

# Projet MAM3 - Compression d'images par transformée de Fourier

Matthew Sanchez - Célia Roess - Adam Sekkat

December 2023

---

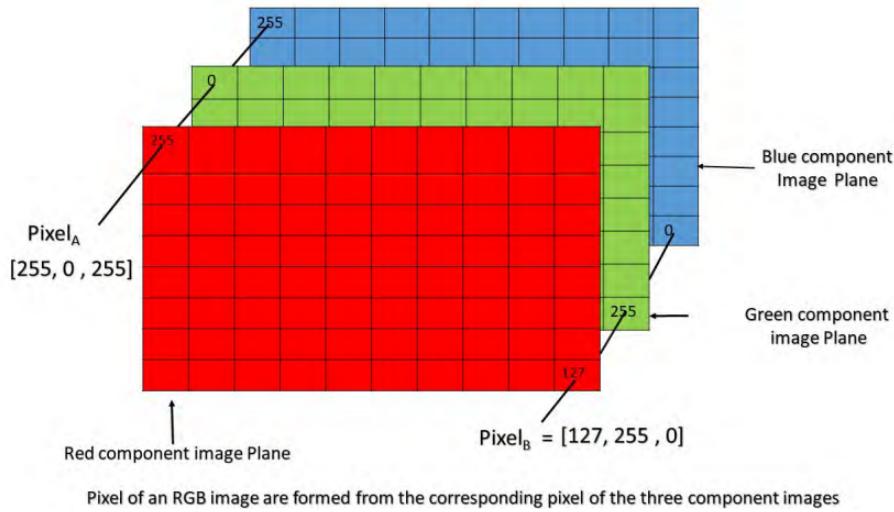


# Sommaire

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Objectifs</b>	<b>3</b>
<b>3</b>	<b>Algorithme</b>	<b>4</b>
3.1	Initialisation . . . . .	4
3.2	Compression . . . . .	6
3.2.1	Utilisation de la matrice de quantification . . . . .	6
3.2.2	Filtrage des hautes fréquences . . . . .	7
3.3	Décompression . . . . .	7
3.4	Post-processing . . . . .	7
3.5	Exécution du programme . . . . .	7
<b>4</b>	<b>Réalisations pratiques</b>	<b>8</b>
<b>5</b>	<b>Conclusion</b>	<b>11</b>
<b>Annexe n°1 : Plan de travail</b>		<b>12</b>
<b>Annexe n°2 : Carnet de bord</b>		<b>13</b>

# 1 Introduction

Une représentation possible d'une image numérique consiste à la décrire à l'aide d'un tableau multidimensionnel. On peut choisir différentes formes de représentation, mais nous nous intéressons ici à la plus élémentaire, qui divise l'image en trois composantes de couleur : Rouge, Vert et Bleu (RGB).



Notre projet consiste en la réalisation d'un code Python permettant la compression et la décompression d'images grâce à la transformée de Fourier.

L'objectif de la compression d'images est de réduire la redondance des données d'une image afin de pouvoir l'emmagerer sans occuper beaucoup d'espace ou la transmettre rapidement.

Il est alors important de connaître le taux de compression ainsi que le taux d'erreur pour pouvoir estimer les pertes d'informations.

La transformée de Fourier a un rôle primordial dans la compression d'images et plus généralement dans le traitement d'images. Dans le but de faciliter les calculs, nous allons appliquer la transformée de cosinus discrète à des blocs 8 x 8 de l'image choisie.

Ainsi, on pourra compresser, décompresser, filtrer les hautes fréquences et débruiter toutes les images.

## 2 Objectifs

Afin de réaliser notre compression d'image, on va procéder par plusieurs étapes distinctes que l'on retrouvera dans notre code final.

Il s'agit de :

1. l'initialisation,
2. la compression,
3. la décompression,
4. le post-processing.

## 3 Algorithme

Lors de l'exécution de notre programme Python, la console nous demandera l'adresse de l'image que l'on souhaite compresser. On conservera cette adresse dans la variable `filepath`.

Afin de mener à bien notre projet, il est nécessaire d'importer de nombreuses bibliothèques :

- `numpy` as `np`
- `matplotlib.pyplot` as `plt`
- `math` as `ma`

### 3.1 Initialisation

Dans l'initialisation, nous avons fait le choix de garder une matrice en 3 dimensions pour pouvoir travailler sur les 3 composantes RGB simultanément.

#### Lire l'image

On utilise la fonction `plt.imread('content/pns_original.png')`, qui crée une matrice qui crée un tableau depuis un fichier image.

