# TP3 - Codage et compression multimédia

## Codage prédictif et code d'Huffman

*Léa Serrano M1 IMAGINE*

Lien de mon git pour ce tp : https://github.com/LeaSerrano/M1-IMAGINE-CompressionDonnees-TP3.git

## Table des matières

**1 Ex 1** **2**

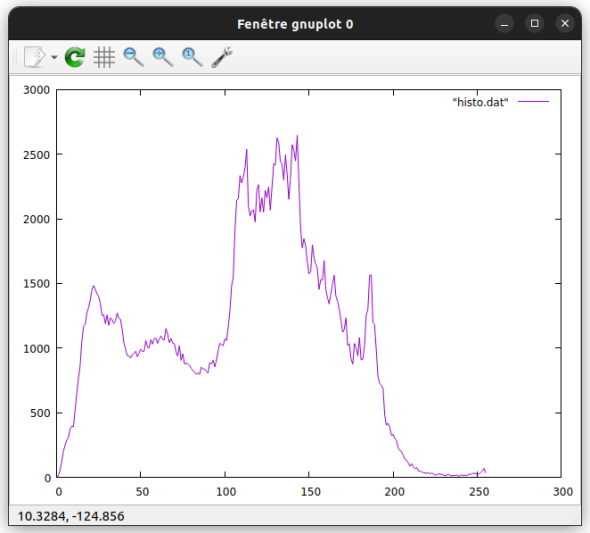
**2 Ex 2** **3**

**3 Ex 3** **5**

# 1 Ex 1

Pour ce TP, nous allons prendre cette photo prise à la Fac des Sciences de Montpellier et transformée en image pgm :

Nous allons commencer par tracer son histogramme et calculer son entropie :




Voici la formule pour calculer l'entropie :

$$H = -\sum p(i) * log2(p(i))$$

```
Donnees-TP3$ ./entropie fac.pgm
L'entropie est de : 7.55741
```

On a une entropie assez haute car notre image comporte beaucoup d'informations.

Ensuite nous allons appliquer l'algorithme de Huffman sur notre image pour obtenir une image compressée. Voici le taux de compression de notre image :

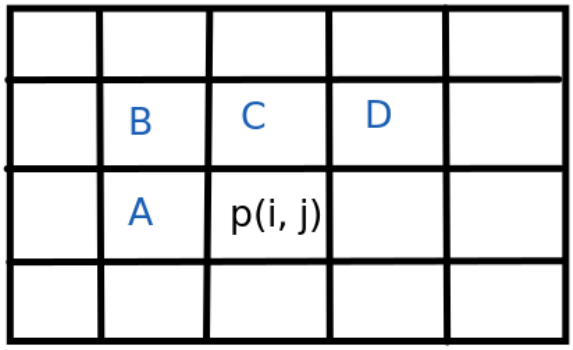
```
Compression en cours...
Compactage effectué en 21655 ◆s. Taux de compression: 1.05
Linux rules!
```

Si on décompresse l'image qu'on a compressée, on obtient bien l'image de base :

| Image de base | Image décompressée |
|---|---|
|  |  |

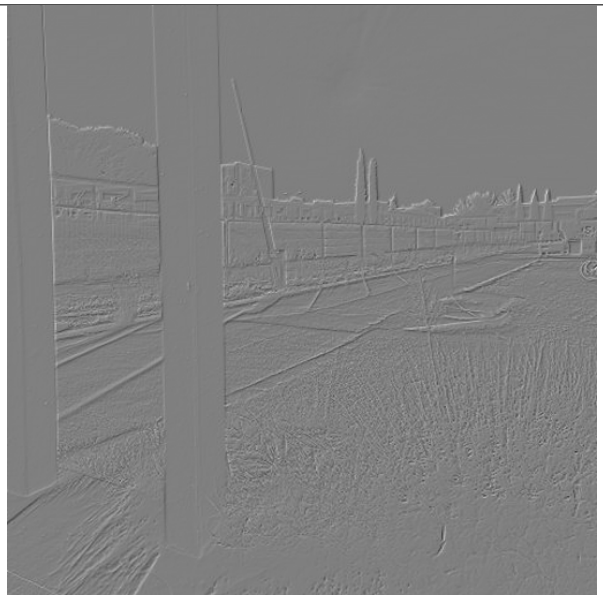
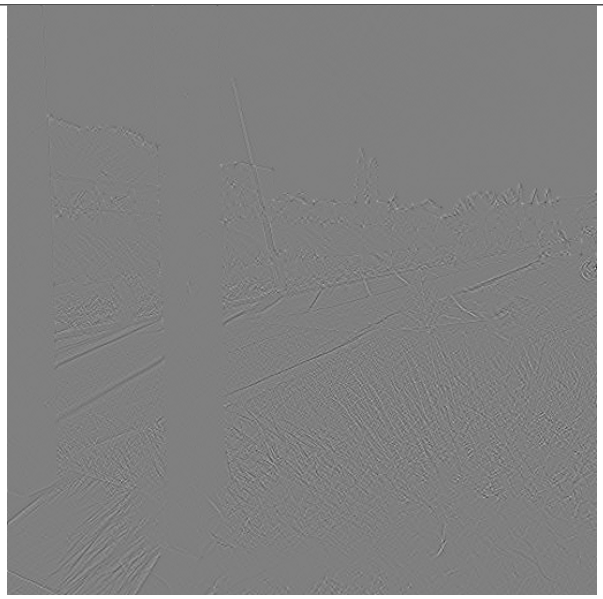
## 2 Ex 2

