# 1 Représentation d'une image

## 1.1 Représentation vectorielle ou bitmap

Pour représenter sur ordinateur l'image d'un disque noir on peut imaginer plusieurs procédés.

1. On peut dire à l'ordinateur qu'on veut tracer un cercle de centre O et de rayon R et préciser les coordonnées de O et la valeur du rayon. Dans ce cas il faut disposer d'un logiciel avec les fonctions nécessaires (tracé de cercle, remplissage).
   De même, la plupart des polices de caractères affichées par un ordinateur sont représentées à l'aide de courbes.

   **On parle alors de représentation vectorielle d'une image**.

   Question : Cette représentation d'une image à l'aide d'outils géométriques, même sophistiqués (courbes de Bézier), est-elle adaptée à tout type d'image ?

2. Une autre méthode consiste à superposer un quadrillage à l'image comme dans la figure 1.

   L'image est divisée en $nb_{lignes} \times nb_{colonnes}$ cellules appelées **pixel** (pour **picture element**). On associe alors une couleur à chaque pixel selon divers encodages.

   On peut voir l'image comme un tableau de perles de couleurs, chaque perle correspondant à un pixel. Les images captées par les appareils photo numériques sont représentées ainsi.

   **On parle alors de représentation matricielle ou bitmap d'une image**.
   Le nombre de pixels d'une image bitmap est sa **définition**.
   La **résolution** d'une image bitmap s'exprime en ppi (pixels per inch), c'est le rapport entre sa définition et la dimension réelle de sa représentation sur support physique (papier ou écran).
   On a la formule : $\textbf{résolution} = \dfrac{\textbf{définition}}{\textbf{dimension}}$

   Par exemple, pour représenter sur ordinateur un disque noir de diamètre 9 tracé au centre d'une feuille de dimensions $13 \times 13$, on superpose au disque un grille de $13 \times 13$ pixels et on noircit les pixels recouvrant le disque. Ensuite on représente la grille de pixels par un tableau de nombres carré $13 \times 13$ en codant noir par 1 et blanc par 0.

   Cette méthode **discrétise** l'image et la représente par **l' échantillonnage** des couleurs en un nombre fini de pixels.

   *Noircir ci-dessous les pixels recouvrant le disque de centre O.*

**Exercice 1**

1. Calculer la résolution d'une image bitmap carrée de côté 10 cm et de définition $800 \times 800$.
2. Quelle est la définition en pixels d'une image bitmap d'une largeur de 8,5 pouces sur une hauteur de 11 pouces en 300 ppi ?

## 1.2 Représentation d'une image en noir et blanc

### 1.2.1 Représentation vectorielle

**Exercice 2** *format SVG*

On donne ci-dessous un code de représentation d'un disque noir en `SVG` (Scalable Vector Graphics) qui est un format ouvert d'image vectorielle spécifié par le World Wide Web Consortium (`http://www.w3.org/`). Le W3C est un organisme de normalisation à but non-lucratif fondé par Tim Berners-Lee créateur du HTML, du protocole HTTP, des adresses Web. Le W3C est chargé de promouvoir les technologies du Web telles que HTML, XML, CSS, PNG, SVG ...

1. Ouvrir le fichier disquenoir.svg avec le navigateur Firefox.
2. Quelle est la ligne de code permettant de tracer le disque ?
   Modifier cette ligne pour tracer un carré rouge de côté 300 pixels.
   On pourra consulter `http://www.siteduzero.com/tutoriel-3-14858-le-svg.html`

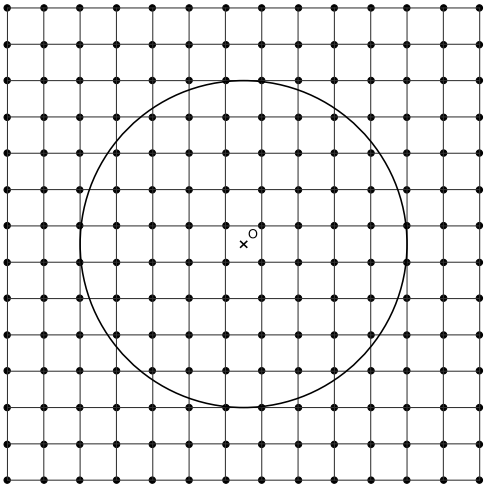


FIGURE 1: Figure 1 : Image bitmap

**3.** Les lignes 1 à 4 représentent l'en-tête de l'image avec la spécification du format. Que représentent les lignes 6 et 8 ?

**4.** Agrandir ou réduire l'image avec le zoom. Le contour du disque est-il moins régulier ?

**5.** Rechercher sur internet d'autres formats vectoriels de représentation d'image.

```
1  <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2  <svg width="800px"
3       height="800px"
4       xmlns="http://www.w3.org/2000/svg">
5  <circle cx="400px" cy="400px" r="100px" fill="black" />
6  <title>Disque noir en SVG </title>
7   <desc>
8  <Creator>Frederic JUNIER</Creator>
9  </desc>
10 </svg>
```

disquenoir.svg

### 1.2.2 Représentation `bitmap`

**Exercice 3** *format PBM*

On donne ci-dessous le code de la représentation bitmap d'une image représentant un disque noir sur un fonds blanc. C'est une image matricielle de format PBM qui permet de coder dans un fichier texte des images en noir et blanc.
Les lignes précédées d'un # sont des commentaires.

1. Que représentent les valeurs de la ligne 4 ?
2. Ouvrir le fichier `disquebnoir.pbm` avec le logiciel `Gimp`. Sachant que la résolution moyenne d'un écran 4 : 3 de diagonale 19" avec une définition de $1024 \times 768$ est d'environ 75 ppi, expliquer la taille de l'image affichée.
   Agrandir cette image avec `SHIFT+`. L'effet observé sur le contour du disque est une pixellisation.
3. Remplacer P1 par P2 sur la ligne 1 puis rafraîchir l'image avec Fichier>Rétablir. Que se passe-t-il ? Remettre P1 pour la suite.
4. Modifier l'image pour représenter un carré noir de côté 9 pixels puis la lettre I dans un cadre de dimensions $Largeur \times Hauteur = 5 \times 10$. Peut-on représenter un carré rouge avec ce format ?

```
1  P1
2  #Createur : F.JUNIER
3  #Titre : Disque noir
4  13 13
5  0000000000000
6  0000000000000
7  0000111110000
8  0001111111000
9  0011111111100
10 0011111111100
11 0011111111100
12 0011111111100
13 0011111111100
14 0001111111000
15 0000111110000
16 0000000000000
17 0000000000000
```

disquenoir.pbm

## 1.3 Représentation d'une image en niveaux de gris

*Dorénavant nous travaillerons principalement sur des formats `bitmap`.*

**Exercice 4** *format PGM P2*

Le fichier `niveauxgrisP2.pgm` contient la représentation bitmap d'une image en niveaux de gris au format P2. C'est un fichier texte ici encodé en ANSI. Les pixels sont codés ligne par ligne, de haut en bas et de gauche à droite.

Pour représenter une image niveaux de gris, on choisit une valeur maximale, par exemple 255, pour exprimer les niveaux de gris et on associe à chaque pixel un nombre compris entre 0 et 255 pour coder les différences nuances de gris.

1. Ouvrir le fichier `niveauxgrisP2.pgm` avec un éditeur de texte pour accéder au code puis avec `Gimp` pour afficher l'image.
2. Lire l'en-tête qui occupe les lignes 1 à 4 avec le nombre magique P2 à la ligne 1, les dimensions de l'image (largeur puis hauteur séparées par un espace) à la ligne 3 et la ligne 4 le nombre maximal de niveaux de gris à la ligne 4.
3. Les pixels de l'image sont codés de la ligne 6 à la ligne 13.
   Comment est codé le noir ? le blanc ? Comment un gris foncé ? un gris clair ?

**Exercice 5** *format PGM P5*

1. Les fichiers `flecheP5.pgm` contient la représentation bitmap d'une flèche en niveaux de gris au format P5. C'est un fichier encodé en binaire.
   a. Ouvrir le fichier avec Gimp pour afficher l'image (agrandir).
   b. On veut accéder au code. Ouvrir le fichier avec un éditeur de texte. Est-ce qu'on peut lire l'en-tête du fichier ? Les valeurs de codage des pixels ?

**c.** Ouvrir maintenant le fichier avec un éditeur hexadécimal comme Okteta (sous Linux) ou HxD (sous Windows) en réglant l'affichage à 18 colonnes.
L'affichage est scindé en deux parties : à gauche on visualise les octets codés en hexadécimal et à droite la correspondance au format texte.
Les valeurs des octets sont affichées à partir de la ligne 4.
Repérer le contour de la flèche, les zones homogènes en teinte et déterminer les valeurs décimales des nuances de gris utilisées dans cette image.

