IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

IMT Nord Europe

UV ODATA - Projet
Optimisation

Optimisation d'une méthode de Stéganographie

Élèves :
Boubker BENNANI
Othmane BENZARHOUNI

Table des matières

1 Introduction

Le besoin de dissimuler des informations, ou stéganographie, trouve son origine dans la nécessité de sécuriser les échanges sensibles et confidentiels. Dans un monde où la confidentialité est devenue une préoccupation majeure, la protection des données et des messages contre les regards indiscrets est devenue une priorité.

Cependant, l'impératif de confidentialité ne peut pas toujours être accompli uniquement par la stéganographie. C'est là que la cryptographie entre en jeu. La cryptographie, qui se concentre sur la transformation des messages pour les rendre incompréhensibles à toute personne non autorisée, complémente la stéganographie en assurant la sécurité du contenu des communications.

Alors que la stéganographie vise à dissimuler l'existence même d'une communication secrète, la cryptographie se charge de rendre le contenu de cette communication inintelligible pour quiconque n'est pas destinataire du message. Ces deux approches, bien que distinctes, convergent souvent pour renforcer la confidentialité et la sécurité des informations.

Ainsi, la stéganographie et la cryptographie travaillent de concert pour garantir la confidentialité, l'intégrité et l'authenticité des communications dans des domaines aussi variés que la sécurité nationale, les transactions financières ou la protection des données personnelles. Ensemble, elles forment un rempart essentiel contre les menaces potentielles pesant sur la confidentialité des informations sensibles.

2 Formulation du problème

Ce projet vise à maximiser le taux d'insertion, exprimé en tant que quantité d'information moyenne par pixel, pour une image donnée. Le défi principal est de réaliser cette maximisation tout en respectant une contrainte de distorsion spécifique. La distorsion $D(\pi)$ est initialement définie par :

$$D(\pi) = \sum_{i=1}^{n} \sum_{k=1}^{M} \rho_i(k)\pi_i(k) \tag{1}$$

Dans un cas simplifié où le profil de poids est uniforme pour chaque pixel, cette expression se réduit à :

$$D(\pi) = n \sum_{k=1}^{M} \rho(k)\pi(k) \tag{2}$$

Par conséquent, la distorsion par pixel, $d(\pi)$, est donnée par :

$$d(\pi) = \frac{D(\pi)}{n} = \sum_{k=1}^{M} \rho(k)\pi(k) \tag{3}$$

Les variables clés sont les valeurs $\pi(1), \ldots, \pi(M)$ avec la contrainte que $\sum_{k=1}^{M} \pi(k) = 1$. La fonction d'optimisation $f_0(\pi)$ est définie comme :

$$f_0(\pi) = \sum_{k=1}^{M} \pi(k) \log_2 \left(\frac{1}{\pi(k)} \right) \text{ bits/pixel} \tag{4}$$

L'objectif est de maximiser $f_0(\pi)$ tout en imposant que la distorsion par pixel $d(\pi)$ soit égale à d_0 ou inférieure à cette valeur.

3 Formulation du Programme d'Optimisation

Dans cette section, nous formulons le programme d'optimisation pour maximiser le taux d'insertion pour une distorsion donnée. L'objectif est d'optimiser la fonction $f_0(\pi)$, tout en respectant les contraintes de distorsion.

3.1 Fonction Objectif

La fonction à maximiser est définie comme suit :

$$\max_{\pi} f_0(\pi) = \max_{\pi} \sum_{k=1}^{M} \pi(k) \log_2 \left(\frac{1}{\pi(k)} \right) \tag{1}$$

où $\pi(k)$ représente la probabilité d'insertion du k-ième élément et M est le nombre total d'éléments.

3.2 Contraintes de Distorsion

La contrainte principale est que la distorsion par pixel ne doit pas dépasser une valeur prédéterminée d_0 :

$$d(\pi) = \frac{D(\pi)}{n} = \sum_{k=1}^{M} \rho(k)\pi(k) \leq d_0 \tag{2}$$

où $D(\pi)$ est la distorsion totale, n est le nombre total de pixels, et $\rho(k)$ est le poids du k-ième élément.

3.3 Contraintes Supplémentaires

Les autres contraintes incluent :

— La somme des probabilités $\pi(k)$ doit être égale à 1 :

$$\sum_{k=1}^{M} \pi(k) = 1 \tag{3}$$

— Chaque probabilité $\pi(k)$ doit être positive ou nulle :

$$\pi(k) \geq 0, \quad \forall k = 1, \ldots, M \tag{4}$$

Cette formulation permet d'encadrer le problème d'optimisation et de définir clairement l'objectif et les contraintes sous lesquelles il doit être résolu.

3.4 Formulation Standard du Programme

La formulation standard du programme d'optimisation est présentée comme un système d'équations et d'inégalités. Ce système englobe la fonction objectif ainsi que l'ensemble des contraintes à respecter.

$$\min_{\pi} \quad -\sum_{k=1}^{M} \pi(k) \log_2 \left(\frac{1}{\pi(k)} \right)$$

sous les contraintes

$$(f1) \quad \sum_{k=1}^{M} \rho(k)\pi(k) - d_0 \leq 0, \tag{5}$$

$$(h0) \quad \sum_{k=1}^{M} \pi(k) - 1 = 0,$$

$$(f2) \quad \pi(k) \geq 0, \quad \forall k = 1, \ldots, M.$$

4 Analyse de la Convexité du Problème

Pour démontrer la convexité du problème d'optimisation formulé, il est crucial d'examiner la nature de la fonction objectif et des contraintes. Un problème d'optimisation est dit convexe si la fonction objectif est convexe et si l'ensemble de contraintes forme un ensemble convexe.