Nous allons maintenant faire la carte des différences à partir de plusieurs méthodes de prédiction sur les voisins.



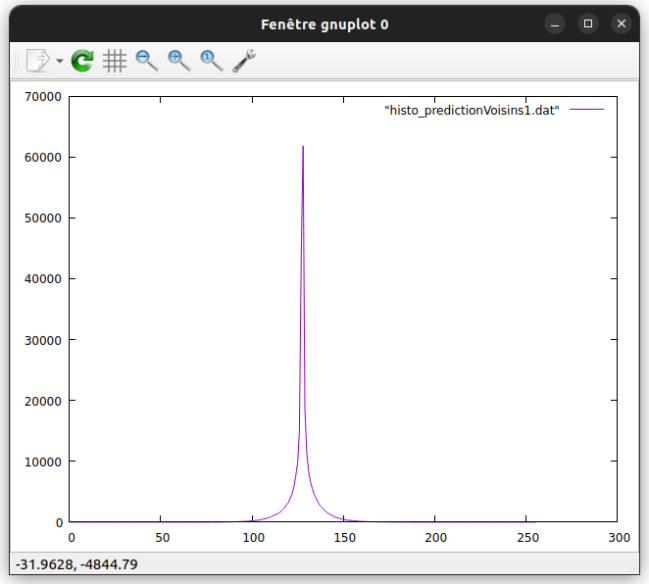

Nous allons utiliser une méthode telle que p(i, j) = (A+C)/2 et une autre telle que p(i, j) = A+C-B.

Nous avons ensuite réalisé l'algorithme de différence avec ces méthodes de prédiction :

| Carte des différences méthode 1 | Carte des différences méthode 2 |
|---|---|
|  |  |

On va ensuite tracer la distribution de la carte des différences ainsi que leur entropie :

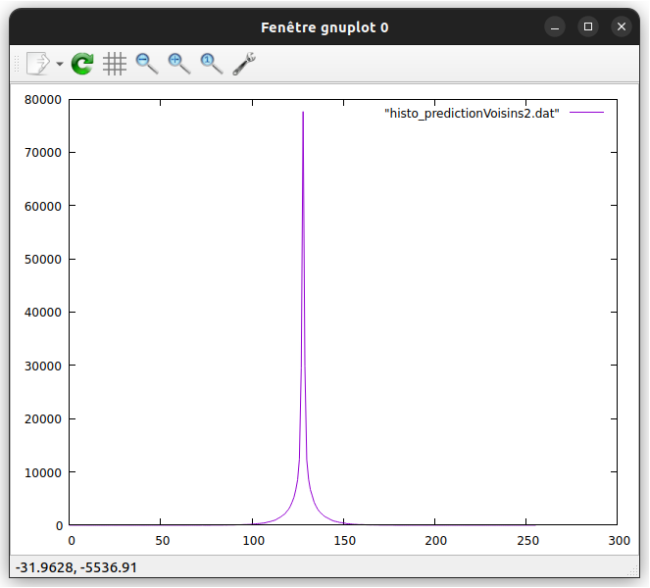
Méthode 1 :




```
Donnees-TP3$ ./entropie carteDiff_predictionVoisins1.pgm
L'entropie est de : 4.35117
```

On a une entropie beaucoup plus petite qu'à l'exercice d'avant.

