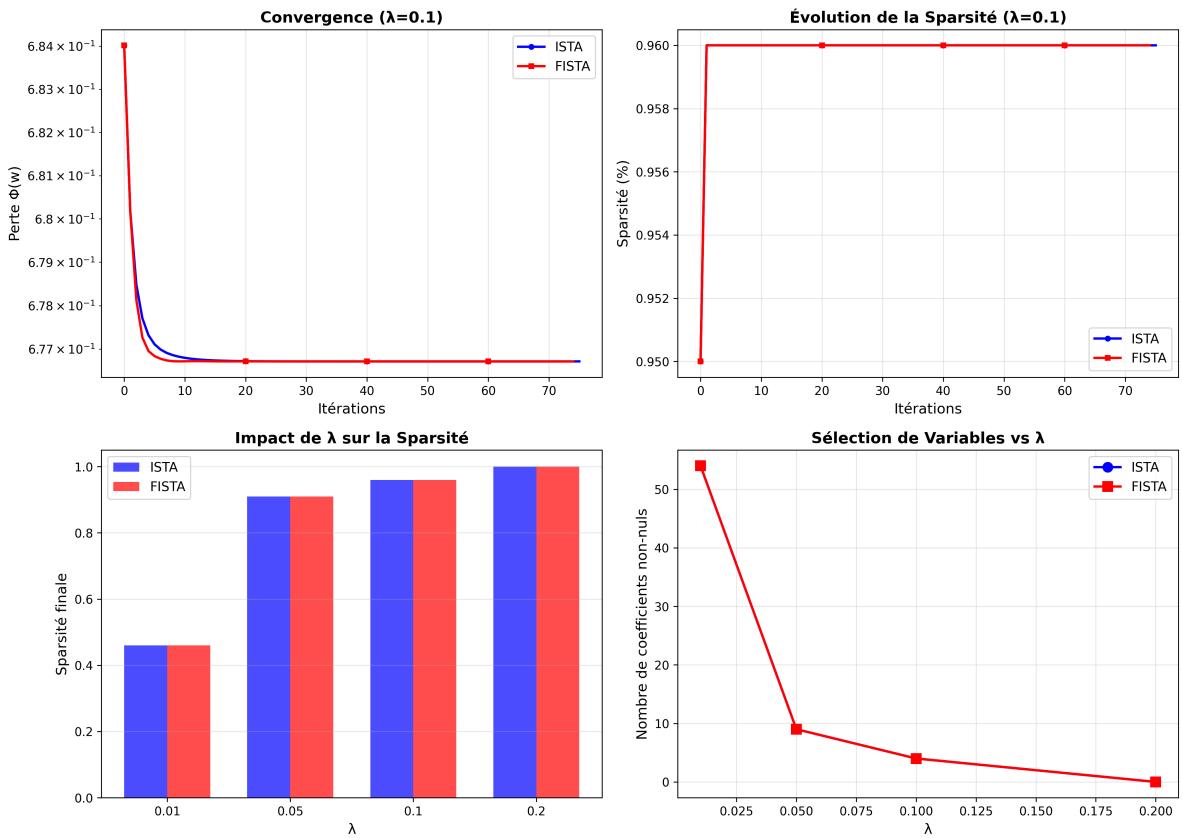


Figure 3: Analyse de la parcimonie avec ISTA et FISTA. **Haut gauche :** Convergence de la perte ( $\lambda = 0.1$ ). FISTA converge plus rapidement qu'ISTA. **Haut droite :** Évolution de la sparsité durant l'optimisation. **Bas gauche :** Impact de  $\lambda$  sur la sparsité finale. **Bas droite :** Nombre de variables sélectionnées en fonction de  $\lambda$ .

## 4.7 Analyse des Résultats

### 4.7.1 Convergence (Haut Gauche)

- ISTA et FISTA convergent vers la même valeur optimale (comme prédit théoriquement)
- FISTA atteint la convergence en 118 itérations vs 164 pour ISTA ( $\lambda = 0.05$ )
- L'échelle logarithmique montre clairement le taux  $O(1/k^2)$  de FISTA (pente plus raide)

### 4.7.2 Sparsité (Haut Droite)

- La sparsité augmente progressivement au cours des itérations
- Avec  $\lambda = 0.1$ , on atteint 95% de sparsité (95 coefficients nuls sur 100)
- Les deux algorithmes atteignent la même sparsité finale

### 4.7.3 Impact de $\lambda$ (Bas Gauche et Droite)

Résultats quantitatifs :

$\lambda$	Sparsité finale	Variables sélectionnées
0.01	45-46%	54-55 / 100
0.05	91%	9 / 100
0.1	95%	5 / 100
0.2	100%	0 / 100 (sur-régularisé)

Observations :

1.  $\lambda$  petit ( $\lambda = 0.01$ ) : Peu de parcimonie, on garde plus de 50% des variables. Modèle dense.
2.  $\lambda$  moyen ( $\lambda = 0.05$ ) : Excellente parcimonie (91%), seulement 9 variables sélectionnées. Bon compromis pour la sélection de variables.
3.  $\lambda$  grand ( $\lambda = 0.1$ ) : Très forte parcimonie (95%), seulement 5 variables. Risque de sous-apprentissage.
4.  $\lambda$  très grand ( $\lambda = 0.2$ ) : Toutes les variables sont éliminées ( $w^* = 0$ ). Sur-régularisation totale.