4.1 Convexité de la Fonction Objectif

La fonction objectif, définie par

$$f_0(\pi) = \sum_{k=1}^{M} \pi(k) \log_2 \left(\frac{1}{\pi(k)} \right) \tag{6}$$

est une somme de fonctions convexes car la fonction $x \log_2 \left(\frac{1}{x}\right)$ est convexe pour $x > 0$. La convexité de cette fonction peut être démontrée en calculant sa seconde dérivée et en montrant qu'elle est positive.

4.2 Convexité des Contraintes

Les contraintes du problème sont également convexes.

— La contrainte (f1) est une inégalité linéaire, et toute inégalité linéaire définit un demi-espace, qui est un ensemble convexe.

— La contrainte (h0), étant une équation linéaire, définit un hyperplan, qui est également un ensemble convexe.

— La contrainte (f2), qui est une inégalité linéaire, définit également un demi-espace convexe.

En conclusion, puisque la fonction objectif et l'ensemble des contraintes sont convexes, le problème d'optimisation dans son ensemble est un problème de programmation convexe.

5 Choix des Algorithmes pour la Résolution du Problème

Pour résoudre le problème d'optimisation convexe formulé, plusieurs méthodes algorithmiques peuvent être envisagées. Nous discutons ici de l'application de la méthode de Newton, de la méthode du gradient projeté, et de l'utilisation d'un solveur exponentiel via la bibliothèque ‘cvxpy‘.

5.1 Méthode de Newton

La méthode de Newton est particulièrement efficace pour les problèmes d'optimisation convexe en raison de sa convergence rapide vers un minimum local, qui est également un minimum global dans le cas d'un problème convexe. Elle utilise la dérivée seconde (matrice hessienne) pour obtenir une approximation quadratique de la fonction autour du point courant, permettant ainsi des mises à jour plus précises que la descente de gradient simple.

1. À partir d'un point initial admissible x_0 (tel que $Ax_0 - b = 0$) et un critère d'arrêt $\epsilon > 0$, calculer $\nabla f_0(x_0)$ et $\nabla^2 f_0(x_0)$. Calculer la direction admissible d_0^* :

$$\begin{bmatrix} d_0^* \\ v_0^* \end{bmatrix} = \begin{bmatrix} \nabla^2 f_0(x_0) & A^T \\ A & 0 \end{bmatrix}^{-1} \times \begin{bmatrix} -\nabla f_0(x_0) \\ 0 \end{bmatrix} \tag{7}$$

2. Tant que $\frac{1}{2}(d_k^*)^T \nabla^2 f(x_k) d_k^* > \epsilon$, mettre à jour : $x_{k+1} = x_k + d_k^*$. Calculer $\nabla f_0(x_{k+1})$ et $\nabla^2 f_0(x_{k+1})$, et la direction admissible d_{k+1}^*.

5.1.1 Implémentation de la Méthode de Newton avec Contraintes d'Égalité

Cette section détaille l'implémentation de la méthode de Newton pour des problèmes d'optimisation avec des contraintes d'égalité, comme décrit dans le code Python fourni.

1. **Initialisation :** Le point de départ π_0 est initialisé et une tolérance ϵ est définie pour la convergence de l'algorithme.
2. **Boucle Principale :** L'algorithme itère jusqu'à un nombre maximal d'itérations.
3. **Calcul du Gradient et de la Hessienne :** À chaque itération, le gradient $\nabla f(\pi)$ et la matrice hessienne $\nabla^2 f(\pi)$ sont calculés.
4. **Vérification de la Définie Positivité :** Il est vérifié si la matrice hessienne est définie positive. Si elle ne l'est pas, l'algorithme s'arrête.
5. **Construction des Contraintes :** Les matrices de contraintes linéaires A et b sont construites, ainsi que la transposée de A.
6. **Résolution du Système Linéaire :** Un système d'équations linéaires est formé et résolu pour obtenir la direction de mise à jour $\Delta\pi$.
7. **Mise à Jour de la Solution :** La solution π est mise à jour en ajoutant $\Delta\pi$.
8. **Condition de Convergence :** L'algorithme vérifie si la norme de $\Delta\pi$ est inférieure à ϵ pour déterminer la convergence.
9. **Résultat :** L'algorithme renvoie le vecteur π optimisé.

Cette implémentation de la méthode de Newton prend en compte les contraintes d'égalité et utilise des techniques numériques pour assurer la convergence vers une solution optimale.

5.2 Méthode du Gradient Projeté

La méthode du gradient projeté est adaptée lorsque le problème inclut des contraintes, comme c'est le cas ici. Après chaque étape de descente de gradient, les solutions sont projetées sur l'ensemble admissible défini par les contraintes. Cette méthode assure que les solutions restent dans l'espace défini par les contraintes à chaque itération.

