

1 Ouverture d'un fichier image de format bitmap

1.1 Fichiers images en informatique

L'ordinateur peut traiter un nombre infini d'informations. Pour représenter une image dans un fichier informatique, il faut donc la **numériser** (ou discrétiser) en sélectionnant un certain nombre de caractéristiques.

On peut distinguer deux grandes types de représentation informatique d'une image :

- la **représentation vectorielle** pour les images qu'on peut représenter à partir d'un nombre fini de figures géométriques élémentaires (ellipses, rectangles, courbes de Bézier ...). Le fichier image contient une suite d'instructions de tracé exprimées dans un langage de programmation (formats pdf, postscript, svg ...)
- la **représentation bitmap** où l'image est quadrillée selon une matrice de *largeur* × *hauteur* pixels, chaque pixel codant la couleur d'un petit carré de l'image. Les fichiers images de type bitmap sont adaptés à la représentation de figures aux contours irréguliers, en photographie par exemple.

La couleur de chaque pixel peut être codée sur un **octet** (=8 bits). On associe à chacune des 256 valeurs de pixel (de 0 à $2^8 - 1 = 255$) une couleur, l'ensemble des 256 couleurs constituant une **palette**. Les images en niveaux de gris sont souvent codées ainsi avec une palette de 256 niveaux de gris (de 0 pour noir à 255 pour blanc) mais on peut avoir une palette de 256 couleurs comme dans les fichiers gif.

On peut aussi coder chaque pixel sur plus d'un octet. Si on utilise le système de représentation RedGreenBlue des couleurs, une couleur de pixel sera codée sur 3×8 bits par un triplet comme (255;128;240) de trois entiers compris entre 0 et 255. Les couleurs de pixels sont obtenues par synthèse additive alors que les couleurs des objets que nous voyons résultent d'une synthèse soustractive : le pixel de l'écran émet de la lumière, il ne la réfléchit pas comme un objet.

On peut rajouter un quatrième octet noté **alpha**, pour coder la **transparence**.

Le nombre de bits utilisés pour coder un pixel s'appelle la **profondeur** de l'image.

Les fichiers images peuvent être de grande taille, ils sont donc souvent compressés avec ou sans perte d'information. Les formats png et jpg sont des exemples de formats d'images bitmap compressés.

1.2 Traitement d'image en Python avec la bibliothèque PIL

PIL est un module (ou bibliothèque) Python permettant de traiter des images. C'est son fork Pillow qui est désormais mis à jour. La documentation est disponible à partir de la page web <http://effbot.org/imagingbook/pil-index.htm>.

Dans une image bitmap, les pixels sont repérés à partir d'une origine (0;0) située dans le coin supérieur gauche. L'axe des abscisses orienté vers la droite est le bord supérieur de l'image et l'axe des ordonnées, orienté vers le bas, le bord gauche. Ainsi 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.

PIL est constitué de plusieurs sous-modules permettant de créer et manipuler des objets à partir de fichiers images. Pour ouvrir une image avec PIL, on commence par importer son sous-module Image avec la directive `from PIL import Image` puis on ouvre le fichier image avec `im = Image.open('chemin_fichier')`. Les attributs (taille, format ...) stockés dans l'en-tête sont lus mais les pixels ne sont pas encore chargés en mémoire. On peut forcer cette action avec `pixels = im.load()`, les pixels de coordonnées (x; y) (intersection de la colonne x et de la ligne y) sont alors accessibles avec `pixels[x,y]`.

Les principaux modes d'images sont 'RGB' pour les images couleur de profondeur 24 bits et 'L' (luminance) pour les images en niveau de gris de profondeur 8 bits.

Principales fonctionnalités du module Image de PIL

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

Exercice 1

Créer un fichier source `exo1.py`.

