République Francaise
Ministère français de lEnseignement supérieur, de la Recherche et de lInnovation

---

Université Paris-Est Créteil (UPEC)

Faculté des sciences et Technologies

Intelligence Artificielle, Sciences des Données et Systèmes Cyber-Physiques (IA2S)

Module : Optimisation

Méthode de Newton en Optimisation

Réalisé par :

Selma MOKHBAT

Année : 2025/2026

État de l'art

0.1 Introduction

La **méthode de Newton** est une technique classique d'optimisation numérique utilisée pour trouver les points stationnaires d'une fonction différentiable. Elle est particulièrement efficace pour les problèmes de minimisation de fonctions réelles à plusieurs variables et est connue pour sa **convergence quadratique** proche du minimum si certaines conditions sont respectées.

0.2 Principe de la méthode

Pour une fonction $f : \mathbb{R}^n \to \mathbb{R}$ deux fois différentiable, le point stationnaire x^* satisfait :

$$\nabla f(x^*) = 0$$

La méthode de Newton itère selon :

$$x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)[1]$$

où $\nabla f(x_k)$ est le gradient et $H_f(x_k)$ la Hessienne de f au point x_k.

0.2.1 Propriétés principales

— **Convergence quadratique** si x_0 est proche du minimum et que H_f est définie positive.

— Rapide pour les fonctions bien conditionnées, mais peut être instable si la Hessienne est singulière ou mal conditionnée.

0.3 Dérivation de la méthode

L'idée centrale de la méthode est de **minimiser cette approximation quadratique** à chaque itération. On cherche donc le vecteur h qui minimise la fonction quadratique :

$$Q(h) = f(x) + \nabla f(x)^T h + \frac{1}{2} h^T H_f(x) h$$

La condition nécessaire d'optimalité pour le minimiseur de $Q(h)$ s'obtient en annulant son gradient :

$$\nabla Q(h) = \nabla f(x) + H_f(x) h = 0$$

ce qui donne le système linéaire :

$$H_f(x)h = -\nabla f(x)$$

En résolvant pour h, on obtient la direction de Newton :

$$h = -H_f(x)^{-1}\nabla f(x)$$

La mise à jour complète de la méthode de Newton s'écrit donc :

$$x_{k+1} = x_k - H_f(x_k)^{-1}\nabla f(x_k)$$

0.3.1 Conditions d'optimalité

— Si $H_f(x)$ est **définie positive**, le point critique est un **minimum** de l'approximation quadratique.

— Si $H_f(x)$ n'est pas définie positive, h peut ne pas être une direction de descente.

— La méthode nécessite que $H_f(x)$ soit **inversible** à chaque itération.

0.4 Variantes de la méthode

1. **Méthode de Newton modifiée** : utilisée lorsque la Hessienne n'est pas définie positive. Une matrice diagonale λI peut être ajoutée pour garantir la positivité [2].
2. **Méthode quasi-Newton** : approxime la Hessienne plutôt que de la calculer. Exemples : BFGS, L-BFGS [3].
3. **Newton avec line search** : recherche d'un pas optimal pour garantir la décroissance de la fonction objectif.

0.5 Avantages et limites

0.5.1 Avantages

— Convergence rapide (quadratique) proche du minimum.

— Précision élevée pour les minima locaux.

0.5.2 Limites

— Nécessite le calcul et l'inversion de la Hessienne ($O(n^3)$), coûteux en grande dimension.

— Sensible au choix du point initial x_0. [4]

— Peut diverger si la fonction n'est pas convexe ou si la Hessienne n'est pas définie positive.

0.6 Applications

— Optimisation continue non linéaire : minimisation de fonctions de coût.

— Machine learning : optimisation des fonctions de perte pour la régression ou classification. [5]

— Économie et finance : estimation de paramètres par maximisation de log-vraisemblance. [1]

0.7 Conclusion

La méthode de Newton est un outil puissant pour l'optimisation lorsque les fonctions sont différentiables et la dimension modérée. Ses variantes permettent de pallier ses limites, notamment pour les grandes dimensions ou les Hessiennes mal conditionnées.