La méthode du gradient projeté est une technique d'optimisation qui intègre les contraintes dans le processus de minimisation d'une fonction. Elle est particulièrement utile pour les problèmes où les solutions doivent rester dans un ensemble admissible défini par des contraintes.

5.2.1 Descente de Gradient et Projection

Considérons une fonction objectif $f : R^n \to R$ à minimiser, sous la contrainte que le vecteur de solution $\mathbf{x} \in \mathcal{C}$, où $\mathcal{C}$ est l'ensemble admissible défini par les contraintes. La méthode du gradient projeté opère comme suit :

— **Étape de Descente de Gradient :** À partir d'un point initial $\mathbf{x}_0$, on calcule le nouveau point par descente de gradient :

$$\mathbf{x}_{\text{temp}} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) \tag{8}$$

où α est le pas de descente et $\nabla f(\mathbf{x}_k)$ est le gradient de f au point $\mathbf{x}_k$.

— **Étape de Projection :** Le point $\mathbf{x}_{\text{temp}}$ est ensuite projeté sur l'ensemble admissible $\mathcal{C}$:

$$\mathbf{x}_{k+1} = \text{proj}_{\mathcal{C}}(\mathbf{x}_{\text{temp}}) \tag{9}$$

où $\text{proj}_{\mathcal{C}}(\mathbf{y})$ désigne la projection du point $\mathbf{y}$ sur l'ensemble $\mathcal{C}$.

5.2.2 Projection sur les Contraintes

La projection $\text{proj}_{\mathcal{C}}(\mathbf{y})$ est le point dans $\mathcal{C}$ le plus proche de $\mathbf{y}$ au sens de la norme euclidienne, résolvant ainsi le problème d'optimisation suivant :

$$\text{proj}_{\mathcal{C}}(\mathbf{y}) = \arg\min_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x} - \mathbf{y}\|_2 \tag{10}$$

5.2.3 Convergence et Choix du Pas de Descente

Le choix du pas de descente α est crucial pour la convergence de l'algorithme. Un α trop grand peut entraîner une divergence, tandis qu'un α trop petit peut ralentir la convergence. Dans le cas d'une fonction objectif convexe, la méthode du gradient projeté garantit la convergence vers un minimum global, à condition que le pas de descente soit choisi de manière appropriée.

5.2.4 Implémentation de la Méthode du Gradient Projeté avec Projection Simplexe

Cette section détaille l'implémentation en Python de la méthode du gradient projeté avec une projection sur le simplexe pour un problème d'optimisation.

5.2.5 Projection sur le Simplexe

La fonction `projection_simplex(v, z=1)` réalise une projection sur le simplexe, une étape clé dans la méthode du gradient projeté. Le processus est le suivant :

1. Le vecteur d'entrée $\mathbf{v}$ est trié dans l'ordre décroissant.
2. On calcule la somme cumulée de $\mathbf{u}$ (le vecteur trié) et on soustrait z pour obtenir `cssv`.
3. On détermine le dernier indice ρ où $\mathbf{u} - \frac{\text{cssv}}{\text{ind}} > 0$.
4. θ, la quantité à soustraire de $\mathbf{v}$, est calculée.
5. Le vecteur projeté est obtenu en soustrayant θ de $\mathbf{v}$ et en prenant la partie positive.

Cette projection assure que le vecteur résultant appartient au simplexe, c'est-à-dire que ses composantes sont positives et leur somme égale à z.

5.2.6 Algorithme de la Méthode du Gradient Projeté

La fonction `solve_optimization_alternative(M, rho, d0, alpha=0.01, max_iter=1000)` implémente la méthode du gradient projeté. Les étapes clés sont les suivantes :

1. Initialisation de π comme un vecteur uniforme de longueur M.
2. Boucle sur un nombre maximal d'itérations :
   — Calcul du gradient de la fonction objectif.
   — Mise à jour de π par une étape de descente de gradient.
   — Projection de π sur le simplexe.
   — Vérification de la contrainte d'inégalité $\rho \cdot \pi \leq d_0$. Si la contrainte est satisfaite, l'algorithme s'arrête.
3. Retour de la solution π optimisée.

Cette implémentation permet d'optimiser la fonction objectif tout en respectant la contrainte de distorsion et en assurant que les solutions restent dans l'ensemble des probabilités admissibles.

5.3 Solveur Exponentiel avec cvxpy

Pour des problèmes convexes complexes, l'utilisation de bibliothèques spécialisées comme 'cvxpy' peut être bénéfique. 'cvxpy' offre une interface de haut niveau pour la modélisation et la résolution de problèmes d'optimisation. Le solveur exponentiel de 'cvxpy' est conçu pour traiter efficacement de s pr oblèmes co nvexes, en off rant des algorithmes optimisés pour une large gamme de problèmes.

5.4 Résolution du problème d'Optimisation avec cvxpy

Cette section détaille l'implémentation en Python d'un problème d'optimisation convexe en utilisant la bibliothèque 'cvxpy'. L'objectif est de maximiser une fonction objectif sous des contraintes linéaires.