1. Compléter le code de la fonction `ouvrir(fichier)` pour qu'elle ouvre le fichier puis affiche l'image et son mode, son format et la valeur du pixel (0;0). Tester cette fonction sur le fichier `lena.png`.
2. Dans la documentation du module Image de PIL, rechercher la méthode `convert`. Utiliser cette méthode pour convertir en niveaux de gris l'image `lena.png` puis l'enregistrer sur le disque sous le nom `lenagray.png`. Tester `ouvrir(lenagray.png)`
3. Créer une fonction `nuanciergris()` sans paramètre qui crée un nuancier de gris de dimensions 256×256 comme ci-contre. On utilisera deux boucles imbriquées pour parcourir tous les pixels de l'image. Modifier la fonction pour que le dégradé s'étale du haut vers le bas.
4. En mode 'RGB', un pixel vert est codé par (0,255,0). Créer une fonction `nuanciervert()`.



```

1 from PIL import Image
2
3 def ouvrir(fichier):
4     """ouvre un fichier image et affiche l'image et
5       quelques informations"""
6     img = Image.open(fichier)
7     .....
8     pixels = img.load()
9     print('Pixel (0;0) :', pixels[0,0])
10    img.show()

```

Désormais toutes les images que nous manipulerons seront en niveaux de gris (profondeur de 8 bits).

2 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 `getdata()` 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 2×2 avec la méthode `flatten()`.

exemple avec la console Python

```

1 >>> import os
2 >>> os.chdir('\MesDocuments\ProgrammesPython') #on règle le répertoire courant de la console
   Python
3 >>> from PIL import Image
4 >>> img = Image.open('lenagray.png')
5 >>> data = img.getdata()
6 >>> data[0] #pixel du coin supérieur gauche
7 162
8 >>> len(data)
9 262144
10 >>> 512**2
11 262144
12 >>> import numpy as np

```

```

13 >>> matrice = np.array(img) #un tableau de dimension 512x512
14 >>> matrice.shape
15 (512, 512)
16 >>> matrice[0][0] #pixel du coin supérieur gauche
17 162
18 >>> data2 = matrice.flatten() #conversion de matrice en un tableau à 1 dimension
19 >>> data2.shape
20 (262144,)
21 >>> data2[0]
22 162

```

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')`.

```

23 >>> import matplotlib.pyplot as plt
24 >>> plt.hist(data, bins=256, range=(0,255), facecolor='green') #on peut remplacer data par data2
25 >>> plt.show()
26 >>> 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`

<code>tableau = matplotlib.pyplot.imread('nom.ext')</code>	ouvre un fichier image et le stocke dans un array numpy
<code>matplotlib.pyplot.imsave(fname='nom.ext', arr=tableau)</code>	enregistre un tableau sous la forme d'un fichier image)
<code>matplotlib.pyplot.imshow(tableau, cmap='gray')</code>	insère l'image correspondant à un tableau dans une figure matplotlib avec l'option palette niveaux de gris
<code>matplotlib.pyplot.show()</code>	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 $[0; 1]$.

```

1 >>> a = np.array([[255, 200, 50], [0, 100, 129]])
2 >>> a.dtype
3 dtype('int32')
4 >>> b = a/255
5 >>> b
6 array([[ 1.          ,  0.78431373,  0.19607843],
7        [ 0.          ,  0.39215686,  0.50588235]])
8 >>> b.dtype
9 dtype('float64')

```

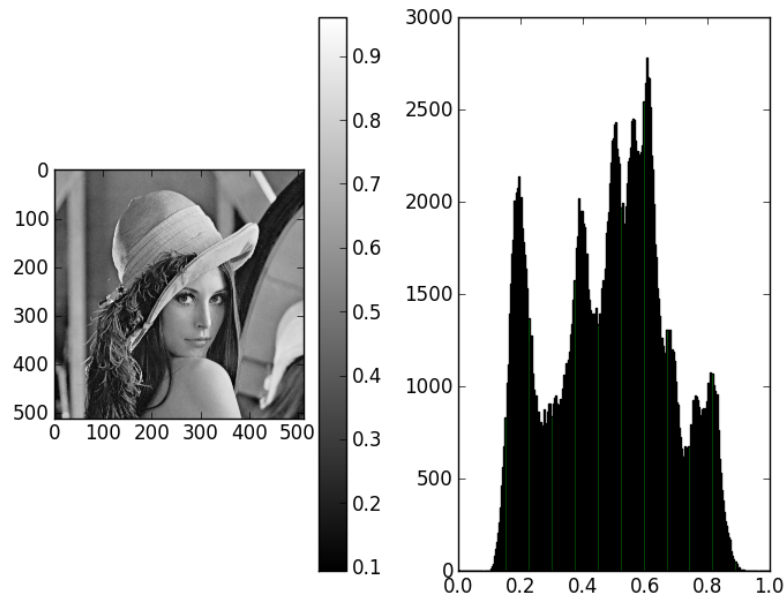
Exercice 2

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