**d.** Modifier la valeur des deux octets correspondant à la pointe de la flèche pour qu'ils s'affichent en gris clair puis rétablir l'image dans `Gimp`.

**e.** Modifier en P2 le nombre magique P5 puis rétablir l'image dans GIMP, que se passe-t-il ? Observer de même l'effet d'une modification des dimensions de l'image dans l'en-tête.

**f.** En parcourant l'en-tête du fichier, retrouver le codage ANSI en hexadécimal puis en décimal des caractères 'P', 'E', 'e' et de l'espace.

**2.** Le fichier `niveauxgrisP5.pgm` représentent la même image que `niveauxgrisP2.pgm` et avec un en-tête de même taille.

**a.** Ouvrir ces deux fichiers avec un éditeur hexadécimal et Gimp.
Quel fichier compte le plus d'octets ? Vérifier avec clic droit > Propriétés.

**b.** Repérer les octets d'en-tête et ceux codant les neuf pixels de l'image. Comment s'explique cette différence de taille de fichier ?

## 1.4 Représentation d'une image en couleurs

Pour les images matricielles, il existe plusieurs types de représentation de la couleur :

- **Chaque pixel peut être codé sur 3 octets appelés canaux (Rouge,Vert,Bleu) notés aussi `(R,G,B)`** : il existe 256 valeurs possibles par canal. On peut ainsi représenter $\left(2^8\right)^3 \approx 16 \times 10^6$ couleurs. Cette représentation s'appuie sur un modèle physique suivant : notre oeil contient des cellules, des cônes de trois sortes, qui sont sensibles respectivement à la longueur d'onde de la lumière rouge, verte ou bleue. Quand notre oeil reçoit une lumière monochrome jaune, les cônes sensibles au rouge et au vert réagissent beaucoup et ceux sensibles au bleu très peu, comme s'il recevait un mélange de lumières monochromes rouges et vertes. Mais si notre oeil ne reçoit pas de lumière, notre cerveau perçoit du noir. C'est comme si les couleurs que nous percevons résultaient d'une synthèse additive de rouge, vert et bleu sur le fonds noir de notre oeil.
  De même sur le fonds noir de l'écran d'un ordinateur, chaque pixel est sensible à trois sources de lumière rouge, verte et bleue qui sont émises avec des intensités variables selon la couleur qu'on veut créer par **synthèse additive**.
  En général, les pixels codés en `(R,G,B)` sont notés en hexadécimal, par exemple la valeur `(R,G,B)` d'un pixel jaune (=rouge+vert) sera `FFFF00`.
  On peut augmenter le nombre d'octets par canal Avec 1 octet par canal, on a un codage RGB sur 24 bits ; avec 2 octets par canal, on a un codage RGB sur 48 bits.
  Le nombre d'octets codant un pixel s'appelle **la profondeur de l'image**.
  Dans certains formats, on rajoute quatrième canal dit alpha pour gérer la transparence du pixel, on parle de codage `RGBA`.
  L'inconvénient de la représentation `(R,G,B)` est sa taille : par exemple une image de 3000 × 3000 pixels au format BMP (Windows bitmap) avec une représentation non compressée sur 3 canaux `(R,G,B)` occupera 27 000 000 d'octets.
  Un formats d'image avec compression de données permet de réduire la taille du fichier image : **une compression peut s'effectuer sans perte ou avec perte** de données. Le taux de compression est plus fort dans un algorithme avec pertes et ces algorithmes exploitent les limites de notre perception : les données perdues peuvent être imperceptibles à l'oeil (photos au format JPG).

- **On peut représenter la couleur avec un seul octet par pixel : les 256 valeurs d'un octet correspondent à une palette dans l'en-tête du fichier.** La palette est restreinte à 256 couleurs mais les fichiers images sont plus légers. Le format `GIF` (*Graphics Interchange Format*) utilise un tel système avec en plus une compression sans pertes grâce à l'algorithme de compression LZW.

- **Il existe d'autres systèmes de représentation de la couleur que le `(R,G,B)`, comme le système `TSL` (Teinte Saturation Luminance)**. Ce dernier est mieux adapté à notre perception visuelle qui est plus sensible à la luminosité et à la teinte (chrominance) qu'au degré de rouge, vert ou bleu. Le format d'image compressé `JPG` (*Joint Photographic Experts Group*), très employé en photographie, utilise le système `TSL`.

**Exercice 6**

1. Faire une recherche internet sur les modes de représentation chromatiques RGB, CMJN (Cyan Magenta Jaune Noir) et TSL (Teinte Saturation Luminance).
2. Calculer le nombre de couleurs disponibles dans un format bitmap BMP qui a une profondeur de 24 bits.
   Calculer le poids approximatif en octets d'une image $800 \times 600$ dans ce format.
3. Ouvrir le logiciel `Gimp`, sélectionner dans la boîte à outils l'outil `Modification de la couleur de premier plan` et cliquer sur l'îcone `Gimp`.
   Sélectionner un canal R, V ou B (T et S représentent respectivement la teinte et la saturation), fixer sa valeur puis jouer sur les valeurs des autres en déplaçant la souris dans la palette.
   En dessous Gimp affiche la notation HTML en hexadécimal de la couleur.

### 1.5 PPM un exemple de format bitmap avec codage RGB

**Exercice 7**

1. Le fichier `rgbP3.ppm` contient une palette de $L \times H = 3 \times 2$ couleurs représentés en bitmap RGB dans un fichier texte.
   L'en tête du fichier occupe les lignes 1 à 4 : les caractères magiques P3 en ligne 1, un commentaire en ligne 2, la largeur puis la hauteur en ligne 3 (séparées par un espace), la valeur maximale d'un canal en ligne 4.
   **a.** Ouvrir le fichier avec un éditeur de texte et avec Gimp. Repérer l'en tête et le codage des neuf pixels.
   **b.** Relever le codage RGB des différentes couleurs.
2. Sachant que le jaune s'obtient en mélangeant à parts égales du rouge et du vert, comment peut-on obtenir de l'orange ?
   Dans un éditeur de texte, créer un fichier représentant selon le format PPM P3 un carré orange de dimensions $10 \times 10$ pixels.
   Ce type de codage vous semble-t-il efficace ? Imaginer une méthode de compression du codage.

## 2 Notion de format de fichier

### 2.1 Format de fichiers

Un programmes exécuté sur un ordinateur peut manipuler :

- des données **temporaires** stockées en mémoire vive, elles sont effacées à la fin du programme ;
- des données **persistantes** stockées dans une mémoire morte (ou de masse) comme un disque dur ou une clef USB, elles sont conservées après extinction de l'ordinateur.

Les données persistantes sont stockées sous formes de bits et organisées en sous-ensembles de la mémoire de masse qui les supporte. Ces sous-ensembles de données persistantes sont appelés fichiers. Les fichiers peuvent eux-mêmes être organisés en **répertoires** ....

**Définition 1**

Le format d'un fichier est le type de codage utilisé pour stocker les données contenues dans un fichier (texte, image, son ...). Il est identifié par l'extension qui suit le nom du fichier ( .txt, .pbm, .bmp, .pdf, .tex, .png ...).
On distingue deux grandes familles de fichiers :

- Les fichiers ASCII, ce sont des fichiers textes lisibles par un éditeur de texte (par exemple les formats d'image PBM P1 ou PGM P2, le format texte brut .txt, le format de données .csv ...)
- Les fichiers binaires qui contiennent des données codées directement en binaire (des 0 et des 1) dont on peut parcourir les octets avec éditeur hexadécimal. Ces fichiers doivent être lus par un logiciel compatible (par exemple les fichiers images .bmp, .png, les fichiers documents .odf ...)

**Propriété 1** *caractérisations juridiques d'un format de fichiers*

Un format de fichiers peut-être :

- ouvert : ses spécifications techniques sont publiques ce qui favorise l'interopérabilité des fichiers (utilisation par des plateformes et des logiciels différents) ;
- libre : format ouvert sous licence libre qui ne restreint pas son utilisation ou sa modification ;
- propriétaire : format élaboré par une entreprise dans un but commercial, peut être fermé (`.doc` de Microsoft) ou non (`.pdf` d'Adobe) ;
- normalisé : ayant fait l'objet d'une normalisation par une institution publique ou internationale (`ISO`, `W3C`), c'est le cas du format ouvert .odf (Open Document Format) et de ses dérivés (`.odt`, `.ods` ...)