5.4.1 Implémentation

Le problème d'optimisation est défini comme suit :

1. **Variable de Décision :** La variable π est définie comme une variable de décision de dimension M dans 'cvxpy'.

$$\pi = \text{cp.Variable}(M) \tag{11}$$

2. **Fonction Objectif :** La fonction objectif à maximiser est la somme des entropies des éléments de π.

$$\text{objective} = \text{cp.Maximize}(\text{cp.sum}(\text{cp.entr}(\pi))) \tag{12}$$

3. **Contraintes :** Les contraintes du problème incluent la somme des éléments de π égale à 1, et le produit scalaire de ρ et π inférieur ou égal à d_0.

$$\text{constraints} = [\text{cp.sum}(\pi) == 1, \text{cp.matmul}(\rho, \pi) \leq d_0] \tag{13}$$

5.4.2 Résolution du Problème

Le problème est résolu en utilisant le solveur ECOS de 'cvxpy'. Le processus est le suivant :

1. **Formulation du Problème :** Le problème est formulé avec l'objectif et les contraintes définies précédemment.

$$\text{problem} = \text{cp.Problem}(\text{objective}, \text{constraints}) \tag{14}$$

2. **Résolution :** Le problème est résolu en utilisant la fonction `solve`. Le solveur ECOS est utilisé pour la résolution.

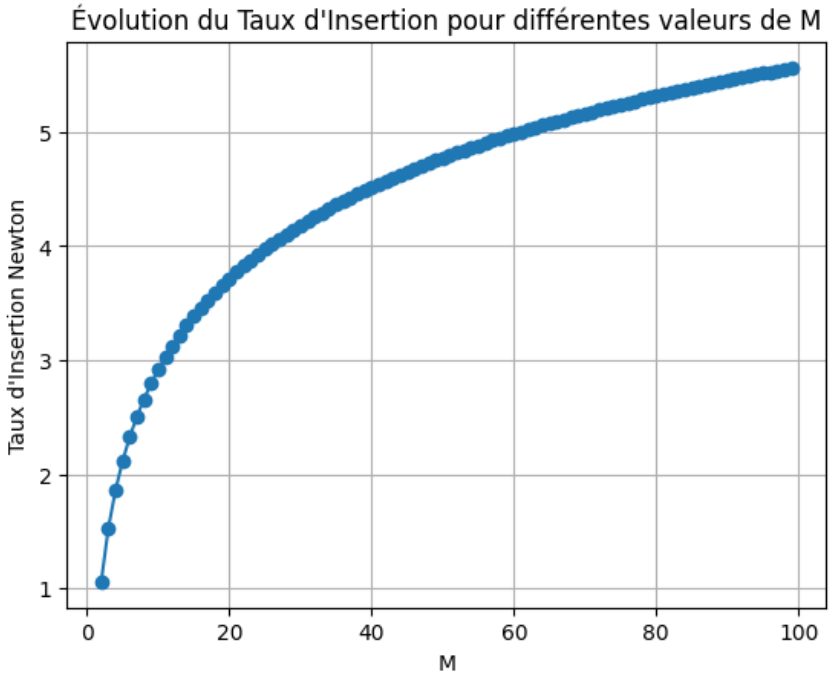
$$\text{result} = \text{problem.solve}(\text{solver} = \text{cp.ECOS}) \tag{15}$$

3. **Vérification du Statut :** Après la résolution, le statut du problème est vérifié. Si le problème est infaisable ou non borné, la fonction renvoie `None`. Sinon, la solution optimisée π est retournée.

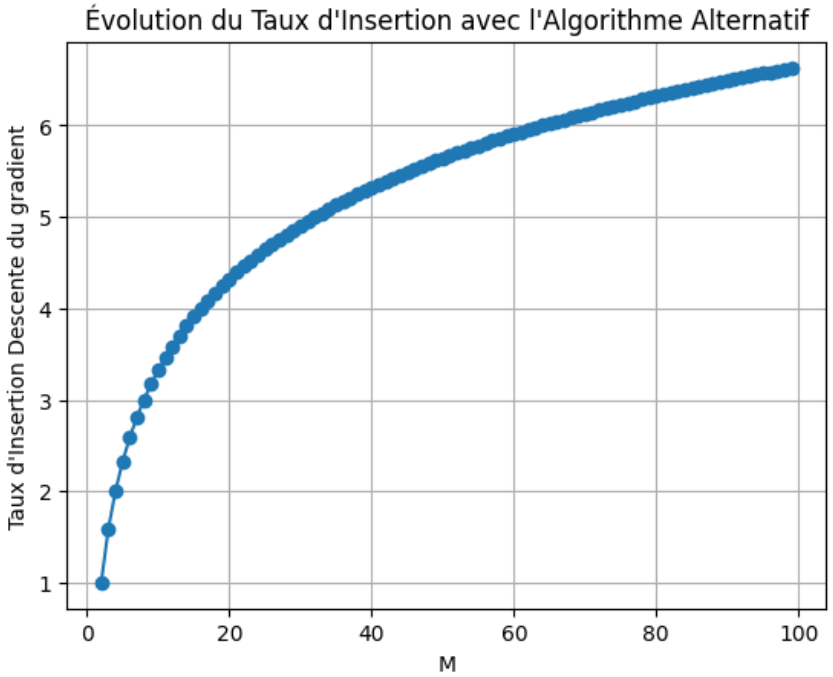
Cette implémentation permet de résoudre efficacement le problème d'optimisation convexe en utilisant des outils de programmation convexe avancés fournis par 'cvxpy'.