```

1 import matplotlib.pyplot as plt
2
3 matrice2 = plt.imread('lena.png')
4 plt.subplot(121) #première sous figure
5 plt.imshow(matrice2,cmap='gray') #ajout de l'image à la figure avec la palette gray
6 plt.colorbar() #ajout d'une barre des couleurs de la palette
7 plt.subplot(122) #seconde sous figure
8 plt.hist(matrice2.flatten(), bins=256,range=(0,1),facecolor='green') #histogramme
9 plt.subplots_adjust(wspace=0.3) #ajustement de l'espace horizontal entre les subplots
10 plt.savefig('lena+histo.png') #enregistrement sur disque
11 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. Créer une fonction `affiche(matrice,palette='gray',histo=False,nom='image')` qui prend en entrée une matrice et selon la valeur du paramètre booléen `histo` affiche l'image correspondante et son histogramme ou l'image seule, puis qui enregistre la figure sous le nom passé en paramètre.

3 Repérage d'un pixel dans la matrice d'une image

La matrice d'une image bitmap de dimensions $W \times H$ (nombre de colonne fois nombre de lignes) est une liste de listes (ou un array si on utilise `numpy`) de dimensions $(H; W)$. En Python, la première dimension d'un tableau multidimensionnel est son nombre de lignes (ici H) et sa seconde dimension son nombre de colonnes (ici W). Mais dans le repère d'une image bitmap, d'origine le coin supérieur gauche, les lignes représentent les ordonnées et les colonnes les abscisses.

Pour accéder au pixel de coordonnées $(x; y)$ à partir d'une matrice de type array ou list on écrira donc `matrice[y][x]`. Avec PIL on peut procéder autrement : `pixels = im.load()` puis `pixels[x,y]` pour le pixel en $(x; y)$.

4 Opérations algébriques sur une image

Exercice 3

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 blancs.

Quelques fonctions de numpy qui nous seront utiles :

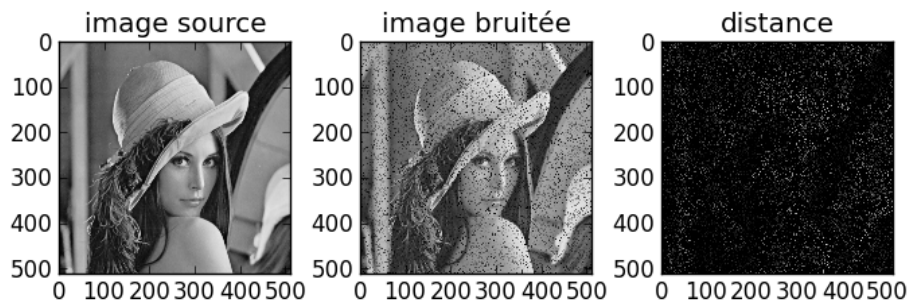
```

1 >>> m = np.zeros((2,3)) #pour créer un tableau de 2 lignes et 3 colonnes rempli de zéros
2 array([[ 0.,  0.,  0.],
3        [ 0.,  0.,  0.]])
4 >>> m+1 #un tableau rempli de 1
5 array([[ 1.,  1.,  1.],
6        [ 1.,  1.,  1.]])
7 >>> np.random.randint(0,512,5) #un tableau de 5 entiers aléatoires choisis entre 0 et 512
8 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.

La fonction `ajout(matrice1,matrice2)` retourne la matrice obtenue par ajout des matrices de deux images de mêmes dimensions, tous les pixels dépassant 255 sont fixés à 255.



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

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 def bruitage(matrice):
5     H,W = matrice.shape
6     matrice2 = matrice.copy()
7     for y in range(H):
8         bruit = np.random.randint(0,W-1,10)
9         for x in bruit:
10             matrice2[y][x] = 0
11     return matrice2
12
13 def distance(matrice1,matrice2):
14     H,W = matrice1.shape
15     if matrice1.shape != matrice2.shape:
16         return "Les images ne sont pas de même dimension"
17     else:
18         matrice3 = np.zeros((H,W))
19         .....
20
21 def ajout(matrice1,matrice2):
22     .....
23
24 matrice1 = plt.imread('lenagray.png')

```

```

25 matrice2 = bruitage(matrice1)
26 matrice3 = distance(matrice1,matrice2)
27 .....

```

5 Traitement d'image avec un filtre, partie 1

5.1 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 souvent 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`).

