# 01 - Le projet PGM

Lionel Bastard & Jean-Christophe Toussaint

2 juin 2025

Le but de ce projet est de développer une application qui permet de lire, écrire et manipuler des images au format PGM.

## 1 Introduction

Une image peut être stockée sous la forme d'un tableau de points élémentaires appelés pixels (abréviation de picture elements). On s'intéressera dans un premier temps à la manipulation d'images en niveaux de gris (ou grayscale en anglais), c'est-à-dire, des images dont les "couleurs" sont uniquement des nuances de gris. On peut représenter une telle image par une matrice d'entiers, dont la valeur des éléments représente l'intensité lumineuse des pixels de l'image. Par conséquent, un traitement d'image peut être réalisé en manipulant la matrice qui la représente.

## 2 Fichier PGM

Un fichier PGM (Portable Gray Map) est composé de deux parties :

1. Un entête indiquant les paramètres de l'image
2. Une matrice dont chaque élément représente le niveau de gris d'un pixel de l'image.

L'entête contient 3 lignes. La première ligne est toujours la même, il s'agit de l'identifiant du fichier PGM : P2 pour une image codée en ASCII et P5 pour une image codée en binaire. La seconde ligne contient 2 entiers séparés par un espace : le nombre de colonnes et le nombre de lignes de pixels de l'image.

```
P2
10 6
255
0 114 255 114 0 0 0 0 0 0
0 0 123 255 123 0 0 0 0 0
0 0 0 112 255 101 0 0 0 0
0 0 0 0 115 255 103 0 0 0
0 0 0 0 0 111 255 121 0 0
0 0 0 0 0 0 145 255 135 0
```

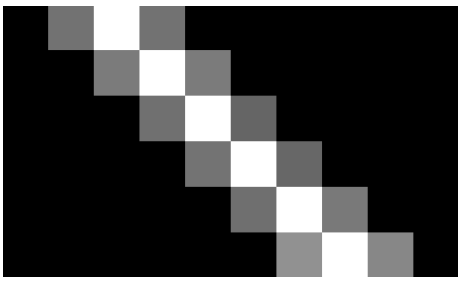

FIGURE 1 – Exemple de fichier PGM et image associée.

Enfin, la troisième ligne contient la valeur maximale que peut prendre un pixel (par exemple : 255 pour un codage des niveaux de gris entre 0 et 255). Les lignes qui suivent donnent l'intensité

de chaque pixel. Ceux-ci sont énumérés ligne par ligne et colonne par colonne. Ainsi la première valeur est l'intensité du pixel du coin supérieur gauche, la seconde est l'intensité du pixel de la première ligne, deuxième colonne, etc.

Par exemple, un fichier PGM contenant les informations ci-dessous est l'encodage d'une image représentant une barre oblique blanche sur fond noir (voir Fig. 1).

# 3 Organisation du programme

Pour travailler sur une image (modifier la valeur des pixels pour changer le contraste de l'image par exemple), il est nécessaire de dissocier deux entités :

1. Le fichier contenant l'image, qui est stocké sur disque dur.
2. La représentation en mémoire de l'image, qui est stockée dans la mémoire vive allouée au programme. C'est sur cette représentation mémoire que l'on effectuera toutes les manipulations de l'image. Une fois l'image modifée à notre convenance, il sera possible de la stocker dans un fichier via une fonction dédiée.

Dans ce projet, vous stockerez l'image en mémoire sous la forme d'un tableau numpy de taille NROWS × NCOLS où NROWS est le nombre de pixels en largeur et NCOLS celui en hauteur. Ce tableau comprendra l'ensemble des pixels de l'image. Par ailleurs, pour minimiser la place mémoire, le tableau sera constitué d'élément du type `uint8`, qui occupent un octet de mémoire seulement. Seules des images contenant au maximum 256 niveaux de gris pourront donc être traitées par notre programme. Le code suivant donne un exemple de création et d'utilisation du tableau contenant les valeurs des pixels de l'image.

On rappelle que

1. A=np.zeros(shape=(NROWS, NCOLS), dtype='uint8') crée un tableau de zéros,
2. U=A.astype(np.uint8) force la conversion de type d'un tableau de réels $R$ en un d'entiers non signés U.

Votre programme devra lire et écrire des images au format PGM au format ASCII (texte). Dans le fichier ASCII de l'image, les données de l'intensité lumineuse des pixels sont écrites sous forme d'un entier, chaque entier étant séparé par un espace vide au moins, et chaque ligne de l'image étant séparée par un retour à la ligne.

# 4 Travail à réaliser

On vous demande d'écrire sous Spyder, l'ensemble de fonctions suivantes permettant de générer, lire, écrire et manipuler les images au format PGM.

## 4.1 Version scalaire

1. créer une fonction `fun(x, y)` qui à partir d'un couple de réels (x, y) retourne l'évaluation de $\frac{1}{2}(\sin(x)\ \sin(y)+1)$. La valeur retournée est dans $[0,1]$.
2. créer une fonction `centering(i, L)` qui à partir de deux entiers $i$ et $L$ retourne $2i/L-1$. La fonction vérifiera que $i \in [0, L[$.
   On pourra mettre en oeuvre l'instruction suivante :
   `assert condition, "out of range"` en adaptant la condition à votre usage.
3. créer une fonction `createImg(fun, NROWS, NCOLS, NLEVELS=255)` qui retourne un tableau numpy `img` de taille NROWS × NCOLS contenant les valeurs des pixels comprises entre 0 et 255.
   Pour tout couple d'indice $(i,j) \in [0, NROWS[\times[0, NCOLS[$, définir `x=`$\pi$ `centering(i, NROWS)` et `y=`$\pi$ `centering(j, NCOLS)` et stocker dans l'élément $(i,j)$ la valeur de `NLEVELS` × `fun(x, y)` dont on ne gardera que la partie entière. Pour ce faire utiliser la fonction `floor` du module math.