6 Graphe montrant l'évolution du taux d'insertion pour différentes valeurs de M pour une même distorsion $d_0 = 1.5$.

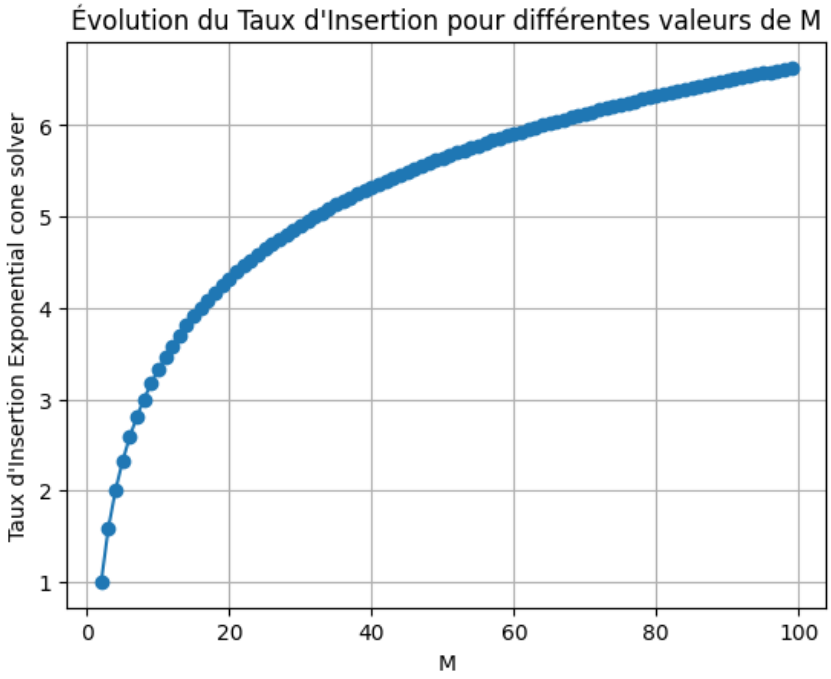
6.1 Méthode de Newton



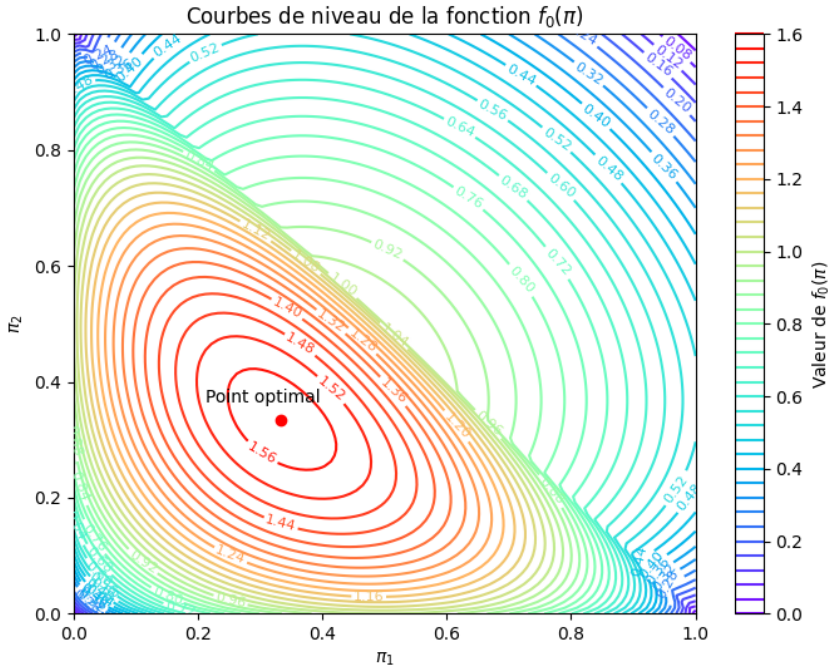
6.2 Méthode du Gradient Projeté

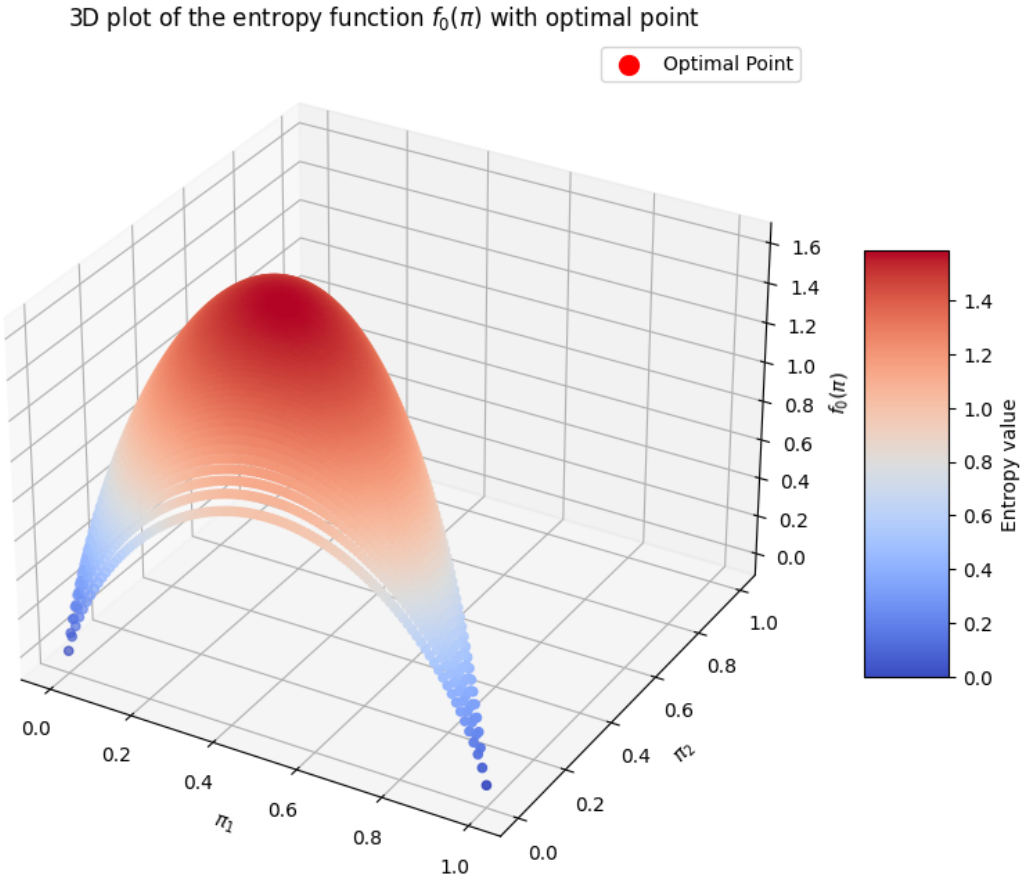


6.3 Solveur Exponentiel avec cvxpy