Pour le filtre d'inversion (exercice 4), on peut travailler sur une matrice de pixels compris entre 0 et 255, puis transformer la matrice obtenue en objet image PIL avec la fonction `Image.fromarray(matrice)`.

Exercice 4

Ecrire un script Python qui ouvre une image puis lui applique un filtre qui inverse les valeurs des pixels (fonction $x \mapsto 255 - x$ si les valeurs sont des entiers entre 0 et 255 ou fonction $x \mapsto 1 - x$ si elles sont dans $[0; 1]$)

5.2 Correction γ de la luminance

Exercice 5

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 $f(x) \geq x$

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.

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

3. Ecrire un script Python qui ouvre une image puis applique à sa matrice une fonction `filtre(matrice, fonction)` qui retourne la matrice de l'image filtrée avec la fonction passée en paramètre. Cette fonction peut être définie à part, éventuellement en important des fonctions du module `math`. On peut aussi utiliser l'opérateur `lambda` :

```

1 >>> fonction = lambda x : x**2
2 >>> fonction(2)
3 4

```

4. Sur l'image `lenagray.png`, tester ainsi plusieurs filtres de correction γ d'éclaircissement ou d'assombrissement.

Si on compose un filtre γ 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 3.

5.3 Accentuation de contraste

Exercice 6

- 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 $f(x) > x$

- 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;
```

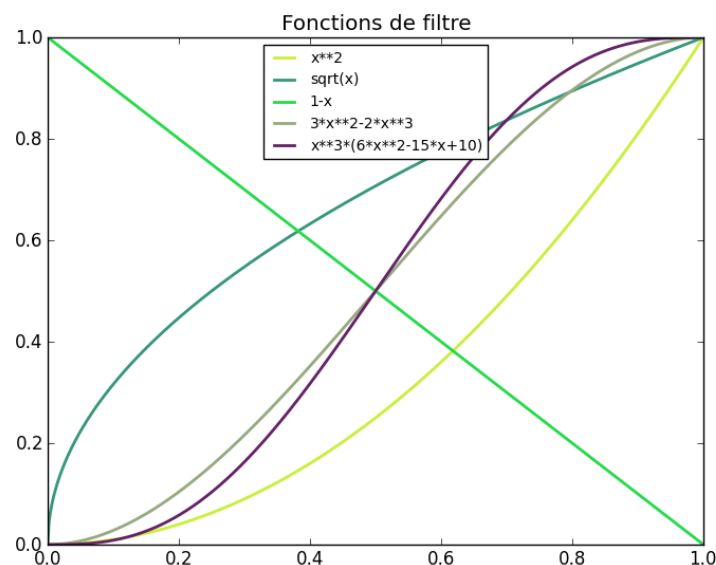
```
(%o1) f(x):=a x^3+b x^2+c x+d
```

```
(%i2) linsolve([f(0)=0,f(1)=1,f(1/2)=1/2,3*a+2*b+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.

- Appliquer un filtre de contraste à l'image `lenagray.png` avec la fonction `filtre(matrice,fonction)` définie dans l'exercice 5.



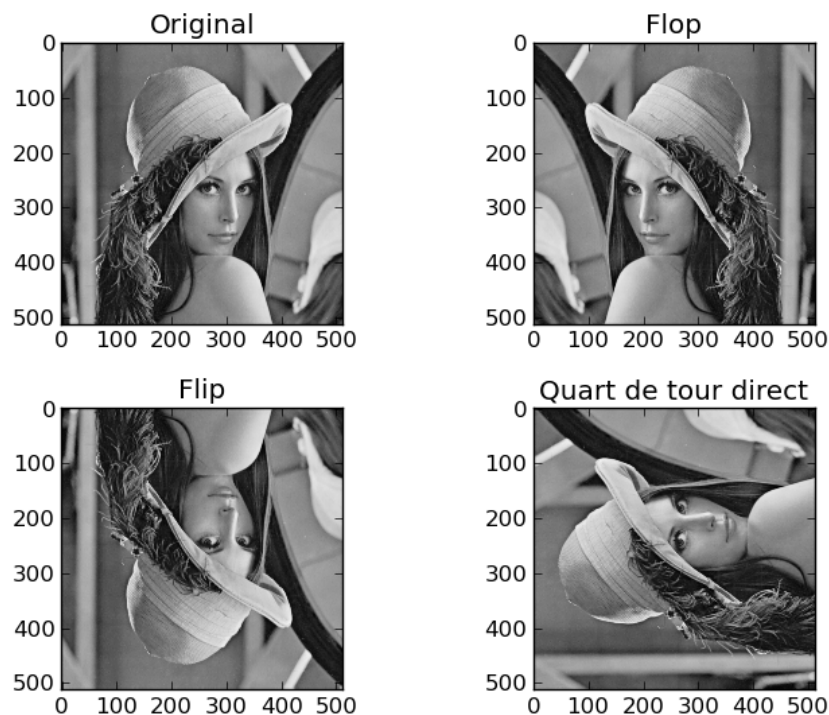
6 Transformations géométriques d'une image