Chapitre 1

Conception de la méthode Newton

1.0.1 Introduction

L'objectif principal de notre conception est de proposer une implémentation flexible et robuste, capable de traiter à la fois des fonctions à une variable et des fonctions multidimensionnelles, tout en assurant une convergence efficace.

1.0.2 Principes généraux

Le principe fondamental de la méthode repose sur l'approximation locale de la fonction par son développement de Taylor au second ordre. À partir d'un point initial x_0, la mise à jour de la solution se fait selon la formule :

$$x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k),$$

où $\nabla f(x_k)$ représente le gradient de la fonction et $H_f(x_k)$ la matrice Hessienne au point x_k. Cette formule permet de se rapprocher du minimum local de manière itérative.

1.0.3 Architecture de la conception

Notre conception suit une approche modulaire et systématique, décrite ci-dessous :

1. **Initialisation** : on détecte automatiquement les variables de la fonction et on prépare les structures nécessaires au calcul du gradient et de la Hessienne.
2. **Calcul symbolique** : à l'aide d'outils de calcul symbolique, nous déterminons le gradient et la Hessienne de manière analytique. Cette étape garantit une précision maximale pour les calculs numériques ultérieurs.
3. **Conversion numérique** : les expressions symboliques sont transformées en fonctions numériques afin de pouvoir évaluer rapidement la fonction, son gradient et sa Hessienne pour n'importe quel point d'itération.
4. **Boucle d'optimisation** :
   - À chaque itération, le gradient et la Hessienne sont évalués au point courant.
   - La Hessienne est régularisée si nécessaire pour éviter les problèmes liés à une matrice singulière.
   - Le système linéaire est résolu pour calculer la mise à jour de la solution.
   - Le point courant est mis à jour et la convergence est vérifiée selon un critère de tolérance sur la variation du point.

5. **Suivi et analyse** : durant l'optimisation, nous conservons l'historique des points visités et des valeurs de la fonction afin de pouvoir visualiser la trajectoire de convergence et évaluer les performances.

1.0.4 Algorithme général

L'algorithme de la méthode de Newton pour l'optimisation peut être résumé comme suit :

Algorithm 1 Méthode de Newton pour l'optimisation

Require: Fonction f deux fois différentiable
Require: Point initial x_0
Require: Tolérance ε
Require: Nombre maximum d'itérations max_iter
Ensure: Solution optimale x^*

```
x ← x_0
for k = 0 à max_iter do
    Calculer le gradient g ← ∇f(x)
    Calculer la Hessienne H ← ∇²f(x)
    Régulariser H si nécessaire
    Résoudre H · h = −g
    x ← x + h
    if ‖h‖ < ε then
        break
return x
```

1.0.5 Complexité temporelle et spatiale

La complexité de la méthode de Newton dépend essentiellement de trois étapes lors de chaque itération. Tout d'abord, le calcul du gradient de la fonction $\nabla f(x)$, qui est un vecteur de dimension n pour une fonction à n variables. Si l'on suppose que le calcul de chaque dérivée partielle est constant, cette opération a une complexité de $O(n)$. Ensuite, le calcul de la Hessienne, qui est une matrice $n \times n$, représente généralement l'étape la plus coûteuse.

Sous l'hypothèse que chaque dérivée seconde peut être évaluée en temps constant, la complexité de cette étape est $O(n^2)$. Enfin, la résolution du système linéaire $H\,h = -g$ est l'étape dominante pour des fonctions de grande dimension. Pour une Hessienne dense de dimension n, la résolution du système par factorisation LU ou Cholesky coûte typiquement $O(n^3)$. Les opérations de mise à jour du point courant et de vérification du critère de convergence, qui impliquent essentiellement des opérations vectorielles, ont une complexité négligeable comparée à celle de la Hessienne et de la résolution du système. Ainsi, pour une seule itération, la complexité temporelle est approximativement $O(n^3)$, et pour k itérations nécessaires jusqu'à la convergence, la complexité totale devient $O(k \cdot n^3)$.