7 courbe iso-niveaux de f_0



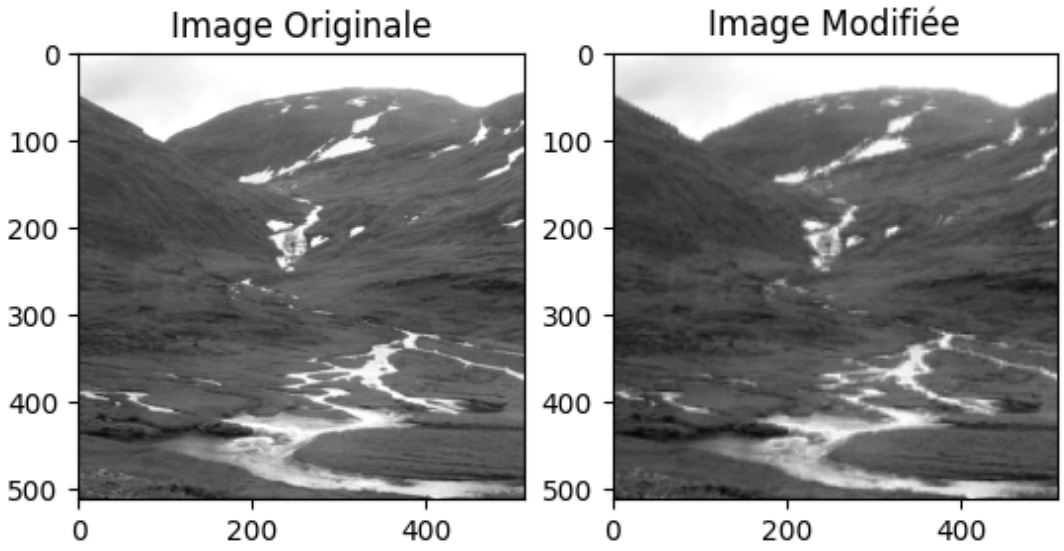

3D plot of the entropy function $f_0(\pi)$ with optimal point
Optimal Point
π_1
π_2
$f_0(\pi)$
Entropy value
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

8

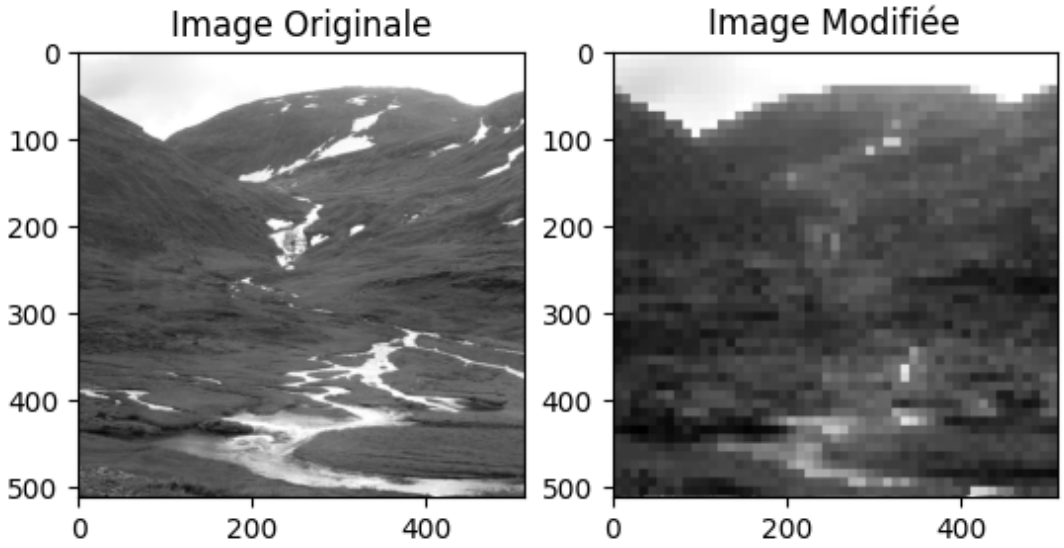
8.1 Hypothèse d'Homogénéité des Poids sur l'Ensemble des Pixels