Exercice 7

On peut appliquer une transformation géométrique à une image, s'il s'agit d'une bijection. Si on note $M1$ sa matrice on procède ainsi :

- on crée une matrice $M2$ de mêmes dimensions remplie de zéros

- on parcourt $M2$ et on affecte à $M2[y][x]$ la valeur du pixel de $M1$ antécédent de $M2[y][x]$ par la transformation
1. Ecrire un script avec une fonction `zoom(matrice, fact=1)` qui prend en entrée une matrice d'image et retourne en sortie la matrice de l'image zoomée d'un facteur donné.
L'image zoomée d'un facteur 0,5 contient-elle la même quantité d'informations que l'image initiale ? Et l'image zoomée d'un facteur 2 ?
Pour plus d'informations sur le thème du redimensionnement d'image, on pourra consulter le paragraphe *Array Interpolation schemes* de la page http://matplotlib.org/users/image_tutorial.html.
 2. Rassembler dans un script Python des fonctions `flip(matrice)`, `flop(matrice)`, `quart_tour_direct(matrice)`, `quart_tour_indirect(matrice)` qui prennent en entrée la matrice d'une image source et retournent en sortie la matrice de l'image obtenue par la transformation géométrique souhaitée. (réflexions, quarts de tour direct et indirect, demi-tour).

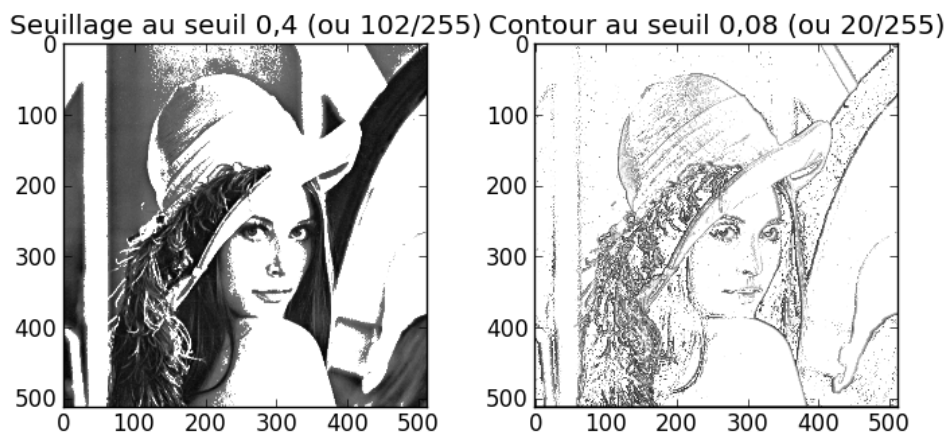


7 Traitement d'image avec un filtre, partie 2

7.1 Seuillage et Détection de contour

Exercice 8

1. Un filtre de seuillage consiste à mettre à 0 (noir) tous les pixels ayant une valeur inférieure à un certain seuil.
Ecrire une fonction `seuil(pixel, s)` qui retourne `pixel` si sa valeur est supérieure à `s` ou 0 sinon.
Appliquer un seuillage à l'image `lenagray.png` avec la fonction `filtre(matrice, fonction)` définie dans l'exercice 5.
2. Un algorithme de détection de contour simple peut consister à parcourir la matrice d'une image et à mettre à 0 (noir) le pixel en $(x; y)$ si son écart absolu avec le pixel du dessous en $(x; y + 1)$ ou avec celui de droite en $(x + 1; y)$, dépasse un certain seuil.
Ecrire une fonction `filtre_contour(matrice, seuil)` et appliquer ce filtre de contour à l'image `lenagray.png`.