Pour information :
`http://wiki.univ-paris5.fr/wiki/Format`

**Exercice 8**

Citer trois exemples de formats de fichiers et rechercher leurs caractérisations juridiques pour les types de données suivants :

- texte ;
- images ;
- sons.

## 2.2 Formats d'image

Les formats d'images sont composés d'un en-tête avec des données caractérisant l'image (type et taille du fichier, dimensions de l'image, type de codage, méthode de compression, palette ...) et des métadonnées (date, heure, créateur Îdots ). Puis viennent les données de l'image proprement dite.
Les données de l'image peuvent être compressées pour réduire la taille du fichier avec perte d'information (en se basant sur les limites de notre perception visuelle) ou sans perte (en codant la redondance des informations).
Pour information :
`wiki.univ-paris5.fr/wiki/Image_numérique`

**Exemple 1** *Exemple du format PBM*

le format PBM est un fichier d'image bitmap en noir et blanc, ouvert et non compressé. C'est un fichier texte encodé en ASCII qui se compose dans l'ordre (avec au plus 70 caractères par lignes) :

- des caractères P1 suivis d'un retour à la ligne ou d'un espace,
- la largeur de l'image, en base 10, suivie d'un retour à la ligne ou d'un espace,
- la hauteur de l'image, en base 10, suivie d'un retour à la ligne ou d'un espace,
- la liste des couleurs des pixels (0 pour blanc et 1 pour noir), ligne par ligne, de haut en bas et de gauche à droite -les retours à la ligne et les espaces étant ignorés dans cette partie.

**Exercice 9**

1. Rechercher sur le Web les caractéristiques juridiques des formats PGM P5, BMP, JPEG et PNG et préciser s'il s'agit de formats compressés.
2. Rechercher les caractéristiques du format GIF.
   a. Combien de bits occupe la représentation d'un pixel dans le format GIF ?
   Que représente la palette qui doit-être précisée dans l'en-tête du fichier ?
   b. Dans quels cas utilise-t-on des fichiers GIF ?
   S'agit-il d'un format libre ? Quels problèmes celà peut-il poser ? (Faire une recherche avec les mots clefs GIF et GNU).

**Exercice 10** *savoir choisir un format adapté*

1. Quels formats d'image peut-on utiliser pour stocker des images de bonne qualité sur une carte mémoire ? pour insérer une photo dans une page Web ?
2. Dans quels cas a-t-on intérêt à préférer le format GIF au format PNG ?

## 2.3 Droit des images, droit à l'image

**Exercice 11**

On effectuera des recherches internet (voir liens ci-dessous).

1. Peut-on insérer une photo qu'on a prise de la Joconde dans son site internet ? et une photo d'un camarade ?
2. Peut-on insérer une photo de la Joconde téléchargée sur un autre site dans son site internet ? et une photo d'un tableau de Dali ? et un logo ?
3. Un enseignant peut-il publier des photos des travaux de ses élèves sur son blog ou sur le site de l'établissement ?
4. Un enseignant peut utiliser des images téléchargées dans un support pédagogique diffusé aux élèves ou sur le site de l'établissement ?

Des liens utiles :

- `http://eduscol.education.fr/internet-responsable/`
- `http://creativecommons.fr/licences/les-6-licences/`
- `http://eduscol.education.fr/cdi/res/banques_dimages_lib`
- `http://www.debian.org/logos/index.fr.html`
- `http://fr.wikipedia.org/wiki/Wikipédia:Ressources_libres`
- `http://fr.wikipedia.org/wiki/Photographie_des_Oeuvres_d'art`

## 2.4 Manipulations d'images codées par des fichiers textes

**Exercice 12** *Inversion ou seuillage d'une image au format `pgm` (niveaux de gris)*

On se propose de modifier une image en niveaux de gris en inversant la valeur $x$ de chaque pixel, c'est-à-dire en appliquant à $x$ une fonction inversion qui retourne $255 - x$.
Le programme ci-dessous permet d'inverser l'image `flecheP5.pgm`. On accède aux fichiers en mode texte, mais ils sont encodés selon l'encodage 'cp1252' pour lequel un caractère est codé sur un octet. On peut donc travailler sur chaque caractère comme sur un octet du fichier.

1. Décrire en français l'algorithme implémenté par ce programme.
2. Faire une recherche internet sur la notion d'`endianness` (ou boutianité) en informatique pour préciser le sens de l'argument 'little' de la méthode `to_bytes` qui convertit un entier en `bytes`.
3. Compléter ce programme avec une fonction seuil qui prend en argument la valeur $x$ d'un pixel et retourne 0 si cette valeur est inférieure ou égale à 127 ou 255 sinon.
   Modifier le programme pour qu'il crée aussi un nouveau fichier `flecheP5seuil.pgm` obtenu par application de la fonction seuil à tous les pixels de l'image `flecheP5.pgm`.
4. Modifier le programme précédent pour qu'il inverse et seuille l'image `niveauxgrisP5.pgm`.

```python
#-*- coding: Utf-8 -*

import os,sys

def inversion (nbr): # On définit ici la fonction inversion
    return 255-nbr # On inverse les couleurs


#ouverture du fichier image initiale en mode lecture binaire
fichier_initial = open("flecheP5.pgm", 'rb')

#création d'un fichier image inversée en mode écriture binaire
fichier_modif1 = open("flecheP5inv.pgm", 'wb')

#affichage de l'endianness (ordre des bytes) du système
#sur un pc c'est 'little'
print('endianness du système :',sys.byteorder)

# on recopie l'entête qui ne sera pas modifié (espaces compris)
# ce sont les 54 premiers octets (à vérifier avec un éditeur hexadécimal)
debut_fichier = fichier_initial.read(54)
fichier_modif1.write(debut_fichier)

# On récupère la fin du fichier initial, le codage de l'image
malistoctet = fichier_initial.read()

#initialisation de la chaine d'octets modifiée
listoctetmodif1 = b''

for i in malistoctet: # On boucle sur les octets de la partie à coder
# on applique la fonction inversion à l'entier i (valeur de l'octet)
# puis on convertit l'entier obtenu en octet de type bytes
    listoctetmodif1 = listoctetmodif1 + inversion(i).to_bytes(1,'little')

#on écrit la liste modifiée à la suite du fichier modifié
fichier_modif1.write(listoctetmodif1)
fichier_initial.close() # Fermeture du fichier source
fichier_modif1.close() # Fermeture du fichier destination (essentiel)
os.system('pause') #commande système pause inconnue sous linux
```

inversionseuillage.py

**Exercice 13** *Changer les couleurs d'une image au format ppm (couleurs en représentation RGB)*

Le fichier `fleur_rouge.ppm` contient une image couleur au format `ppm` de type P3.

1. Faire une recherche documentaire sur ce format de fichier image.
2. Compléter le script ci-dessous qui crée un fichier `fleur_bleue.ppm` à partir de `fleur_rouge.ppm` en changeant le rouge en bleu.

```python
# -*- coding: utf-8 -*-
source = open('fleur_rouge.ppm','r')
but = open('fleur_bleue.ppm','w')

#on recopie l'en-tete du fichier source
for i in range(3):
    ligne = source.readline()
```

```
    but.write(ligne)
    #on récupère la hauteur et la largeur de l'image qui étaient sur la ligne 2
    if i==1:
L,H = [int(i)for i in ligne.rstrip().split()]
nbpixels = L*H
#on parcourt tous les pixels, codés par des triplets d'entiers
# entre 0 et 255 et séparés par des sauts de ligne
for i in range(nbpixels):
    .........
```

changer_couleur_ppm.py

# 3 Traitement d'images avec PIL

## 3.1 Présentation du module PIL

La bibliothèque (ou package) PIL (Python Image Library) offre une bibliothèque de fonctions permettant de manipuler une image.
On utilisera principalement le module Image qui s'importe avec la directive `from PIL import Image`.
Le parcours des pixels d'une image s'effectue ligne par ligne de haut en bas et de gauche à droite en partant du pixel du coin supérieur gauche de coordonnes (0,0).
En général on crée un objet de la classe Image avec `im = Image.open('chemin_fichier')`, puis on accède aux attributs de cet objet comme `im.format` ou on le manipule avec ses méthodes comme `im.split()`. Attention, la fonction open ne fait que lire l'en-tête du fichier image pour lire ses attributs (taille, format ...) mais elle ne charge pas en mémoire les valeurs de tous les pixels. Ce chargement est effectué si nécessaire lors d'une méthode qui modifie des pixels par exemple.
On peut forcer le chargement en mémoire d'un objet image créé par PIL avec `pixels = im.load()` , les pixels de coordonnées $(x;y)$ sont alors accessibles avec la syntaxe `pixels[x,y]`.