L'image peut ensuite être affichée par la fonction `plt.imshow(image_matrix)` avec `image_matrix` le tableau précédemment créé par la fonction `imread`.

#### Identification des différents types d'images

On cherche tout à d'abord à distinguer si l'image est en noir et blanc ou en couleurs. On regarde alors la longueur de `np.shape(image_matrix)` : elle sera égale à 3 si l'image est couleurs, sinon l'image est en noir et blanc.

La deuxième identification effectuée est celle du format de l'image. Les 3 ou 4 derniers caractères de `filepath` sont égaux à `.png` ou `.jpeg` selon le format de l'image.

#### Tronquer l'image en multiples de 8

Toujours dans l'objectif de simplifier les calculs, nous travaillons sur des blocs 8 x 8.

Ainsi, il faut récupérer le nombre de lignes, de colonnes et de canaux de la matrice originale. Pour ce faire, on utilise la fonction `np.shape(image_matrix)` qui nous renvoie un tuple correspondant aux dimensions du tableau donné.

On récupère le plus grand multiple de 8 pour les lignes et les colonnes. On fait alors la division euclidienne de la taille puis on multiplie par 8. On a donc la formule suivante : `newLignes = (lignes//8)*8`.

La dernière ligne correspond au troncage de la matrice de l'image avec la manipulation des tableaux :

```
image_matrix_tronque = image_matrix[0:newLignes, 0:newColonnes, 0:3].
```

#### Transformation des intensités

L'objectif de cette partie est de transformer les intensités RGB de réels compris entre 0 et 1, en entiers centrés compris entre -128 et 127. Les lignes de code suivantes permettent d'avoir ce résultat.

Pour les images en format `.jpeg`, les valeurs de la matrice de l'image sont initialement comprises entre 0 et 255.

```

new_matrix = image_matrix_tronque * 255
centred_matrix = new_matrix - 128
centred_matrix = centred_matrix.astype('int8')

```

### Calcul de la matrice de passage P

On a :

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^7 \sum_{j=0}^7 \cos \frac{(2i+1)k\pi}{16} \cos \frac{(2j+1)l\pi}{16} \quad (1)$$

avec  $C_0 = \frac{1}{\sqrt{2}}$  et  $C_k = 1$  si  $k > 1$

On cherche  $M$  tel que  $D = PMP^T$ .

On a :

$$\begin{pmatrix} D_{0,0} & \dots & D_{0,7} \\ \vdots & \ddots & \vdots \\ D_{7,0} & \dots & D_{7,7} \end{pmatrix} = \begin{pmatrix} P_{0,0} & \dots & P_{0,7} \\ \vdots & \ddots & \vdots \\ P_{7,0} & \dots & P_{7,7} \end{pmatrix} \times \begin{pmatrix} M_{0,0} & \dots & M_{0,7} \\ \vdots & \ddots & \vdots \\ M_{7,0} & \dots & M_{7,7} \end{pmatrix} \times \begin{pmatrix} P_{0,0} & \dots & P_{7,0} \\ \vdots & \ddots & \vdots \\ P_{0,7} & \dots & P_{7,7} \end{pmatrix}$$

On pose :  $B = PM$

D'où :

$$\left. \begin{array}{l} B_{0,0} = \sum_{a=0}^7 P_{0,a} M_{a,0} \\ B_{1,0} = \sum_{a=0}^7 P_{1,a} M_{a,0} \\ B_{0,1} = \sum_{a=0}^7 P_{0,a} M_{a,1} \end{array} \right\} \quad B_{i,j} = \sum_{a=0}^7 P_{i,a} M_{a,j} \quad (2)$$

On a alors :

$$\begin{pmatrix} D_{0,0} & \dots & D_{0,7} \\ \vdots & \ddots & \vdots \\ D_{7,0} & \dots & D_{7,7} \end{pmatrix} = \begin{pmatrix} B_{0,0} & \dots & B_{0,7} \\ \vdots & \ddots & \vdots \\ B_{7,0} & \dots & B_{7,7} \end{pmatrix} \times \begin{pmatrix} P_{0,0} & \dots & M_{7,0} \\ \vdots & \ddots & \vdots \\ P_{0,7} & \dots & P_{7,7} \end{pmatrix}$$

De la même manière, on trouve :

$$\left. \begin{array}{l} D_{0,0} = \sum_{b=0}^7 B_{0,b} P_{0,b} \\ D_{2,1} = \sum_{b=0}^7 B_{2,b} P_{1,b} \end{array} \right\} \quad D_{i,j} = \sum_{b=0}^7 B_{i,b} P_{j,b} \quad (3)$$