7.2 Filtres par convolution

7.2.1 Principe

On choisit d'appliquer à une image A en niveaux de gris (plus simple) un traitement dit par filtre de convolution. Un filtre (ou noyau) de convolution est une matrice généralement de taille impaire, par exemple 3×3 comme $K = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$. L'application de ce filtre

K à la matrice de l'image ci-dessous consiste pour chaque pixel à superposer la matrice filtre avec la matrice source en la centrant sur le pixel et à effectuer la somme des produits coefficients par coefficients de la matrice filtre et de l'extrait de la matrice source qu'elle recouvre. Cette nouvelle valeur est affectée au pixel correspondant de l'image filtrée qui a mêmes dimensions que la source. On recommence le traitement pour tous les pixels de la source en déplaçant la matrice filtre. La figure ci-dessous donne un exemple de traitement.

	0	1	2	3	4	5
0	48	45	60	81	83	65
1	58	82	115	133	104	55
2	99	130	154	147	96	37
3	136	160	163	138	86	39
4	156	158	157	139	89	42
5	154	154	156	145	98	45

a	b	c
d	e	f
g	h	i

On associe au pixel de valeur 115 de la source le pixel de valeur $45 \times a + 60 \times b + 81 \times c + 82 \times d + 115 \times e + 133 \times f + 130 \times g + 154 \times h + 147 \times i$ dans l'image filtrée.

Dans Gimp, après avoir chargé une image, on accède aux filtres par convolution dans Filtre > Génériques > Matrices de convolution et on peut entrer sa matrice filtre 3×3 ou 5×5 .

Voir aussi l'aide <http://docs.gimp.org/2.6/fr/plugin-in-convmatrix.html>.

Si le filtre a pour dimensions 3×3 , un problème se pose pour les pixels sur les bords de l'image : on peut ignorer ces pixels qui resteront noirs (couleur par défaut) dans l'image filtrée ou étendre l'image initiale en rajoutant une bordure d'un pixel sur laquelle on duplique les pixels voisins, ou ceux de la ligne ou colonne opposée (voir les options de Bordure dans Gimp). Le problème est accru pour les filtres 5×5 .

Le dernier point concerne la normalisation. Si la somme des coefficients est non nulle, on divise le résultat de la convolution par cette somme (ou on choisit une matrice filtre dont la somme des coefficients est 1) afin de conserver la même luminance globale. Si la somme des coefficients vaut 0, on divise par 1. Enfin on peut ajouter un biais au résultat pour jouer sur la luminance globale de l'image filtrée (Dans Gimp on rajoute 128).

7.3 Exemples de filtres

On donne une liste de filtres 3×3 .

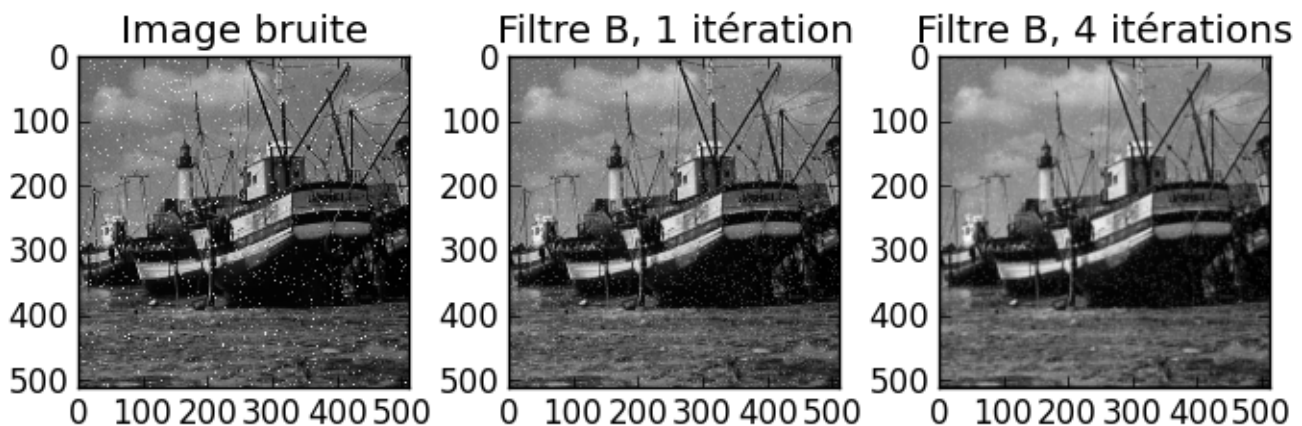
$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad C = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \quad D = \begin{pmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{pmatrix}$$

$$E = \begin{pmatrix} -3 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 3 \end{pmatrix} \quad F = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad H = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

On pourra tester avec Gimp sur les images `lenagray.png` ou `bateaubruite.png`.