**Principales fonctionnalités du module Image de PIL**

| | |
|---|---|
| `nom_image = Image.open("chemin_fichier")` | ouverture d'un fichier image |
| `nom_image.load()` | force le chargement de l'image en mémoire) |
| `nom_image.mode` | mode de l'image : 'L' en niveaux de gris, 'RGB' en couleurs |
| `nom_image.format` | format de l'image |
| `nom_image.size` | taille de l'image sous la forme (Largeur,Hauteur) |
| `nom_image.save("chemin_fichier")` | sauvegarde d'une image dans un fichier |
| `nom_image = Image.new('RGB',(L,H))` | création d'une image 'RGB' de dimensions (Largeur,Hauteur) |
| `r,g,b = nom_image.split()` | récupère les composantes (r,g,b) de l'image |
| `nom_pixel = nom_image.getpixel((x,y))` | lecture du pixel de coordonnées (x,y) |
| `nom_image.putpixel((x,y),(r,g,b))` | écriture de la valeur (r,g,b) dans le pixel (x,y) |

Ouvrir le programme `tutorielPIL.py` (extraits ci-dessous) qui donne des exemples de fonctions manipulant des images en niveaux de gris (mode 'L') ou en couleur (mode 'RGB').
Tester ce programme avec les fichiers images 'niveauxgrisP5.pgm' (image $3\times3$ en niveaux de gris de profondeur 8 bits), 'rgbP6.ppm' (image RGB couleur $3\times2$ de profondeur 32 bits) et 'joconde.bmp' (image RGB en couleur).

```python
#-*- coding: Utf-8 -*
from PIL import Image

def parcours_pixels(image):
    """Parcourt et affiche la valeur des pixels
    d'une image PIL, l'image est parcourue ligne par ligne
    de haut en bas et de gauche à droite en partant du coin supérieur gauche"""
    #Dimensions de l'image sous la forme (Largeur,Hauteur)
    L,H = image.size
    # on balaie toutes les lignes de l'image source, de 0 à H-1
    for y in range(H):
        # on balaie toutes les lignes de l'image source, de 0 à L-1
        for x in range(L):
          #la méthode getpixel permet d'accéder à la valeur d'un pixel
          valeur = image.getpixel((x,y))
          print valeur


def monochromeGray(image,valeur):
    """Parcourt les pixels d'une image en niveaux de gris de profondeur 8 bits
    et affecte à chacun l'argument valeur (un nombre) """
    L,H = image.size
    # on balaie toutes les lignes de l'image source, de 0 à H-1
    for y in range(H):
        # on balaie toutes les lignes de l'image source, de 0 à L-1
        for x in range(L):
            #la méthode putpixel permet de modifier la valeur d'un pixel
            image.putpixel((x,y),valeur)


def monochromeRGB(L,H,valeur):
    """Retourne une nouvelle image obtenue à partir d'une image
    RGB de profondeur 32 bits en affectant à chacun l'argument valeur (un triplet (r,g,b))
        """
    #création d'une nouvelle image de dimensions (L,H)=(Largeur,Hauteur)
    im = Image.new('RGB',(L,H))
    # on balaie toutes les lignes de l'image source, de 0 à H-1
    for y in range(H):
      # on balaie toutes les lignes de l'image source, de 0 à L-1
      for x in range(L):
          im.putpixel((x,y),valeur)
    return im
```

tutorielPIL.py

```python
def monochromeRGB2(L,H,valeur):
    """Parcourt les pixels d'une image RGB de profondeur 32 bits
    et affecte à chacun l'argument valeur (un triplet (r,g,b))"""
    im = Image.new('RGB',(L,H))
    #on charge les pixels de l'image avec la méthode load
    pixels = im.load()
    # on balaie toutes les lignes de l'image source, de 0 à H-1
    for y in range(H):
        # on balaie toutes les lignes de l'image source, de 0 à L-1
        for x in range(L):
            #accès en lecture / écriture avec opérateur crochets
            pixels[x,y] = valeur
    return im


#programme principal
```

```
17
18 #On manipule d'abord une image en niveaux de gris
19 #En Python2, raw_input a la meme fonctionnalité qu'input en Python3
20 #et la fonction print s'utilise sans ()
21 chemin = raw_input("Entrez le chemin de l'image en niveaux de gris : \n")
22 im1 = Image.open(chemin)
23 L, H = im1.size
24 print "Parcours des pixels de l'image :"
25 parcours_pixels(im1)
26 valeur = int(raw_input("Entrez une valeur de gris "))
27 #création d'une nouvelle image de mêmes dimensions que im1
28 im2 = Image.new('L',(L,H))
29 #Transformation de im2 en image monochrome
30 monochromeGray(im2,valeur)
31 #sauvegarde de l'image modifiée (plusieurs format possibles)
32 im2.save('imagegraymodif.png')
33 #affichage de l'image modifiée
34 im2.show()
```

**Exercice 14**

1. Dans le code ci-dessous, la fonction `permute_bands(im)` prend en entrée une image RGB et crée une nouvelle image par permutation circulaire de ses composantes (RGB donne BRG). Elle utilise la fonction `merge` du module `Image` de `PIL` .

   Tester cette fonction avec l'image `'joconde.bmp'` puis créer une fonction `isole_bande(im,bande)` qui retourne une nouvelle image avec une seule composante, celle indiquée par `bande`.

2. La documentation de `PIL` se trouve à l'adresse `www.pythonware.com/products/PIL`.

   Dans la documentation du module `Image`, rechercher le rôle et la syntaxe des méthodes `crop` et `paste` d'un objet de la classe Image.

3. Dans le code ci-dessous, la fonction `rouler_horizontal(im,delta)` retourne une image obtenue par un roulé d'une image initale, de delta pixels dans le sens de sa largeur.

   Ecrire une fonction similaire `rouler_vertical(im,delta)`.

4. Ecrire une fonction `moustache(im,color)` qui dessine une moustache de couleur à la Joconde (ou à toute autre figure) contenue dans l'image `im`.

```
1  from PIL import Image
2
3  def rouler_horizontal(im,delta):
4      """retourne une image obtenue  par un roulé horizontal de im de delta  pixels"""
5      w,h = im.size
6      if delta == 0:
7          return im
8      else:
9          delta = delta%w
10         #découpage de la partie 1
11         part1 = im.crop((0,0,delta,h))
12         #on charge part1 en mémoire
13         part1.load()
14         #découpage de la partie 2
15         part2 = im.crop((delta,0,w,h))
16         #on charge part2 en mémoire
17         part2.load()
18         #on colle part2 dans im
19         im.paste(part2,(0,0,w-delta,h))
20         #on colle part1 à la suite
21         im.paste(part1,(w-delta,0,w,h))
```

```
22
23 def rouler_vertical(im,delta):
24     .....
25 def permute_bands(im):    """permutation circulaire des composantes RGB"""
26     r,g,b = im.split()
27     return Image.merge("RGB",(b,r,g))
28
29 def moustache(im,color):
30     """dessine une moustache de couleur au format (R,G,B)"""
31     .......
32
33
34 im = Image.open('joconde.bmp')
35 #copie de l'image
36 copie = im.copy()
37 rouler_vertical(copie,100)
38 copie.show()
39 copie = im.copy()
40 rouler_horizontal(copie,100)
41 copie.show()
42 im2 = permute_bands(im)
43 im2.show()
44 moustache(im,(240,240,40))
45 im.show()
```

## 3.2 Création de nuanciers Rouge, Vert, Bleu ou Gris

**Exercice 15**

1. Ecrire une fonction `nuancier_rouge` qui crée et affiche une image RGB de dimensions $256 \times 256$ représentant un nuancier des différentes nuances de rouge : du plus foncé à gauche au plus clair à droite.
2. Modifier cette fonction pour qu'elle affiche les nuances de rouge du plus clair à gauche au plus foncé à droite puis du plus clair en haut au plus foncé en bas.
3. Ecrire des fonctions `nuancier_couleur` qui réalisent un nuancier de couleur pour couleur=vert ou couleur=gris.