D'après les équations (2) et (3), on en déduit :

$$D_{i,j} = \sum_{b=0}^7 \sum_{a=0}^7 P_{i,a} M_{a,b} P_{j,b} \quad (4)$$

$$D_{k,l} = \sum_{a=0}^7 \sum_{b=0}^7 P_{k,a} M_{a,b} P_{l,b} \quad (5)$$

Or, d'après l'équation (1) et par identification des termes, on a :

$$P_{k,i} = \frac{1}{2} C_k \cos \frac{(2i+1)k\pi}{16} \quad (6)$$

$$P_{l,j} = \frac{1}{2} C_l \cos \frac{(2j+1)l\pi}{16} \quad (7)$$

Finalement, d'après l'équation (7), en posant  $l = i$ , on en déduit :

$$P_{i,j} = \frac{1}{2}C_i \cos \frac{(2j+1)i\pi}{16} \quad (8)$$

### Définition de la matrice P avec la fonction PMatrix()

On définit une fonction PMatrix() qui crée la matrice de passage P dont la formule a été donnée précédemment.

Il faut cependant être vigilant à bien distinguer 2 cas distincts pour la première ligne où le coefficient  $C_k$  est différent selon si  $k = 0$  ou  $k \geq 1$

## 3.2 Compression

Selon si l'image est en couleurs ou en noir et blanc, on distingue 2 fonctions pour la compression : dans la fonction spécifique à la compression en couleurs, il faut parcourir chaque matrice pour chaque couleur (RGB).

On décompose la matrice de l'image en tableau contenant des tableaux de taille 8 x 8.

### 3.2.1 Utilisation de la matrice de quantification

On définit la matrice de quantification Q.

### Application du changement de base

On utilise la fonction `np.transpose(P)` pour obtenir la matrice transposée de la matrice de passage P.

Afin de faire les multiplications de matrice, on utilise la fonction :

```
Temp = np.matmul(P, M[:, :, i])
D = np.matmul(Temp, Pprime)
avec i qui parcourt chaque canal de couleur RGB.
```

Dans le cas d'une image en noir et blanc, on a le code suivant :

```
Temp = np.matmul(P, M)
D = np.matmul(Temp, Pprime)
```

### Application de la matrice de quantification

On divise terme à terme la matrice D par la matrice de quantification préalablement définie. On récupère ensuite uniquement la partie entière.

```
D = np.divide(D, Q)
D = np.trunc(D)
```

### Stockage des blocs dans la matrice de compression

Pour créer la matrice compressée et débruitée, on ajoute chaque bloc de 8 x 8 dans une nouvelle grande matrice avec `compressed_tab.append(D)`.

### Comptage des coefficients non nuls

Grâce à la fonction `np.count_nonzeros(D[i])`, on compte le nombre de coefficients non nuls dans les nouvelles matrices pour chaque canal.

On en déduit le taux de compression :

$$\text{taux\_compression} = \frac{\text{volume\_final}}{\text{volume\_initial}} = \left(1 - \frac{\text{nb\_nonNuls}}{\text{tronqueLignes} * \text{tronqueColonnes} * 3}\right) * 100$$

### 3.2.2 Filtrage des hautes fréquences

Afin de réaliser une compression puis décompression par la méthode de filtrage des hautes fréquences, il est avant tout nécessaire de décomposer la matrice en blocs de 8 x 8. On fait ensuite la multiplication terme par terme de  $D = PMP^T$ .

L'étape suivante, spécifique au filtrage des hautes fréquences, consiste à mettre l'indice (i,j) de la matrice D à 0 si la somme de l'indice i et de l'indice j est supérieure ou égale à la fréquence de coupure.

```
if i+j>=threshold:  
D[i,j]=0
```

On réalise ensuite le changement de base inverse pour décompresser l'image. (*cf. 3.3*)

## 3.3 Décompression

On multiplie terme à terme chaque bloc par la matrice de quantification Q, on note  $\tilde{D}$  le résultat.

On applique la transformée inverse  $P^T \tilde{D} P$  avec la fonction `np.matmul()`.

On rajoute chaque nouveau bloc de décompression à une grande matrice de décompression avec la fonction `append()`.

## 3.4 Post-processing

La partie **Post-processing** permet notamment de calculer l'erreur.