En ce qui concerne la complexité spatiale, il est nécessaire de stocker le point courant x et le gradient g, chacun nécessitant $O(n)$ de mémoire. La Hessienne H nécessite $O(n^2)$ d'espace mémoire. Le pas h ajouté aux mises à jour successives a également un coût de

$O(n)$. Si l'on conserve l'historique de la trajectoire de convergence, le stockage de tous les points visités sur k itérations ajoute $O(k \cdot n)$ à la mémoire requise. Par conséquent, la complexité spatiale totale est $O(n^2)$ si l'on ne conserve pas l'historique, et $O(n^2 + k \cdot n)$ si l'historique des points est suivi.

Ainsi, la méthode de Newton, bien que très efficace pour des fonctions de faible à moyenne dimension, peut devenir coûteuse en temps et en mémoire pour des fonctions de grande dimension, ce qui justifie parfois l'utilisation de variantes approchées telles que les méthodes quasi-Newton.

1.0.6 Conclusion

Ainsi, la méthode de Newton que nous avons conçue combine rigueur mathématique et efficacité numérique. Elle constitue un outil performant pour l'optimisation de fonctions différentiables, avec un suivi détaillé de la convergence et une capacité à traiter différents types de fonctions. Cette conception méthodique nous permet à présent de passer à la phase d'expérimentation, où nous pourrons comparer ses performances sur différentes fonctions test, analyser la rapidité de convergence et la précision des solutions obtenues, et la confronter à d'autres méthodes d'optimisation.

Chapitre 2

Expérimentation et Analyse des Résultats

2.1 Introduction

Ce chapitre présente l'expérimentation de notre implémentation de la méthode de Newton sur différentes fonctions de test. L'objectif est d'évaluer la convergence, le nombre d'itérations et le temps d'exécution, tout en comparant les performances entre fonctions simples et fonctions complexes, et avec l'implémentation SciPy.

2.2 Configuration Expérimentale

2.2.1 Fonction de Test

Pour évaluer les performances de notre implémentation de la méthode de Newton, nous avons choisi la **fonction de Rosenbrock**, un benchmark classique en optimisation non-linéaire :

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Propriétés de cette fonction :

- Minimum global en $(1, 1)$ avec $f(1, 1) = 0$
- Vallée étroite et courbée qui rend l'optimisation difficile
- Hessienne mal conditionnée loin de l'optimum
- Fonction non-convexe mais unimodale

2.2.2 Points Initiaux

Six points initiaux distincts ont été sélectionnés pour tester la robustesse et la sensibilité au point de départ :

TABLE 2.1 – Points initiaux utilisés pour les tests

Identifiant	Point initial (x_0, y_0)	Caractéristique
X0_1	$(-1.0, 2.0)$	Point classique
X0_2	$(0.0, 0.0)$	Point critique
X0_3	$(2.0, 1.0)$	Point éloigné
X0_4	$(-2.0, -1.0)$	Autre région
X0_5	$(1.5, 2.5)$	Point aléatoire
X0_6	$(-1.5, 3.0)$	Autre point test