## 3.3 Inversion,seuillage, extraction du contour d'une image en niveaux de gris

**Exercice 16**

Le programme ci-dessous a pour but d'appliquer trois fonctions à une image en niveaux de gris (par exemple 'LenaGray.bmp' :

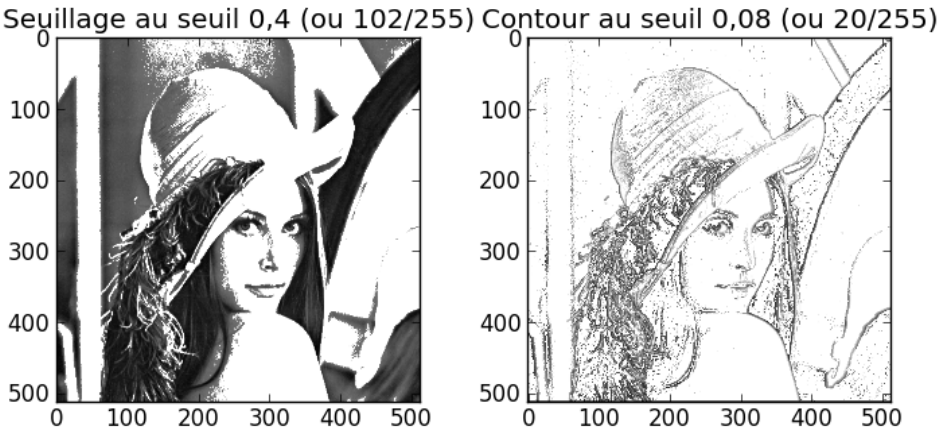
- `enregistrer` qui enregistre l'image à l'emplacement indiqué par le chemin puis affiche l'image ;
- `filtre_inversion` qui prend en argument une image en niveaux de gris et affiche l'image inversée en l'enregistrant à l'emplacement indiqué par le chemin ;
- `filtre_seuil` qui prend en argument une image en niveaux de gris,la transforme l'image noir et blanc selon un seuil et l'enregistre à l'emplacement indiqué par le chemin ;
- `filtre_contour` qui extrait le contour d'une image en noircissant les pixels dont la différence de valeurs avec leurs voisins excède un seuil.

1. Ouvrir le fichier, compléter le corps des fonctions `enregistrer`, `filtre_seuil` et `filtre_contour`, puis tester le programme.

**2.** Ecrire en Français l'algorithme d'extraction de contour implémenté par la fonction `filtre_contour`.

```
#-*- coding: Utf-8 -*
import osfrom PIL import Image

def enregistrer(image,chemin):
     .......

def filtre_inversion(image1,chemin):
     .....

def filtre_seuil(image1,seuil,chemin):
   ....

def filtre_contour(image1,seuil,chemin):
    L,H = image1.size
    image2 = Image.new('L',(L,H))
    pixels = image1.load()
    pixels2 = image2.load()
    for y in range(H-1):
        for x in range(L-1):
           if abs(pixels[x,y]-pixels[x,y+1])>seuil or abs(pixels[x,y]-pixels[x+1,y])>seuil:
               pixels2[x,y] = 0
           else:
               pixels2[x,y] = 255
    enregistrer(image2,chemin)

#Programme principal
    chemin = 'LenaGray.bmp'
    image = Image.open(chemin)
    ....
```

traitementimageGray.py



## 3.4 Transformations diverses d'images RGB

**Exercice 17**

1. Ouvrir l'image `LenaRGB.png` avec `Gimp` et prélever la couleur de quelques pixels avec l'outils `Boîte à outils > Pipette`. Les trois canaux (R,G,B) ont-ils la même valeur ? Dans le menu `Image` cliquer sur `Mode > Niveaux de Gris` puis prélever les valeurs de plusieurs pixels avec l'outil pipette. Que remarque-t-on ?
2. Programmer en Python le corps d'une fonction `niveauxgris` dont on donne l'en-tête ci-dessous. Tester cette fonction sur l'image `LenaRGB.png`.
3. Programmer en Python le corps d'une fonction `ajout_images` dont on donne l'en-tête ci-dessous et qui retourne une image dont la composante R est celle de l'image 1 et les composantes G et B celles de l'image 2. Tester cette fonction avec des images RGB mêmes dimensions et de même profondeur telles que `fleurs.png` et `papillon.png`.
4. Programmer en Python le corps d'une fonction `ajout_masque` dont on donne l'en-tête ci-dessous et qui retourne une image RGB obtenue par application d'une image masque à une image source. L'image masque est en général une forme géométrique. L'algorithme consiste à créer une image destination de couleur blanche, de même dimensions que l'image source et le masque. On parcourt tous les pixels du masque à l'aide d'une double boucle : si le pixel (x,y) n'est pas blanc, alors on copie le pixel (x,y) de l'image source en (x,y) sur l'image destination.
   Tester cette fonction avec les images `masquecirculaire.png` et `LenaRGB.png`.

```
#-*- coding: Utf-8 -*
from PIL import Image

def niveauxgris(image1,chemin):
    """retourne une image RGB en niveaux de gris
    obtenue à partir d'une image couleur RGB et l'enregistre
    à l'emplacement indiqué par le chemin"""
    ....

def ajout_images(image1,image2):
    """retourne une image 3 RGB composée de
    la composante R de l'image 1 et des composantes G et B de l'image2"""
    ....

def ajout_masque(image1,masque):
    """retourne une image RGB obtenue par application
    d'une image masque à une image RGB"""
    ....
```

traitementRGBeleve2.py

## 3.5 Transformations géométriques d'une image

**Exercice 18**

Le programme `rotationRGBeleve.py` a pour but d'appliquer diverses rotations simples à une image.
On crée un objet `PIL` en ouvrant un fichier image de dimensions $(L\,;\,H)$ avec `Image.open('nom_fichier')`. Les coordonnées des pixels situés aux coins de l'image sont :

- $(0\,;\,0)$ = pour le coin supérieur gauche et $(L-1\,;\,0)$ = pour le coin supérieur droit ;
- $(0\,;\,H-1)$ = pour le coin inférieur gauche et $(L-1\,;\,H-1)$ = pour le coin inférieur droit.

On peut appliquer une transformation géométrique à cette image, s'il s'agit d'une bijection. Si on note `pixels1` sa matrice de pixels on procède ainsi :

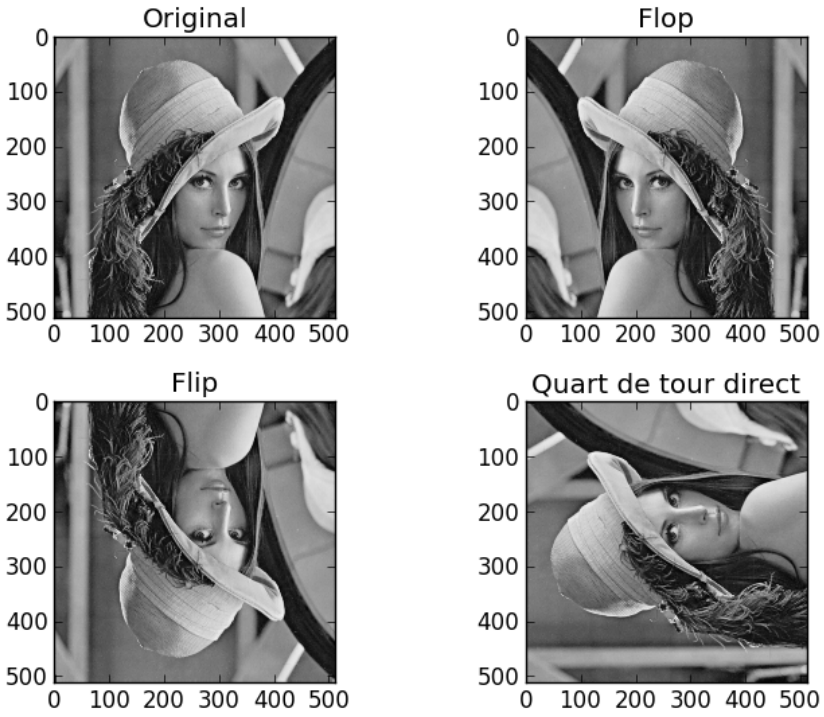
- on crée une image de mêmes dimensions et on note `pixels2` sa matrice de pixels

- on parcourt M2 et on affecte à `pixels2[x,y]` la valeur du pixel de `pixels1` antécédent de `pixels2[x,y]` par la transformation.