8.1.1 Méthode de Newton

Taille du block $N_b = 3$

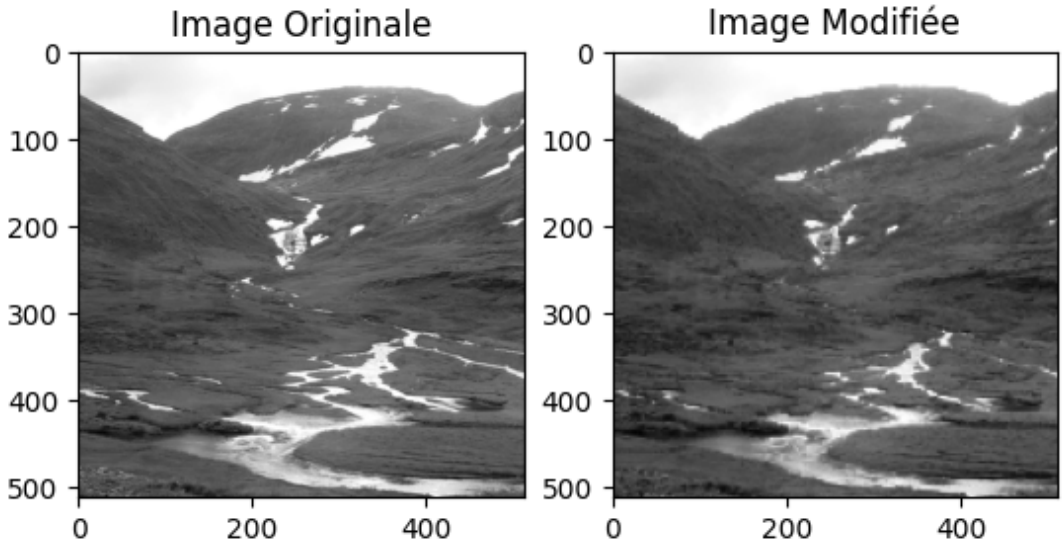


Taille du block $N_b = 10$

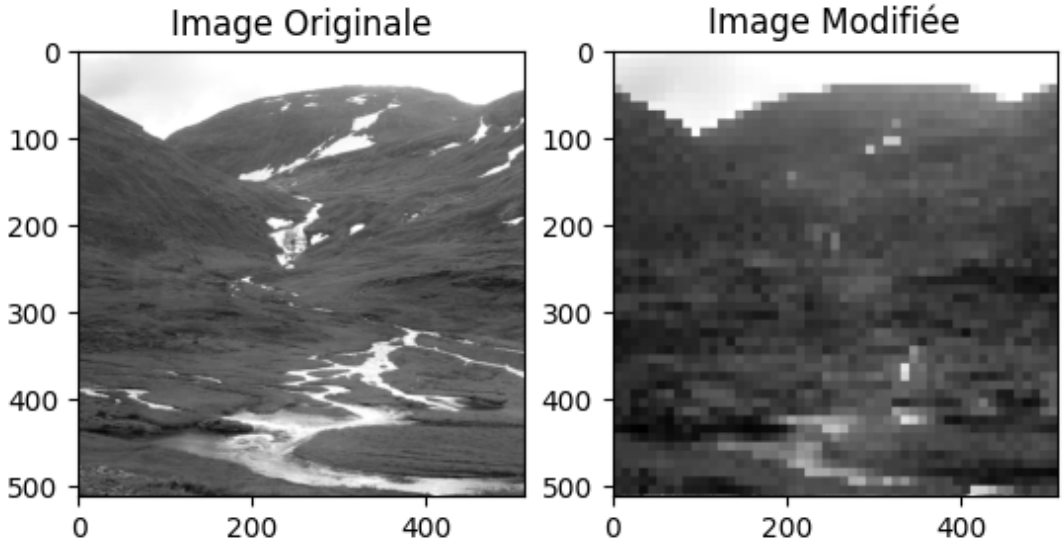


8.1.2 Solveur Exponentiel avec cvxpy

Taille du block $N_b = 3$



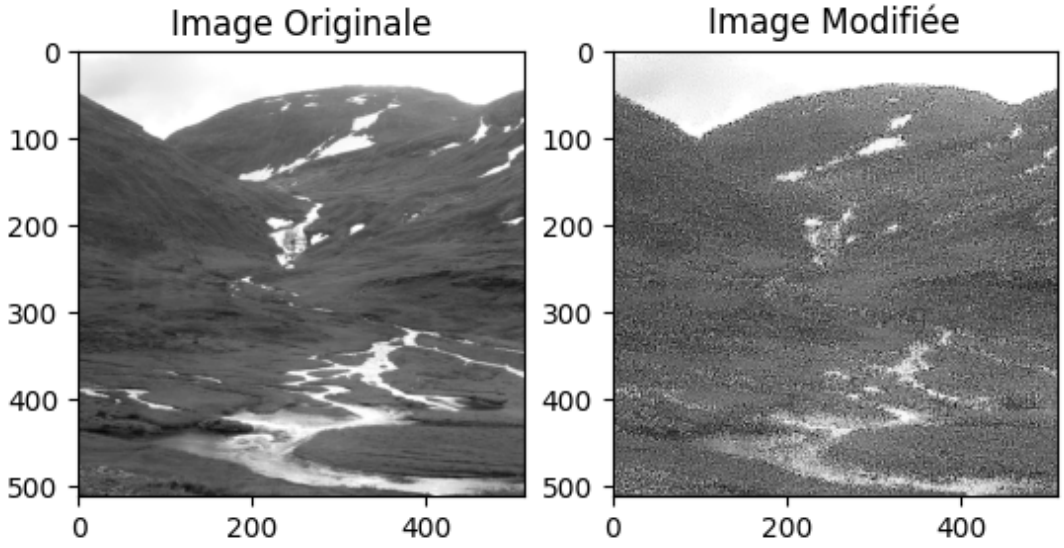
Taille du block $N_b = 10$