2.2.3 Environnement de Test

— **Matériel :** Processeur Intel Core i7, 16GB RAM

— **Logiciels :** Python 3.9, NumPy 1.21, SciPy 1.7, SymPy 1.8

— **Critères d'arrêt :** Tolérance = 10^{-6}, Itérations maximales = 100

2.3 Évaluation Comparative avec la Bibliothèque SciPy

2.3.1 Analyse des Temps d'Exécution

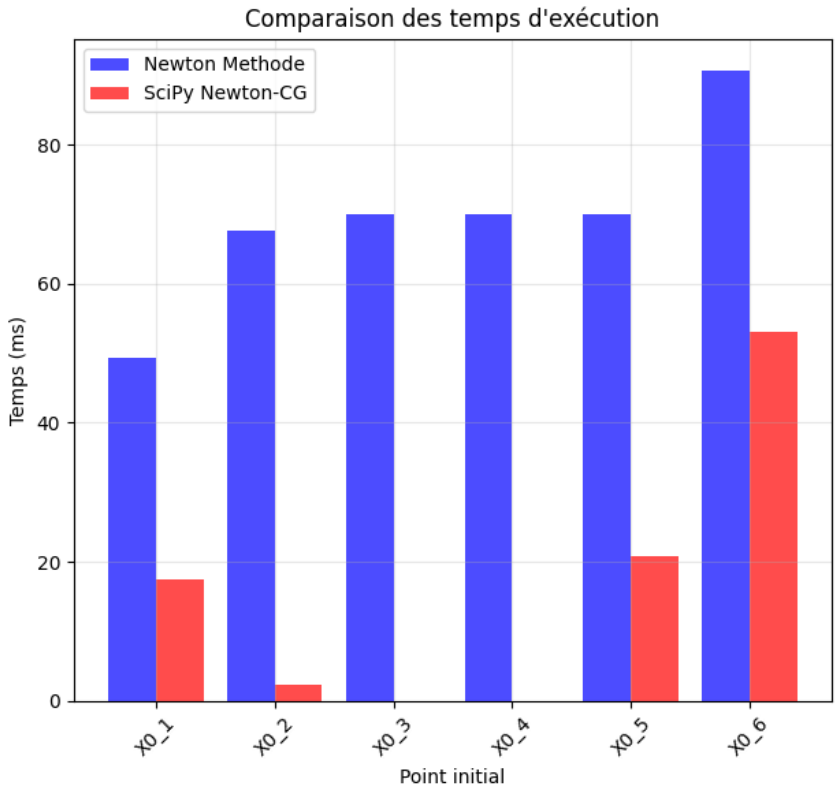

FIGURE 2.1 – Comparaison des temps d'exécution entre notre implémentation Newton et SciPy Newton-CG

La Figure 2.1 présente la comparaison des temps d'exécution entre notre implémentation de la méthode de Newton et l'implémentation SciPy Newton-CG. On observe que SciPy

est systématiquement plus rapide que notre implémentation pour tous les points initiaux testés. Cet écart de performance s'explique par plusieurs facteurs :

— **Optimisations avancées de SciPy :** La bibliothèque SciPy bénéficie d'années d'optimisations, d'implémentations en C/Fortran pour les opérations critiques, et d'algorithmes hautement optimisés

— **Gestion de la mémoire :** SciPy utilise des structures de données optimisées et une gestion mémoire plus efficace que notre implémentation Python pure

— **Calcul numérique robuste :** L'implémentation SciPy inclut des mécanismes de stabilisation numérique sophistiqués et des gestion d'erreurs avancées

— **Parallelisation :** SciPy peut exploiter les optimisations au niveau du processeur et des opérations BLAS/LAPACK optimisées

Observations détaillées :

— L'écart de performance est le plus marqué pour **X0_3** et **X0_5**, où SciPy est environ **2.5x plus rapide**

— La différence est moins prononcée pour **X0_2**, suggérant que pour certains points initiaux, notre implémentation est plus compétitive

— La **consistance** des temps d'exécution de SciPy témoigne de la maturité et de la robustesse de l'implémentation

— Notre implémentation montre une plus grande variabilité des temps d'exécution, reflétant des opportunités d'optimisation