L'erreur est calculé selon la fonction suivante (avec `np.linalg.norm()` la norme d'une matrice) :

```
(np.linalg.norm(mat_init)-np.linalg.norm(mat_fin))/np.linalg.norm(mat_init)
```

Dans le cas d'une image en couleur, on calcule l'erreur pour chaque canal de couleur puis, en faisant la moyenne, on obtient l'erreur globale.

Dans le cas d'une image en noir et blanc, on suit simplement la formule donnée ci-dessus.

## 3.5 Exécution du programme

Grâce à la ligne `if __name__ == "__main__"`, il est possible d'exécuter le programme.

Il sera demandé à l'utilisateur de rentrer le nom du chemin pour accéder au fichier image qu'il souhaite compresser puis décompresser. Ce chemin sera conservé dans la variable `filepath`. Le programme va alors calculer la matrice de l'image.

L'utilisateur aura un deuxième choix à effectuer concernant le type de compression/décompression qu'il souhaite : s'il renvoie 1, la matrice de quantification sera utilisée ; s'il renvoie 2, le filtrage des hautes fréquences sera utilisé.

Dans le cas où il demande le filtrage des hautes fréquences, il sera demandé à l'utilisateur de rentrer la fréquence de coupure qu'il souhaite utiliser.

Selon la couleur de l'image (*RGB* ou *noir et blanc*) et le type de compression/décompression demandé, différentes fonctions de compression, décompression et recomposition seront effectuées.

Les dernières étapes de notre programme consistent en l'affichage des données sous la forme d'une image, de l'enregistrement du nouveau fichier image dans le dossier courant et de l'affichage de l'image compressé/décompressé dans une fenêtre Python.

## 4 Réalisations pratiques

Lors de l'exécution de notre code Python, la console nous demande `entered filepath` : à l'aide d'un input. On rentre alors le chemin pour accéder au fichier image que l'on souhaite compresser en indiquant le format retenu.

On teste alors avec le chemin suivant : `pns_original.png` qui correspond à l'image ci-dessous.



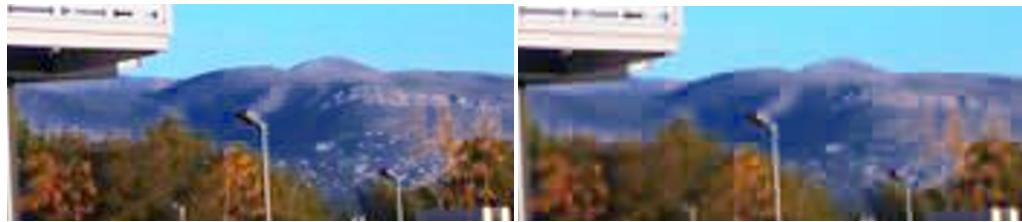
Figure 1: Image originale

En utilisant notre programme pour compresser puis décompresser notre image, on obtient l'image suivante :



Figure 2: Image compressée puis décompressée

Lorsque l'on observe de plus près les détails de nos 2 images, on remarque que la photo qui a été compressée puis décomposée est beaucoup plus pixelisée que l'originale.



(a) Photo originale

(b) Photo compressée/décompressée

Figure 3: Zoom sur les deux photos précédentes



(a) Photo originale

(b) Photo compressée/décompressée

Figure 4: Zoom différent sur les deux photos précédentes

On a également pu tester les fonctions pour une image initialement en format .jpg et en noir et blanc.



Figure 5: Test pour une image en noir et blanc

Le dernier test que l'on a effectué est celui de la compression/décompression par filtrage des hautes fréquences. On obtient alors les images suivantes :



Figure 6: Photo de Polytech - Originale et filtrage de hautes fréquences  $\leq 6, 4, 2$

## 5 Conclusion

Pour conclure, ce projet nous a permis de réaliser que la transformée de Fourier, et plus précisément, la transformée de cosinus discrète, est très efficace dans le traitement d'images.

Après avoir défini la matrice de l'image, nous avons appliqué la transformée de Fourier pour compresser l'image, puis nous avons effectué la transformée de Fourier inverse pour décompresser l'image.

L'objectif de la compression d'images est de minimiser la redondance des données présentes dans une image, permettant ainsi de stocker celle-ci de manière plus compacte ou de la transmettre de manière plus rapide.

On a pu remarquer que lors de la compression puis décompression d'une image, on obtient une nouvelle image de qualité toujours plutôt correcte. En effet, on n'observe pas de grandes différences de façon générale. C'est seulement lorsque l'on zoomme que l'on peut voir que la nouvelle image est beaucoup plus pixelisée.