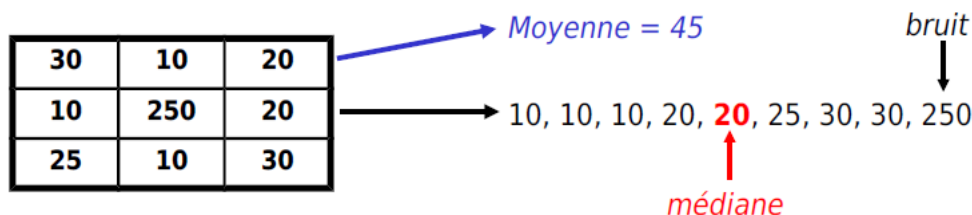
Exercice 9

1. Ecrire en Python une fonction `filtre(matrice, noyau)` qui retourne une nouvelle matrice d'image obtenue par application à une matrice d'image d'un filtre de convolution dont on donne le noyau.
2. Rajouter un paramètre `iterations` à la fonction précédente pour choisir le nombre de fois qu'on applique successivement le filtre par convolution.
3. Tester le filtre de convolution B avec trois itérations sur l'image `bateaubruite.png`.
Expliquer pourquoi ce filtre réalise un floutage de l'image, qu'on peut utiliser pour la débruiter.
Lequel parmi les filtres ci-dessus permet également de réaliser un floutage de l'image ?
4. Choisir trois autres filtres de convolution parmi ceux proposés, conjecturer leur effet puis vérifier avec le programme Python.



7.4 Floutage par filtre médian

Pour nettoyer le bruit dans une image, on peut lui appliquer plusieurs types de filtres comme le filtre moyenneur ou le filtre gaussien qui sont des filtres par convolution (voir exercice 8). Le filtre médian n'est pas un filtre par convolution (dit filtre linéaire) : il consiste à remplacer la valeur d'un pixel par la valeur médiane dans son voisinage $n \times n$.



Une méthode pour trier les listes ou les array :

```
>>> from random import* #module avec des générateurs de nombres pseudo-aléatoires
>>> randint(1,100) #retourne un entier au hasard entre 1 et 100
11
```

```
>>> L = [randint(1,100) for i in range(20)]
>>> L
[98, 5, 32, 28, 49, 15, 16, 30, 60, 66, 19, 19, 77, 39, 7, 46, 12, 65, 25, 50]
>>> L.sort() #la méthode sort trie sur place une liste, elle existe aussi pour les array numpy
>>> L
[5, 7, 12, 15, 16, 19, 19, 25, 28, 30, 32, 39, 46, 49, 50, 60, 65, 66, 77, 98]
>>> L.sort(reverse=True) #tri décroissant
```

Exercice 10

1. Ecrire en Python une fonction `filtre_median(matrice)` qui retourne une nouvelle matrice d'image obtenue par application d'un filtre médian de blocs 3×3 à l'image source.
2. Rajouter un paramètre `iterations` à la fonction précédente pour choisir le nombre de fois qu'on applique successivement le filtre médian.
3. Tester le filtre médian avec trois itérations sur l'image `bateaubruite.png`. Comparer avec le résultat obtenu grace à un filtre par convolution.

Table des matières

1 Ouverture d'un fichier image de format bitmap	1
1.1 Fichiers images en informatique	1
1.2 Traitement d'image en Python avec la bibliothèque PIL	1
2 Histogramme d'une image avec numpy et matplotlib	2
3 Repérage d'un pixel dans la matrice d'une image	4
4 Opérations algébriques sur une image	5
5 Traitement d'image avec un filtre, partie 1	6
5.1 Principe	6
5.2 Correction γ de la luminance	6
5.3 Accentuation de contraste	7
6 Transformations géométriques d'une image	7
7 Traitement d'image avec un filtre, partie 2	8
7.1 Seuillage et Détection de contour	8
7.2 Filtres par convolution	9
7.2.1 Principe	9
7.3 Exemples de filtres	10
7.4 Floutage par filtre médian	10