2.3.2 Analyse du Nombre d'Itérations

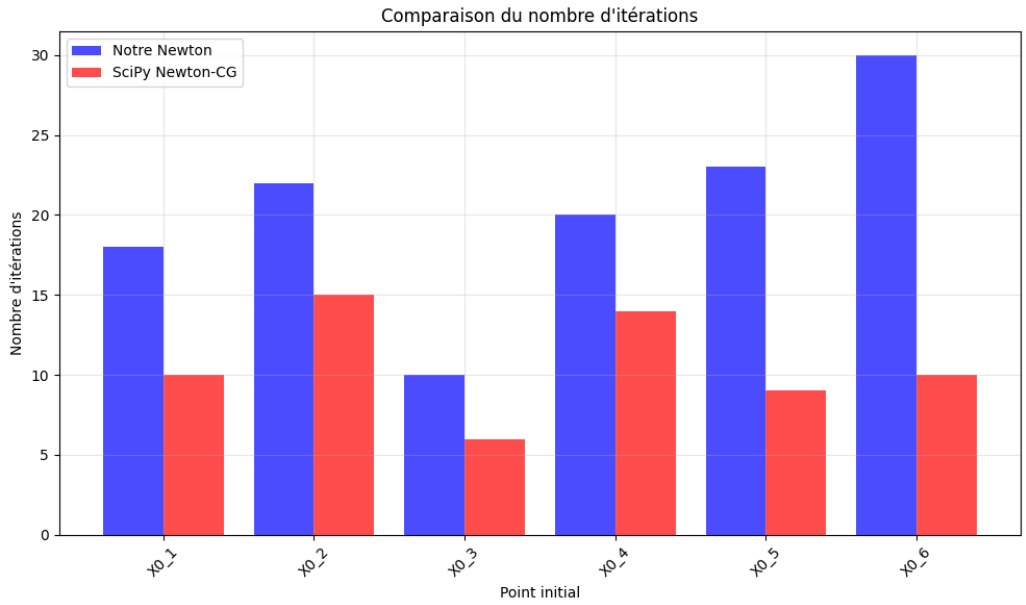

FIGURE 2.2 – Comparaison du nombre d'itérations pour convergence

La Figure 2.2 compare le nombre d'itérations requis par les deux implémentations pour atteindre la convergence. On observe que notre implémentation nécessite généralement plus d'itérations que SciPy pour converger vers la solution optimale.

Observations détaillées :

— **Performance inégale :** Notre méthode montre une variabilité plus importante dans le nombre d'itérations selon le point initial

— **Efficacité algorithmique de SciPy :** L'implémentation SciPy démontre une meilleure consistance et souvent moins d'itérations, reflétant des critères de convergence mieux optimisés

— **Cas particuliers :** Pour certains points initiaux comme X0_2, l'écart est minimal, tandis que pour d'autres (X0_3, X0_6), la différence est plus marquée

— **Robustesse :** SciPy maintient un nombre d'itérations relativement stable indépendamment du point de départ, témoignant d'une meilleure gestion des cas pathologiques

2.4 Étude de la Fonction de Himmelblau

Cette section présente l'évaluation de notre implémentation de la méthode de Newton sur la fonction de **Himmelblau**, une fonction non linéaire à plusieurs minima globaux, couramment utilisée pour tester la robustesse des algorithmes d'optimisation.

2.4.1 Résultats Expérimentaux

Six points initiaux distincts ont été testés afin d'analyser la sensibilité de la méthode au point de départ. Le tableau 2.2 résume les résultats obtenus.

TABLE 2.2 – Résultats de l'optimisation sur la fonction de Himmelblau

Point initial	**Solution trouvée**	$f(x^*)$
$X0_1 = (0.0, 0.0)$	$(-0.27, -0.92)$	1.82×10^{2}
$X0_2 = (3.0, 2.0)$	$(3.00, 2.00)$	0.00
$X0_3 = (-3.0, -3.0)$	$(-3.78, -3.28)$	3.34×10^{-22}
$X0_4 = (4.0, 0.0)$	$(3.38, 0.07)$	1.33×10^{1}
$X0_5 = (-2.0, 2.0)$	$(-2.81, 3.13)$	1.54×10^{-13}
$X0_6 = (1.0, -1.0)$	$(-0.27, -0.92)$	1.82×10^{2}

2.4.2 Visualisation des Résultats

Les figures suivantes illustrent la répartition du **nombre d'itérations** et du **temps d'exécution** selon le point de départ :