1. Expliquer le rôle des deux boucles imbriquées de la fonction `flip`. Tester cette fonction, quelle transformation réalise-t-elle sur l'image ?
2. Programmer une fonction `flop` qui réalise une rélexion de l'image par rapport au bord droit de l'image.
3. Programmer une fonction `quart_tour_direct` qui réalise un quart de tour direct de l'image de centre le coin supérieur gauche de l'image.
4. Programmer des fonctions `quart_tour_indirect` et `demi_tour` qui réalisent respectivement un quart de tour indirect et un demi tour, de l'image de centre le coin supérieur gauche de l'image.

```python
#-*- coding: Utf-8 -*
import os
from PIL import Image

def flip(image1):
    L,H = image1.size
    image2 = Image.new('RGB',(L,H))
    pixels1 = image1.load()
    pixels2 = image2.load()
    for y in range(H):
        for x in range(L):
            pixels2[x,y] = pixels1[x,H-1-y]
    return image2

....
#test des fonctions
im4 = Image.open('LenaRGB.png'); im5 = flip(im4); im5.save('imageflip.png')
```

rotationRGB.py



# 4 Traitement d'images avec `matplotlib` et `numpy`

## 4.1 Histogramme d'une image avec `numpy` et `matplotlib`

On veut représenter l'histogramme de la distribution des pixels d'une image bitmap.

Pour récupérer la série des valeurs des pixels (énumérées de haut en bas et de gauche à droite) on peut ouvrir l'image avec `PIL` :

- puis utiliser la méthode `getadata()` de l'objet image
- ou importer le module `numpy` sous l'alias `np` et récupérer la matrice de l'objet image avec la fonction `np.array()`. Les `array` de `numpy` sont des tableaux multidimensionnels comme les listes de Python mais qui ne peuvent contenir que des objets du même type (int,float,bool,complex) contrairement aux listes. Enfin on applatit cette matrice $2\times2$ avec la méthode `flatten()`.

```
>>> import os
>>> os.chdir('\MesDocuments\ProgrammesPython') #on règle le répertoire courant de la console
      Python
>>> from PIL import Image
>>> img = Image.open('lenagray.png')
>>> data = img.getdata()
>>> data[0] #pixel du coin supérieur gauche
162
>>> len(data)
262144
>>> 512**2
262144
>>> import numpy as np
>>> matrice = np.array(img) #un tableau de dimension 512x512
>>> matrice.shape
(512, 512)
>>> matrice[0][0] #pixel du coin supérieur gauche
162
>>> data2 = matrice.flatten() #conversion de matrice en un tableau à 1 dimension
>>> data2.shape
(262144,)
>>> data2[0]
162
```

Numpy et matplotlib avec la console Python

Pour représenter graphiquement l'histogramme, on va importer le module `matplotlib.pyplot` avec l'alias `plt`, appliquer la fonction `plt.hist()` à nos données, afficher l'histogramme avec plt.show() puis sauvegarder la figure avec `plt.savefig('nom.png')`.

```
>>> import matplotlib.pyplot as plt
>>> plt.hist(data, bins=256,range=(0,255),facecolor='green') #on peut remplacer data par
      data2
>>> plt.show()
>>> plt.savefig('histo.png')
```

Quelques liens vers la documentation des modules `numpy` et `matplotlib` :

- liste des fonctions de `matplotlib.pyplot` : `http://matplotlib.org/api/pyplot_summary.html`
- pour `matplotlib.pyplot`, un tutoriel, `http://matplotlib.org/users/pyplot_tutorial.html`
- un autre, `http://www.loria.fr/~rougier/teaching/matplotlib/`
- tutoriel de `numpy`, `http://wiki.scipy.org/Tentative_NumPy_Tutorial`

`matplotlib.pyplot` est un sous-module du module `matplotlib` dédié aux représentations graphiques de données. `matplotlib.pyplot` fournit une liste de fonctions permettant de réaliser des figures à partir d'éléments graphiques comme des courbes, des histogrammes, des images ...

**Manipulation d'images avec `matplotlib.pyplot`**

| | |
|---|---|
| `tableau = matplotlib.pyplot.imread('nom.ext')` | `ouvre un fichier image et le stocke dans un array numpy` |
| `matplotlib.pyplot.imsave(fname='nom.ext',arr=tableau)` | `enregistre un tableau sous la forme d'un fichier image)` |
| `matplotlib.pyplot.imshow(tableau,cmap='gray')` | `insère l'image correspondant à un tableau dans une figure matplotlib avec l'option palette niveaux de gris` |
| `matplotlib.pyplot.show()` | `affiche à l'écran la figure matplotlib` |

En général on importe ce module sous un alias comme `plt` avec `import matplotlib.pyplot as plt`, dans ce cas on obtient la matrice d'une image avec `matrice = plt.imread('lenagray.png')`.

Attention si l'image est de profondeur 8 bits, les valeurs des pixels ne sont pas des entiers entre 0 (noir) et 255 (blanc) mais des flottants entre 0 et 1. En effet, si on veut manipuler les valeurs de ces pixels avec des fonctions mathématiques, il est plus simple de considérer des fonctions définies sur $[0;1]$. Mais il faut garder à l'esprit qu'il n'y a que 256 valeurs de niveaux de gris possibles dans cet intervalle.
Si $g$ est le niveau de gris dans $[0;1]$, on passe à sa représentation entière sur un octet avec $\text{Ent}(g \times 255)$ et dans l'autre sens il suffit de diviser par 255.

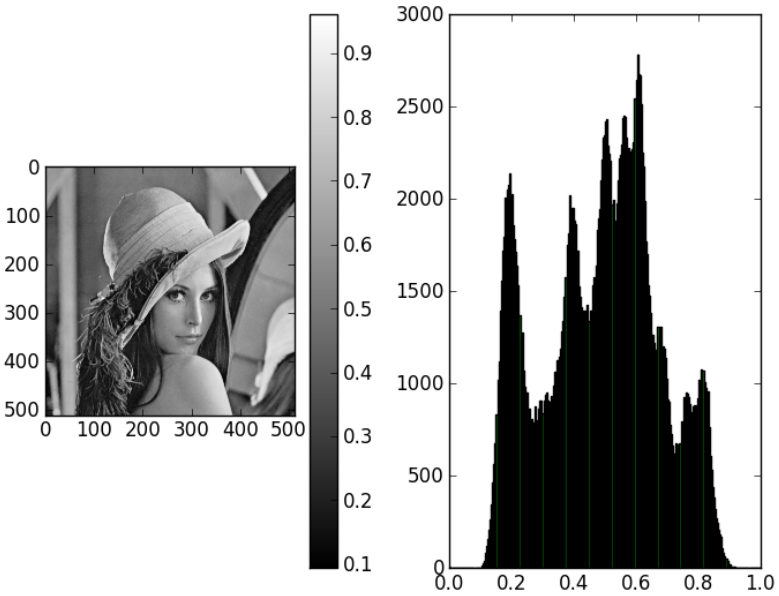
Dans `numpy`,les opérateurs et fonctions mathématiques classiques sont vectorialisés et peuvent être appliqués directement à un `array` . On peut facilement ramener toute une matrice de pixels compris entre 0 et 255 dans l'espace de valeurs $[0;1]$.

```
>>> a = np.array([[255,200,50],[0,100,129]])
>>> a.dtype
dtype('int32')
>>> b = a/255
>>> b
array([[ 1.        ,  0.78431373, 0.19607843],
       [ 0.        ,  0.39215686, 0.50588235]])
>>> b.dtype
dtype('float64')
```

**Exercice 19**

1. Tester le script ci-dessous qui permet d'afficher une image et son histogramme :

```
import matplotlib.pyplot as plt

matrice2 = plt.imread('lenagray.png')
plt.subplot(121) #première sous figure
plt.imshow(matrice2,cmap='gray') #ajout de l'image à la figure avec la palette gray
plt.colorbar() #ajout d'une barre des couleurs de la palette
plt.subplot(122) #seconde sous figure
plt.hist(matrice2.flatten(), bins=256,range=(0,1),facecolor='green') #histogramme
plt.subplots_adjust(wspace=0.3) #ajustement de l'espace horizontal entre les subplots
plt.savefig('lena+histo.png') #enregistrement sur disque
plt.show() #affichage à l'écran
```




2. Modifier la valeur de l'option colormap dans `imshow()`. Toutes les palettes (colormap) sont référencées sur la page :
   `http://matplotlib.org/examples/color/colormaps_reference.html`
3. Rechercher dans la documentation de `matplotlib.pyplot` la syntaxe de la fonction `subplot()` qui permet de créer une sous-figure.