Méthode 2 :




```
Donnees-TP3$ ./entropie carteDiff_predictionVoisins2.pgm
L'entropie est de : 4.13594
```

On a une entropie encore plus petite qu'avec la méthode 1.

On va ensuite appliquer l'algorithme d'Huffman et obtenir le taux de compression.

Algorithme d'Huffman sur la méthode 1 :

```
Compression en cours...
Compactage effectué en 13877 ♦s. Taux de compression: 1.82
Linux rules!
```

Algorithme d'Huffman sur la méthode 2 :

```
Compression en cours...
Compactage effectué en 13737 ♦s. Taux de compression: 1.91
Linux rules!
```

# 3 Ex 3

Dans l'espace des pixels, notre taux de compression est de 1.05. Dans l'espace de prédiction, pour notre première méthode on obtient un taux de compression de 1.82. Pour la seconde méthode, le taux de compression est de 1.91.
La compression dans l'espace de prédiction est meilleure que celle dans l'espace des pixels.

On peut ensuite essayer de réaliser d'autres méthodes pour essayer de trouver un taux de compression encore plus intéressant en utilisant l'algorithme DPCM ou encore l'algorithme LOCO-I.

— DPCM : L'algorithme DPCM consiste à compresser une image telle qu'un pixel de l'image compressée soit égal à la valeur de ce pixel dans l'image d'origine soustraite à la valeur du pixel précédent dans l'image d'origine.

Voici notre image de la fac lorsqu'on applique notre algorithme dessus :

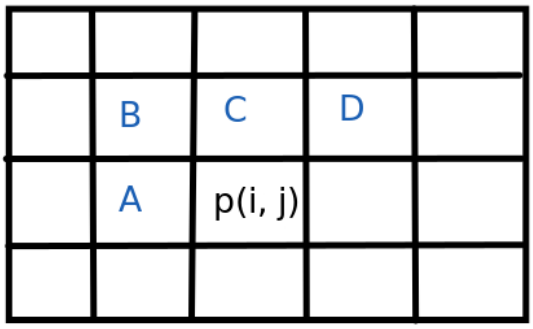
Puis on va mesurer son taux de compression :

```
Compression en cours...
Compactage effectué en 17144 ◆s. Taux de compression: 1.70
Linux rules!
```

Le taux est plutôt satisfaisant.

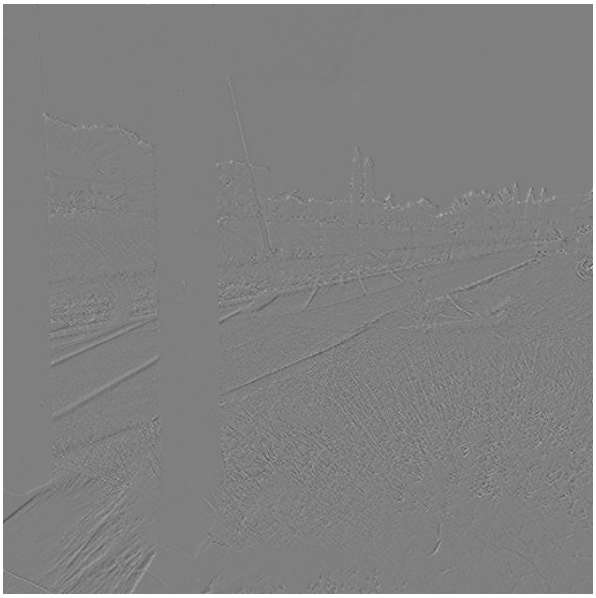
— LOCO-I : L'algorithme LOCO-I consiste réaliser une prédiction sur la valeur de chaque pixel en fonction de la valeur des pixels adjacents. Chaque pixel de l'image compressée aura pour valeur, la différence entre la valeur qui a été prédite et la réelle valeur du pixel.

L'algorithme fonctionne de la façon suivante :




p(i, j) =
min(A, C) si B >= max(A, C)
max(A, C) si B <= min(A, C)
A + C - B sinon

Voici notre image de la fac lorsqu'on applique la carte des différences de notre algorithme dessus :



Puis on va mesurer son taux de compression :

```
Compression en cours...
Compactage effectué en 13004 ◆s. Taux de compression: 1.97
Linux rules!
```

On a un meilleur taux.

Au final dans tout ce qu'on a essayé, le meilleur taux est lorsqu'on utilise l'algorithme LOCO-I sur une image et qu'on réalise sa carte des différences.