4. créer une fonction `writeImg(img, filename)` qui sauvegarde dans le fichier `filename` l'image `img` au format *PGM* ASCII.
5. Proposer un petit programme test sur une petite image. Vérifier avec un éditeur de texte que le fichier a la bonne structure. Tester-le en prenant des valeurs de NROWS et NCOLS différentes !

   On pourra utiliser un petit programme comme OpenSeeIt sous windows ou eog sous linux pour visualiser l'image stockée.
6. Un de vos collègues vous a fourni une fonction `readImg(filename)` qui lit le fichier PGM `filename` et qui retourne un tuple, en omettant de la documenter ! A vous de le faire maintenant.

```
1  def readImg(filename):
2      with open(filename) as f:
3          lines = f.readlines()
4
5  #to be commented
6      for l in list(lines):
7          if l[0] == '#':
8              lines.remove(l)
9
10 #to be commented
11     assert lines[0].strip() == 'P2'
12
13 #to be commented
14     data = []
15     for line in lines[1:]:
16         data.extend([int(c) for c in line.split()])
17
18 #to be commented
19     return (np.array(data[3:]),(data[1],data[0]),data[2])
```

   Quelle est la structure du tuple retourné ? Où se trouve le tableau contenant l'image, quelles sont ses dimensions ? A quoi correspond le triplet (data[1],data[0]),data[2]) ?
7. Compléter le programme test en lisant une image PGM stocker dans un fichier en utilisant la fonction précédente `tup = readImg(filename` puis en insérant la portion de code suivante :

```
1 plt.imshow(np.reshape(tup[0],tup[1]), cmap=cm.jet)
2 plt.colorbar()
3 plt.show()
```

   N'oublier pas d'insérer en tête de programmes les modules suivants :

```
1 import matplotlib.pyplot as plt
2 import matplotlib.cm as cm   # colormap
```

   En vous référant à la documentation de la fonction `plt.imshow`, expliquer que fait la fonction numpy `np.reshape(tup[0],tup[1])` ?
8. On désire mesurer le temps CPU nécessaire pour générer une image de taille $512 \times 512$. Utiliser la portion de code suivant :

```
1 from time import process_time
2
3 # Start the stopwatch / counter
4 t1_start = process_time()
```

```
5
6  Z=createImg(fun, NROWS, NCOLS)
7
8  # Stop the stopwatch / counter
9  t1_stop = process_time()
10 print("Elapsed time (s):", t1_stop-t1_start)
```

Noter le temps écoulé.

## 4.2 Version vectorielle

On complète notre programme test avec des versions vectorisées des fonctions `fun` et `createImg`.

1. créer une fonction `createImgVec(funvec, NROWS, NCOLS, NLEVELS=255)` qui utilisera la fonction numpy `np.meshgrid(x, y)` où x est un tableau numpy de NROWS valeurs équi-espacées entre $[-\pi, \pi]$ et y de NCOLS valeurs équi-espacées entre $[-\pi, \pi]$. La fonction `funvec` mettra en oeuvre les versions vectorielles numpy de la fonction sinus.

   La documentation de `np.floor` nous informe que la valeur retournée est un réel. L'image étant codée avec des entiers non signés 8 bits, il faut donc convertir le type en utilisant la méthode `astype`, vue dans la section 3.
2. mesurer de nouveau le temps écoulé pour créer une image de taille $512 \times 512$ avec la fonction `createImgVec`. Que remarquez-vous ?
3. développer une fonction de seuillage `threshold(img, seuil)` où seuil est un entier compris entre 0 et NLEVELS. Cette fonction remplace toute valeur supérieure au seuil par cette valeur de seuil. Montrer qu'il est possible d'écrire le corps de cette fonction en une ligne. *indication : faire une recherche sur internet en tapant numpy array with condition.*

## 4.3 BONUS : stockage des images en format binaire 'P5'

On dupliquera le projet précédent avant de modifier toutes fonctions.

1. On désire modifier la fonction `writeImg(img, filename)` pour stocker l'image en niveau de gris dans un fichier au format binaire 'P5'.

   Pour écrire l'entête ASCII du fichier, on utilisera la fonction python `open(filename, 'w')` en mode écriture (**w**rite). Puis, on le reouvrira en mode ajout (**a**ppend) avec la fonction python `open(filename, 'ab')` pour le compléter. Noter que la conversion d'une variable numérique z en octets peut être réalisée avec la fonction membre `z.to_bytes`.
2. Proposer un petit programme test sur une petite image.
3. On désire modifier la fonction `readImg(filename)` qui lit le fichier PGM `filename` au format binaire 'P5'. Noter que la conversion d'octets en valeur numérique peut être effectuée avec la fonction membre `int.from_bytes`.