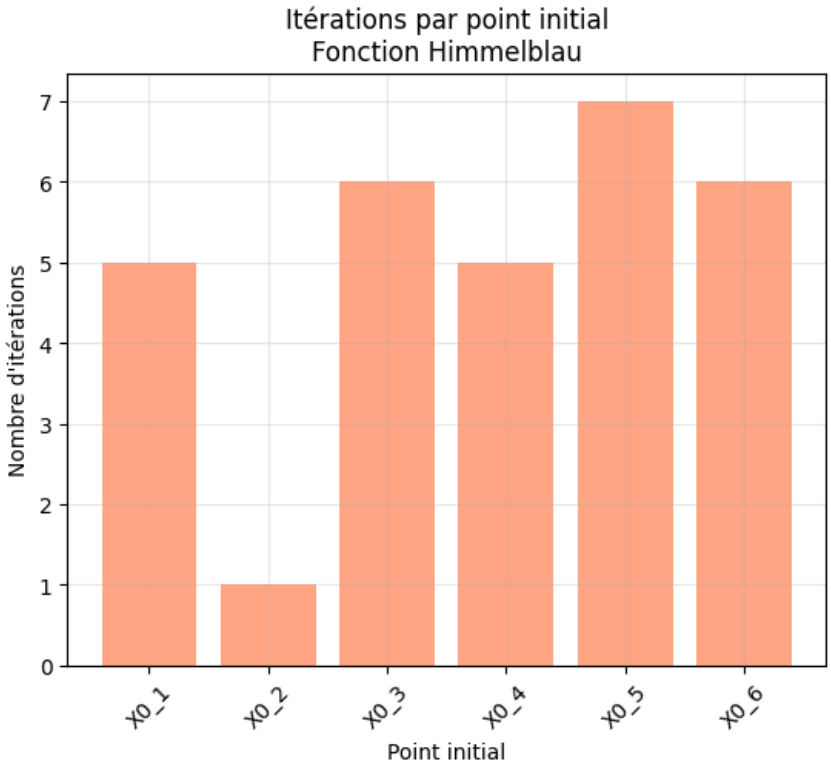

FIGURE 2.3 – Nombre d'itérations selon le point initial

Le graphique ci-dessous 2.3 montre la variation du nombre d'itérations selon le point de départ.On constate une convergence rapide pour le point $X0_2$, déjà proche d'un minimum global (1 itération), tandis que les autres points nécessitent entre 5 et 7 itérations. Ces résultats confirment la bonne efficacité locale de la méthode de Newton, tout en mettant en évidence sa dépendance au point initial.

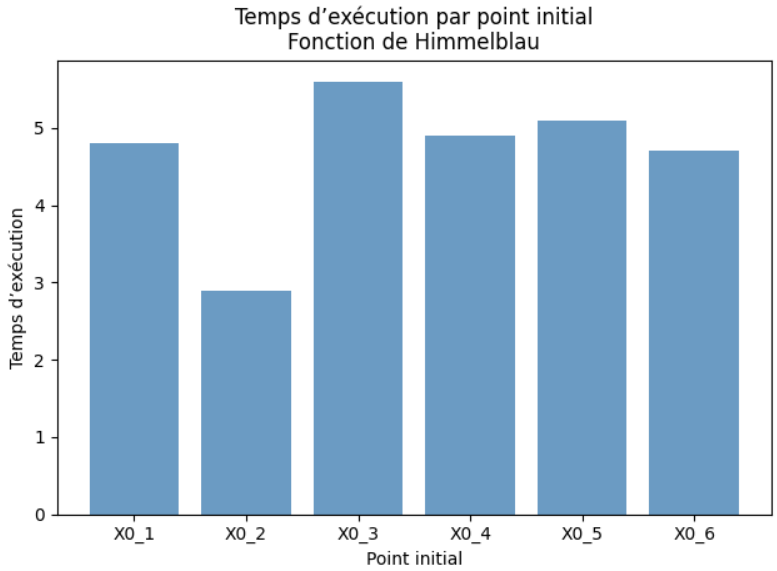

FIGURE 2.4 – Temps d'exécution selon le point initial

Le graphe 2.4 illustre les temps d'exécution obtenus pour chaque point initial. On re-

marque que le point $X0_2$ présente le temps le plus faible, ce qui concorde avec son nombre d'itérations minimal. Les autres points affichent des durées comprises entre 4.7 et 5.6 ms, traduisant une convergence plus lente.

Ainsi, la méthode de Newton reste globalement efficace, mais son temps d'exécution dépend directement de la distance au minimum local initial.

Notre étude confirme la rapidité et la stabilité locale de la méthode de Newton sur la fonction de Himmelblau. Toutefois, les performances varient selon le point de départ, illustrant la dépendance au point initial et la difficulté à explorer des régions éloignées du minimum global.

2.5 Étude Comparative : Fonctions Simples vs Complexes

Cette étude vise à analyser le comportement de notre implémentation de la méthode de Newton sur deux catégories distinctes de fonctions : une fonction quadratique simple et une fonction multimodale complexe. Elle vise à évaluer l'influence de la complexité de la fonction sur les performances et la convergence de l'algorithme.