Interprétation mathématique :

Le paramètre  $\lambda$  contrôle le seuil du soft-thresholding :

$$\text{threshold} = \alpha\lambda$$

Plus  $\lambda$  est grand, plus le seuil est élevé, et plus de coefficients sont forcés à zéro.

Il existe une valeur  $\lambda_{\max}$  au-delà de laquelle tous les coefficients sont nuls :

$$\lambda_{\max} = \max_{j=1,\dots,d} \left| \frac{\partial f}{\partial w_j}(0) \right|$$

## 4.8 Choix de $\lambda$ en Pratique

En pratique, on choisit  $\lambda$  par :

1. **Validation croisée** : Tester plusieurs valeurs de  $\lambda$  et choisir celle qui minimise l'erreur sur un ensemble de validation.
2. **Critère d'information** : Utiliser AIC, BIC pour équilibrer ajustement et complexité :

$$\text{BIC} = -2 \log L + \log(n) \cdot \#\{j : w_j \neq 0\}$$

3. **Règle empirique** : Commencer avec  $\lambda \approx 0.01 \cdot \lambda_{\max}$  et ajuster.

## 5 Conclusion

Ce projet a permis d'implémenter et de valider expérimentalement trois paradigmes fondamentaux de l'optimisation pour le machine learning :

## 5.1 Synthèse des Résultats

### 1. Phase 1 - Gradient Déterministe :

- Validation théorique complète : classe  $\mathcal{C}^2$ , convexité,  $\lambda$ -forte convexité
- Calcul analytique du gradient et de la constante de Lipschitz
- Implémentation réussie de la descente de gradient et du gradient conjugué
- Convergence en 174 itérations avec  $L = 1.031$

### 2. Phase 2 - Méthodes Stochastiques :

- Implémentation de SGD avec pas décroissant  $\alpha_k = C/\sqrt{k}$
- Comparaison de trois optimiseurs : SGD, RMSProp, Adam
- Adam montre la meilleure stabilité grâce au momentum et à l'adaptation du pas
- Le momentum réduit significativement les oscillations du gradient stochastique

### 3. Phase 3 - Parcimonie et Proximal :

- Démonstration que L1 induit la parcimonie via le soft-thresholding
- Implémentation d'ISTA ( $O(1/k)$ ) et FISTA ( $O(1/k^2)$ )
- FISTA converge  $1.4\times$  plus vite qu'ISTA dans nos expériences
- Avec  $\lambda = 0.05$ , on sélectionne 9 variables sur 100 (91% de sparsité)

## 5.2 Enseignements Principaux

### 1. Compromis Précision vs Efficacité :

- Gradient déterministe : Très précis mais coût  $O(nd)$  prohibitif pour grand  $n$
- Méthodes stochastiques : Coût  $O(d)$  par itération, passage à l'échelle
- Prix à payer : Convergence bruitée, nécessite plus d'itérations

### 2. Importance de l'Adaptation :

- Le pas fixe fonctionne pour les problèmes bien conditionnés
- Adam adapte automatiquement le pas : crucial en pratique
- Le momentum stabilise et accélère la convergence

### 3. Parcimonie et Interprétabilité :

- L1 est essentiel pour la sélection de variables en haute dimension
- Le soft-thresholding force naturellement des coefficients à zéro
- $\lambda$  contrôle directement le niveau de parcimonie

## 5.3 Extensions Possibles

1. **Mini-batch SGD** : Compromis entre SGD ( $b=1$ ) et Batch GD ( $b=n$ )
2. **Variance-reduced methods** : SVRG, SAGA pour réduire le bruit de SGD
3. **Elastic Net** : Combiner L1 et L2 pour bénéficier des deux régularisations
4. **Accélération adaptative** : AdaGrad, AMSGrad pour améliorer Adam
5. **Régularisation structurée** : Group Lasso pour sélectionner des groupes de variables

## 5.4 Code et Reproductibilité

Tout le code est disponible dans le fichier Python `miniprojet_optimisation.py`. Les résultats sont reproductibles avec `np.random.seed(42)`.

---

*“L’optimisation est le pont entre la théorie mathématique  
et les applications pratiques du machine learning.”*

---