## Annexe n°1 : Plan de travail

### Titre et description du projet

Compression d'images par transformée de fourier

### Buts et objectifs du projet

L'intérêt de la compression d'image est multiple. Elle permet de réduire la Grâce à la transformée de cosinus discrète, nous allons pouvoir compresser puis décompresser plusieurs types d'images. L'objectif est de traiter différents cas : l'image en couleur RGB, l'image en noir et blanc, le format .jpeg ou encore le format .png.

On enregistre ensuite le nouveau fichier dans le dossier courant.

### Portée et limites du projet

Notre programme n'est fonctionnel que pour les images de format .jpeg ou .png.

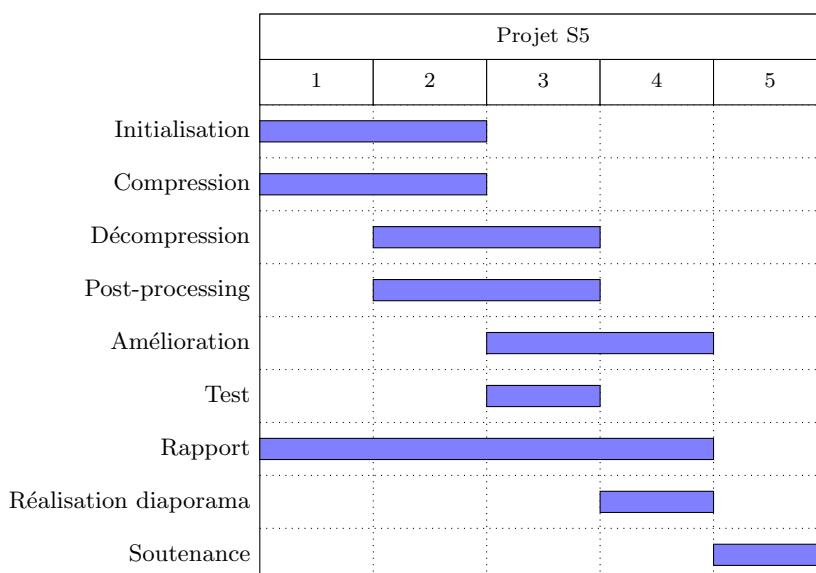
Un problème que l'on a actuellement avec notre projet est qu'il enregistre tous les fichiers images nouvellement créés sous le même nom : 'pfig1.png'.

Une amélioration possible de notre projet serait la réalisation d'une interface graphique où l'utilisateur peut aller sélectionner le fichier image qu'il souhaite dans ses documents à l'aide d'un bouton.

Il pourrait également être intéressant de créer une fonctionnalité à travers un bouton permettant à l'utilisateur d'enregistrer la nouvelle image compressée/décompressée et ainsi de choisir le nom qu'il souhaite lui donner.

### Calendrier et échéancier du projet

Date de fin de projet : 22/12/2023



## Annexe n°2 : Carnet de bord

### Lundi 18 décembre 2023 - matinée

Amphithéâtre de présentation du sujet du projet

### Lundi 18 décembre 2023 - après-midi

Matthew : Calcul de la matrice de passage P

Célia : Rédaction du rapport de projet

Adam : Codage des parties **Initialisation** et **Compression**

### Mardi 19 décembre 2023 - matinée

Matthew : Commentaires et déboggage du code

Célia : Apprentissage du codage en LATEX

Adam : Codage de la partie **Décompression** et test

### Mardi 19 décembre 2023 - après-midi

Matthew : Codage de la partie **Filtrage des hautes fréquences**

Célia : Optimisation du début du code et rédaction du rapport

Adam : Codage de la partie **Recomposition d'image** et test

### Mercredi 20 décembre 2023 - matinée

Matthew : Amélioration de la partie **Filtrage des hautes fréquences**

Célia : Déboggage du script Python et rédaction du rapport

Adam : Commentaires et création du script Python avec le traitement des différents cas (couleurs, format) + création du notebook

### Mercredi 20 décembre 2023 - après-midi

Matthew : Rédaction claire de la démonstration de la matrice de passage P

Célia : Rédaction du code final et codage de la fonction pour changer le nom du nouveau fichier à chaque exécution

Adam : Codage de l'input pour demander le choix de méthode de compression/décompression

### Jeudi 21 décembre 2023 - matinée

Matthew : Réalisation du diaporama

Célia : Finalisation du rapport

Adam : Réalisation du diaporama

### Jeudi 21 décembre 2023 - après-midi

Préparation de l'oral