## 4.2 Repérage d'un pixel dans une matrice d'image

Avec `PIL` on récupérait la matrice des pixels d'une image à travers un objet par `pixels = im.load()` puis `pixels[x,y]` pour le pixel en $(x;y)$. Comme l'origine du repère est au coin supérieur gauche, $x$ est l'index de la colonne et $y$ celui de la ligne.
En combinant `numpy` et `matplotlib`, on peut récupérer la matrice d'une image bitmap de dimensions $W \times H$ (nombre de colonne fois nombre de lignes) sous la forme d'un `array` `numpy` de $H$ lignes et $W$ colonnes.
**Attention, pour accéder au pixel de coordonnées $(x;y)$ on écrira `matrice[y][x]`.**

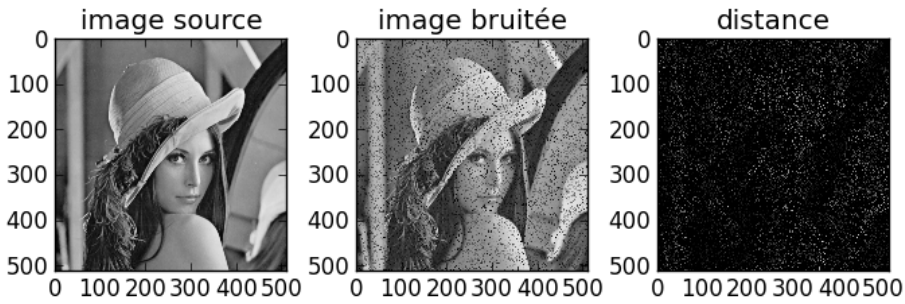
## 4.3 Opérations algébriques sur une image

**Exercice 20**

Dans le script ci-dessous, la fonction `bruitage(matrice)` prend une matrice d'image et retourne une matrice d'image bruitée où certains pixels sont remplacés par des pixels blnacs.
Quelques fonctions de `numpy` qui nous seront utiles :

```
>>> m = np.zeros((2,3)) #pour créer un tableau de 2 lignes et 3 colonnes rempli de zéros
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
>>> m+1   #un tableau rempli de 1
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> np.random.randint(0,512,5) #un tableau de 5 entiers aléatoires choisis entre 0 et 512
array([ 41, 438, 349, 98, 7])
```

La fonction `distance(matrice1,matrice2)` retourne la matrice obtenue par différence des matrices de deux images de mêmes dimensions, ou un message d'erreur si les dimensions sont différentes.




1. Commenter le code de la fonction `bruitage(matrice)`.
2. Compléter les codes de la fonction `distance(matrice1,matrice2)`.

```
import matplotlib.pyplot as plt
import numpy as np

def bruitage(matrice):
    H,W = matrice.shape
    matrice2 = matrice.copy()
    for y in range(H):
        bruit = np.random.randint(0,W-1,10)
        for x in bruit:
            matrice2[y][x] = 0
    return matrice2

def distance(matrice1,matrice2):
    H,W = matrice1.shape
    if matrice1.shape != matrice2.shape:
        return "Les images ne sont pas de même dimension"
    else:
        matrice3 = np.zeros((H,W))
        ........


matrice1 = plt.imread('lenagray.png')
matrice2 = bruitage(matrice1)
matrice3 = distance(matrice1,matrice2)
.......
```

## 4.4 Traitement d'image avec un filtre

### 4.4.1 Principe

On a parfois besoin d'améliorer certaines caractéristiques d'une image ou d'en extraire de l'information. On lui applique alors une fonction mathématique dite de filtre. Si on note `M1` la matrice de l'image source, on procède ainsi :

- on crée une matrice `M2` de mêmes dimensions remplie de zéros avec par exemple la fonction `zeros` de `numpy`.
- on parcourt `M2` et on affecte à `M2[y][x]` l'image de la valeur du pixel de `M1` par la fonction de filtre.

Pour la modélisation, il est plus simple de définir la fonction de filtre sur $[0;1]$ et de manipuler des valeurs de pixels dans cet intervalle (ce que renvoie la fonction `imread` de `matplotlib.pyplot`).

### 4.4.2 Correction $\gamma$ de la luminance

**Exercice 21**

1. Expliquer pourquoi on peut éclaircir une image, en lui appliquant une fonction filtre $f$ définie sur $[0;1]$ telle que :
   - $f([0;1]) = [0;1]$ et $f(0) = 0$ et $f(1) = 1$
   - pour tout $x \in [0;1]$, on a $f(x) \leqslant x$

   En déduire les caractéristiques d'une fonction filtre qui assombrisse l'image.

   Parmi les fonctions usuelles lesquelles pourrait-on utiliser pour éclaircir ou assombrir une image ?

2. Pour corriger la luminance, on trouve souvent dans les logiciels spécialisés un outil de correction Gamma. Celui-ci applique à l'image une fonction filtre définie sur $[0;1]$ par $f(x) = x^{\frac{1}{\gamma}}$ avec $\gamma > 0$. Si on compose $f$ avec $g(x) = x^{\gamma}$ alors $g \circ f(x) = x$ et on retrouve l'image initiale.

   Si $x$ est un entier entre 0 et 255 on le remplace par $\text{Ent}\left(255 \times \left(\frac{x}{255}\right)^{1/\gamma}\right)$.

   Quel est l'effet obtenu si $0 < \gamma < 1$ ? et si $\gamma > 1$ ?

3. Ouvrir le fichier `lenagray.png` avec `Gimp`.

   Dans le menu `Couleurs > Courbes`, faire apparaître l'histogramme de l'image et un segment initialement d'équation $y = x$. En abscisse figurent les niveaux de gris initiaux, et en ordonnée les niveaux de gris résultants

   **a.** Saisir un un point au milieu du segment et tirer vers le bas. Comment varie la luminance (intensité lumineuse) de l'image ? Et si on tire vers le haut ?

   **b.** Ouvrir le menu `Couleurs > Niveaux`, le triangle au milieu de la barre de couleur contrôle la valeur de $gamma$. Faire varier ce curseur pour éclaircir ou assombrir l'image.

4. Ecrire un script Python qui récupère la matrice d'une image png en niveaux de gris avec la fonction `imread` de `matplotlib` puis qui applique une fonction `filtre(matrice,fonction)` retournant la matrice de l'image filtrée avec la fonction passée en paramètre. C

5. Sur l'image `lenagray.png`, tester ainsi plsuieurs filtres de correction $\gamma$ d'éclaircissement ou d'assombrissement.

   Si on compose un filtre $\gamma$ avec un filtre $\frac{1}{\gamma}$, retrouve-t-on une image identique à l'image source ? On peut afficher l'image différence avec la fonction `distance` définie dans l'exercice .

### 4.4.3 Accentuation de contraste

**Exercice 22**

1. Expliquer pourquoi on peut accentuer le contraste d'une image, en lui appliquant une fonction filtre $f$ définie sur $[0;1]$ telle que :
   - $f([0;1]) = [0;1]$ et $f(0) = 0$, $f(1) = 1$ et $f(0,5) = 0,5$.
   - pour tout $x \in ]0;0,5[$, on a $f(x) < x$ et pour tout $x \in ]0,5;1[$, on a $f(x) > x$

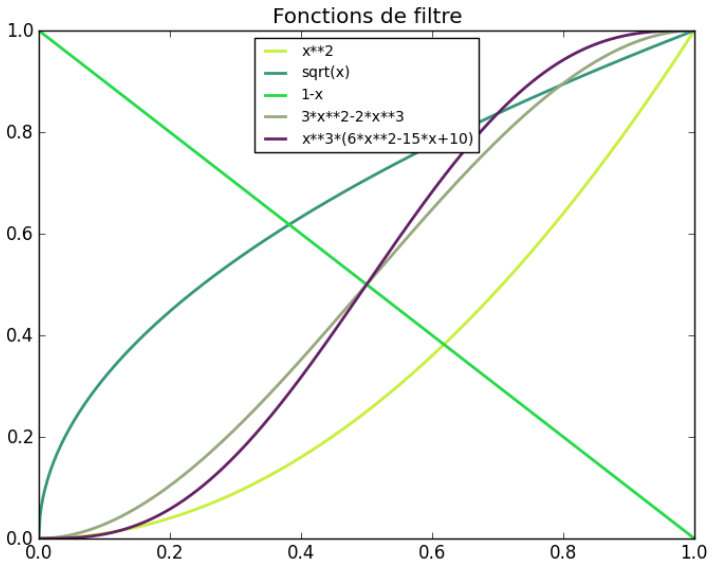
2. On peut ajouter la contrainte que $f$ dérivable sur $[0;1]$ et que $f'(0) = f'(1) = 0$.