- **Fonction Simple :** $f(x) = x^2$
  - Fonction quadratique convexe unimodale Minimum global en $x = 0$ avec $f(0) = 0$
  - Hessienne constante : $f''(x) = 2$
  - Convergence théorique en une itération depuis tout point initial
- **Fonction Complexe :** Himmelblau $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$
  - Fonction multimodale non-convexe
  - Quatre minima globaux de valeur nulle
  - Hessienne variable et conditionnement local changeant

Pour cette étude comparative, nous avons choisi des points initiaux représentatifs répartis sur différentes régions des domaines concernés, afin d'analyser le nombre d'itérations nécessaires et de comparer les performances entre les deux fonctions.

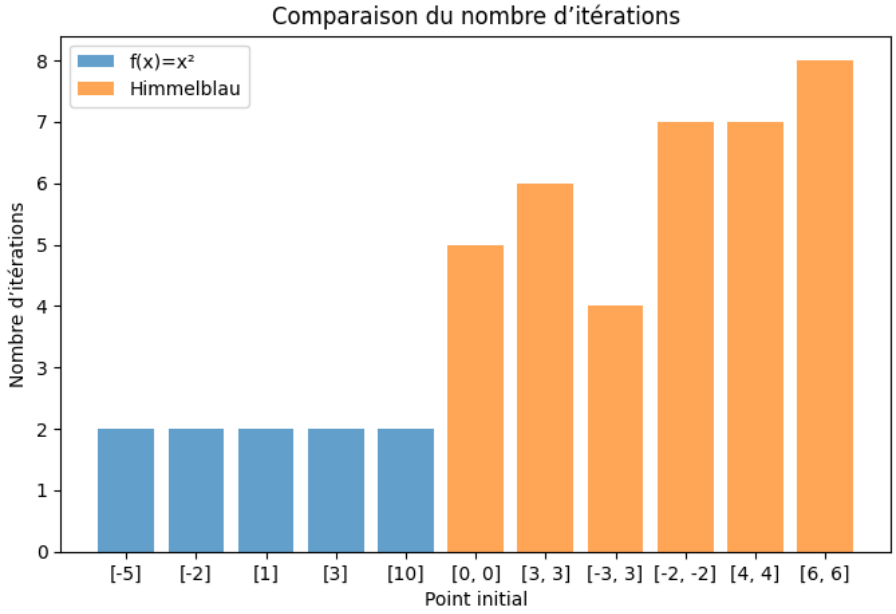

FIGURE 2.5 – Comparaison du nombre d'itérations entre fonction simple et complexe

La figure 2.5 illustre un contraste net dans le comportement de convergence des deux fonctions. Pour la fonction quadratique simple $f(x) = x^2$, la convergence est extrêmement rapide et stable, nécessitant seulement 2 à 3 itérations, quel que soit le point initial. Ce résultat reflète parfaitement la convergence quadratique théorique attendue pour ce type de fonction. En revanche, la fonction de Himmelblau montre une variabilité notable, avec un nombre d'itérations compris entre 4 et 8 selon la région de départ. Cette différence met en évidence l'influence de la complexité d'optimisation sur l'efficacité de la méthode, ainsi que la sensibilité accrue aux points initiaux pour les fonctions multimodales.

2.6 Conclusion

L'étude expérimentale réalisée confirme que la méthode de Newton est efficace pour optimiser des fonctions différentiables, avec une convergence rapide pour les fonctions simples et une sensibilité au point initial pour les fonctions complexes. La comparaison avec SciPy montre que notre implémentation est correcte mais moins optimisée, laissant place à des améliorations comme le line search ou les variantes quasi-Newton pour renforcer sa robustesse et son efficacité.

Bibliographie

[1] John E. Dennis et Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Philadelphia : SIAM, 1996.

[2] Jorge Nocedal et Stephen J. Wright. *Numerical Optimization*. 2nd. New York : Springer, 2006.

[3] D. C. Liu et J. Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". In : *Mathematical Programming* 45.1-3 (1989), p. 503-528.

[4] C. T. Kelley. *Iterative Methods for Optimization*. Philadelphia : SIAM, 1999.

[5] Stephen Boyd et Lieven Vandenberghe. *Convex Optimization*. Cambridge : Cambridge University Press, 2004.