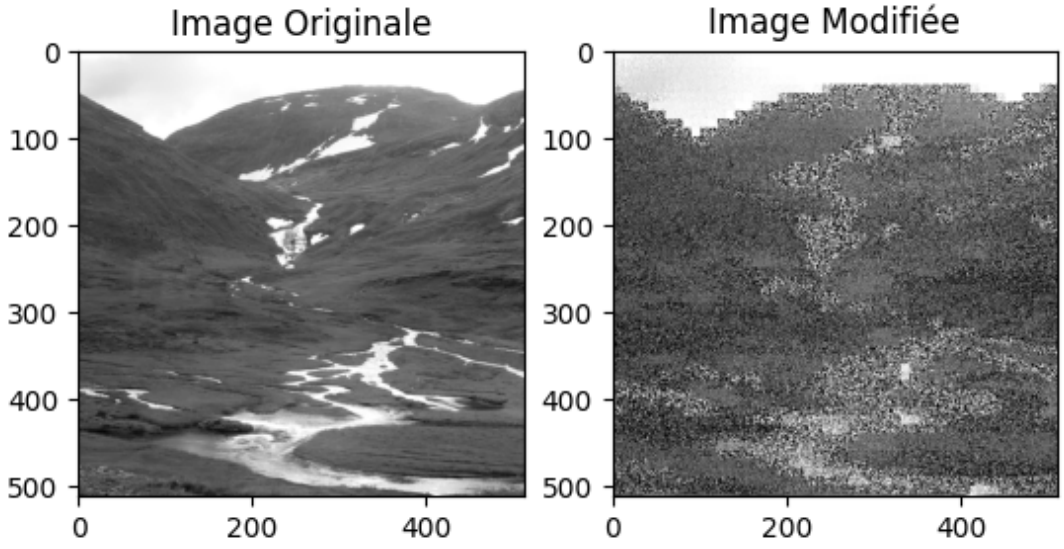
8.2 Segmentation de l'Image en Blocs et Attribution de Cartes de Poids Distinctes

8.2.1 Méthode de Newton

Taille du block $N_b = 3$

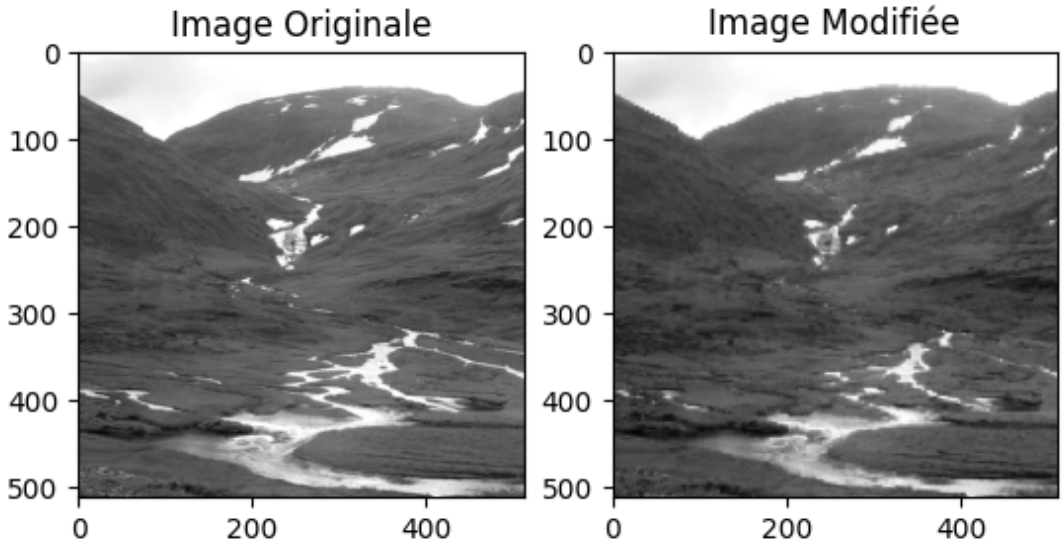


Taille du block $N_b = 10$



8.2.2 Solveur Exponentiel avec cvxpy

Taille du block $N_b = 3$



Taille du block $N_b = 10$