   Commenter l'extrait ci-dessous d'une feuille de calcul formel réalisée avec Maxima :

```
(%i1)    f(x):=a*x^3+b*x^2+c*x+d;
(%o1)  f(x) := a x^3 + b x^2 + c x + d

(%i2)    linsolve([f(0)=0,f(1)=1,f(1/2)=1/2,3*a^2+2*b*x+c=0],[a,b,c,d]);
(%o2)  [a = -2, b = 3, c = 0, d = 0]
```

   En déduire une fonction filtre pour accentuer le contraste. En modifiant la condition sur la dérivée en 1, déterminer d'autres fonctions filtres de contraste avec Maxima.

3. Appliquer un filtre de contraste à l'image `lenagray.png` avec la fonction `filtre(matrice,fonction)` définie dans l'exercice 20.




# 5 Représentation des sons

## 5.1 Numérisation d'un son

On peut définir le son comme la variation de la pression de l'air au cours du temps.
C'est une grandeur physique **analogique** car elle varie continument dans le temps. La plupart des grandeurs physiques (son, intensité lumineuse, température ...) sont analogiques. En électronique numérique (dans un ordinateur, un téléphone mobile, un lecteur CD ...) , l'information est modélisée sous la forme d'un nombre d'états finis et discontinus, matérialisés par une succession de tensions positives (état1) ou négatives (état 0). Ces 0 et ces 1 sont les bits permettant de représenter n'importe qu'elle grandeur numérique en binaire.
Pour numériser un signal analogique comme le son (ou la lumière) on procède en trois étapes :

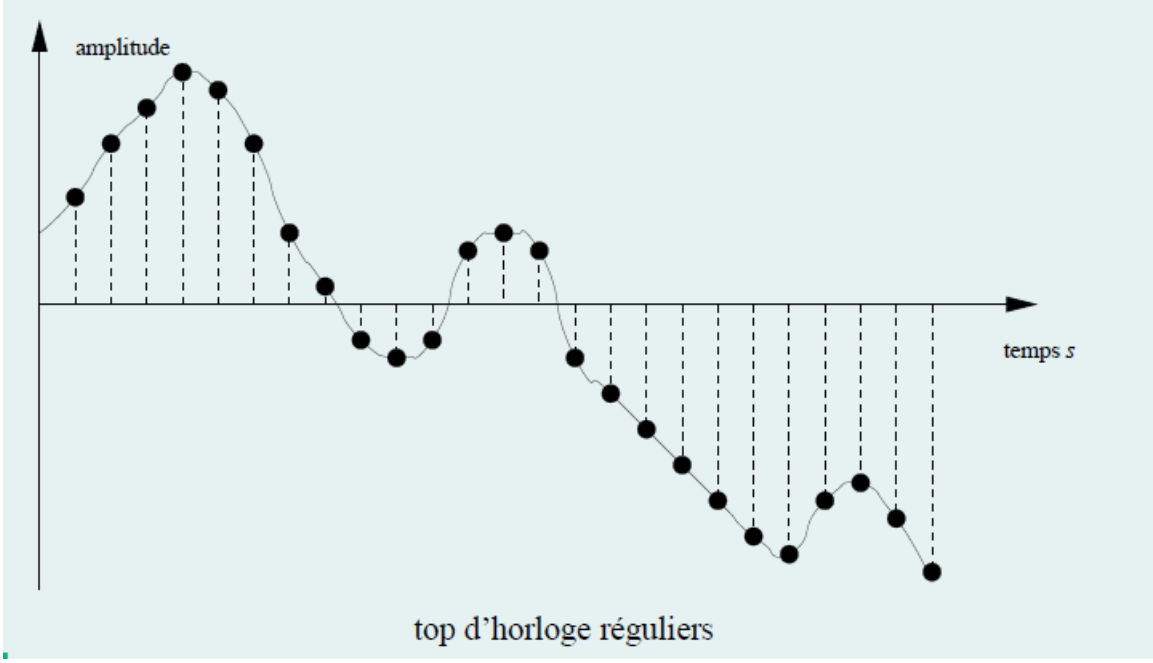
1. on **échantillonne** le signal en mesurant avec un capteur un nombre fini de valeurs à intervalles réguliers. Pour le son c'est un échantillonnage temporel et pour une image un échantillonnage spatial ;
2. on **quantifie** ensuite chaque échantillon sur une échelle comportant un nombre limité de valeurs discrètes (par exemple de 0 à 255 pour le niveau de gris d'un pixel pour une image). ;
3. enfin on **code** chacune de ces valeurs en binaire .

Evidemment plus la fréquence d'échantillonnage est grande , plus la représentation numérique d'un son sera fidèle au son réel analogique, mais le poids en bits de cette représentation sera d'autant plus grand.

**Théorème 1** *Nyquist-Shannon*

La fréquence d'échantillonnage d'un son analogique doit être supérieure ou égale à deux fois la fréquence maximale contenue dans le son.

Le son sinusoïdal le plus aigu que notre oreille perçoit est de 22 000 Hz environ, c'est pourquoi on échantillonne en général à 44 000 Hz.




**Exercice 23**

1. Faire une recherche internet sur Claude Shannon et Harry Nyquist.
2. Ouvrir le logiciel libre Audacity (sous licence GNU/GPL), choisir le menu `Générer > Son`, vérifier que la fréquence d'échantillonnage est réglée à 44 100 Hz et générer des sons avec les fréquences suivantes :
   - 440 Hz
   - 22 000 Hz
   - 22 050 Hz
   - 30 000 Hz

   Utiliser les outils de zoom temporel et d'affichage du spectre des fréquences. Que remarque-t-on ?
3. On peut étudier le problème réciproque, en ouvrant un nouveau fichier, en observant l'effet de la variation de la fréquence d'échantillonnage d'un son de 440 Hz. Avant de générer le son, on règlera la fréquence d'échantillonnage dans le menu `Edition > Préférences`

## 5.2 Précisions sur les formats de fichier audio

On pourra consulter la page `http://wiki.univ-paris5.fr/wiki/Format_audio`.
Un format de fichier audio possède des caractéristiques techniques :

- liées au type de numérisation de la source : fréquence d'échantillonnage, profondeur en octets du codage et nombre de canaux enregistrés (son mono = 1, stéréo = 2, multipiste = 3 et plus) qui permettent une restitution sur plusieurs enceintes.

  **Poids du fichier (kilooctets/seconde) = Fréquence × Codage × Nb de pistes**

- la compression (**codec** pour COmpression DEcompression)) utilisée : les fichiers audio sont lourds à cause des fréquences d'échantillonnage élevées, ils nécessitent d'être compressés : avec perte (codec `MP3, WMA, AAC, VORBIS`) ou sans perte (codec PCM).

Les formats de fichiers audio peuvent être des codecs (mp3, aac) ou des **fichiers conteneurs**, sortes de valises qui permettent d'archiver plusieurs flux audio ou video encodés avec des codecs et des meta-données (titre, auteur ...) : par exemple les formats `waw` ou `ogg` pour l'audio ou `MP4` pour la vidéo.
Par ailleurs, les formats audio peuvent être libres, propriétaires, ouverts ...

**Exercice 24**

Faire une recherche internet pour déterminer les caractéristiques des formats audio suivants :

| | | | |
|---|---|---|---|
| • WAV | • OGG VORBIS | • AAC | • AIF |

## 5.3 Tailles de fichiers, savoir choisir un format adapté

**Exercice 25**

Revoir les unités de taille de fichiers informatiques à la page `http://fr.wikipedia.org/wiki/Octet`.
Une commande de la console Linux pour afficher la taille d'un fichier en octets est `du nom_fichier` avec éventuellement l'option `-h`, `du -h nom_fichier` pour un affichage en multiples binaires de l'octets.
Une commande pour afficher la liste des fichiers d'un répertoire classés par taille est `ls -S`.

1. On enregistre un son pendant 10 mn avec 10 000 échantillons par seconde et 16 bits pour chaque échantillon, sans compresser les données. Quelle est la taille du fichier en megabits ? megaoctets (Mo) ? en mébioctets (Mio) ?
2. Calculer le poids d'1 minute audio en 44 100 Hz, 16 bits, stéréo. Donner le résultat en mébioctets.
3. On enregistre une image 10 cm × 10 cm, avec 100 pixels par centimètre, chaque pixel étant représenté par trois nombres entiers, chacun codé sur un octet. Quelle est la taille du fichier ?
4. Mon système d'exploitation affiche une capacité de disque dur de 304 933 236 736 octets arrondis à 283 Go. Comment expliquer cette incohérence ?
5. Pour une photographie de vacances, peut-on accepter un format compressé avec perte ? Si on veut envoyer cette photo par un courrier électronique, quelle taille maximale de fichier doit-on prévoir ? Comment s'assurer que les récepteurs pourront lire cette image ?



# Table des matières