Préliminaires

pour utiliser les fonctionnalités de scilab en imagerie numérique, se reporter au chapitre 6 du polycopié du cours sur la prise en main de `sivp` et suivre les étapes ci-dessous :

- récupérer les fichiers `menuIN*.sce`, `TP4.sce`, `lena.pgm` et `lena_bruit.pgm`...à l'adresse : `http://perso.univ-rennes1.fr/philippe.roux/#mathsIN`
- Lancer Scilab et exécuter `menuIN*sce`
- Charger la toolbox Image (via le menu `IN_maths`)
- Charger l'image `lena.pgm` (via le menu `IN_maths`)
- stocker le tableau `image` dans la variable `image0`
- afficher l'image `lena.pgm` (via le menu `IN_maths`)
- afficher l'histogramme de l'image `lena.pgm` (via le menu `IN_maths`)

Exercice 1

Calculs statistiques sur une image, rapport signal/bruit

Le corps d'une image en niveaux de gris (de taille $n \times m$) est en fait un tableau de nombres entiers compris entre 0 et I_{max}(=255 pour une image sur 8 bits). On va commencer par faire quelques calculs sur les images :

- Pour calculer le niveau de gris moyen de l'image $\bar{I} = \frac{1}{n \times m} \sum_{i,j} I(i,j)$ faire en scilab `mean(image0)`
- Pour calculer l'écart-type des niveau de gris de l'image (par rapport au niveau moyen)

$$\sigma_I = \sqrt{\frac{1}{n \times m} \sum_{i,j} (I(i,j) - \bar{I})^2}$$

  faire en scilab `stdev(image0)`
- Pour calculer Le rapport signal bruit d'une image I (stockée dans la variable `image`) obtenue par traitement d'une image de référence I_0 (stockée dans la variable `image0`)

$$SNR = 10 \log_{10} \left(\frac{\sum_{i,j} I_0(i,j)^2}{\sum_{i,j} (I(i,j) - I_0(i,j))^2} \right)$$

  faire en scilab `10*log10(sum(image0.^2)/sum((image-image0).^2))` (bien faire attention aux opérations termes à termes !)

1. Calculer le niveau de gris moyen et l'écart-type pour l'image `lena.pgm`, comparer avec l'histogramme.
2. Calculer le niveau de gris moyen et l'écart-type pour l'image `lena_bruit.pgm`, comparer avec les résultats obtenus pour `lena.pgm`.
3. Calculer le rapport signal Bruit de l'image `lena_bruit.pgm`
4. Écrire une fonction `s=SNR(image0,image)` qui calcule le rapport signal bruit de `image` par rapport à l'image de référence `image0`.

Exercice 2

Modification des niveaux de gris, contraste d'une image

Une image étant un tableau nous pouvons lui appliquer une fonction (bien définie) pour en modifier les niveaux de gris : `image=f(image0)`. Nous allons passer à quelques modifications de bases sur les niveaux de gris.

1. Modifier l'image `image0` en utilisant les fonctions suivantes :

$$f_1(x) = 255\sqrt{\frac{x}{255}}, \quad f_2(x) = 255\left(\frac{x}{255}\right)^2$$

   commenter l'effet de la modification sur la luminosité de l'image, sur la valeur moyenne du niveau de gris, sur l'histogramme.

2. le négatif d'une image I_0 est une autre image I_{neg} dont les niveaux sont définis par

$$I_{neg}(i,j) = I_{max} - I_0(i,j) = I_{max}\left(1 - \frac{I_0(i,j)}{I_{max}}\right)$$

   faire le négatif de l'image `image0` avec scilab (afficher la nouvelle image pour vérifier).

3. le seuillage d'une image consiste à faire les modifications suivantes :

$$I_{seuil}(i,j) = \begin{cases} I_0(i,j) & si \quad I_0(i,j) > seuil \\ 0 & sinon \end{cases}$$

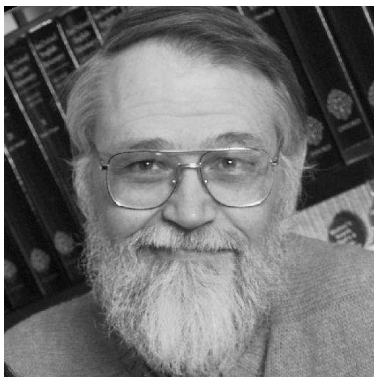
   seuiller l'image `image0` avec le $seuil = 127$ (afficher la nouvelle image pour vérifier).

4. Rendre une image binaire consiste à créer une image avec 2 niveaux de gris : 0 si $I(i,j) < seuil$ et 255 sinon. Binariser l'image `image0` avec le $seuil = 127$.

Exercice 3

Fondu enchainé





1. Charger les images `lena.pgm` et `brian_kernighan.pgm`
2. Créer une nouvelle image à partir d'un mélange composé de 50% des deux images :

```
image3=image1*0.5+image2*0.5
```

3. Créer une nouvelle image à partir d'un mélange composé de 25% de l'un des images et 75% de l'autre.
4. Faire un fondu enchaîné pour passer d'une image à l'autre, :

   pour $t = 0$ **jusqu'à** 1 **par** 0.05 **faire**
   mélanger les images à t% de l'une et $(1-t)$% de l'autre
   afficher l'image obtenue
   fin faire

Exercice 4

Ajouter du bruit à une image

Un bruit est une image dont les niveaux de gris $B(i,j)$ ont une répartition aléatoire, nous allons donc nous servir de `grand` pour ajouter du bruit à une image. On distingue plusieurs sortes de bruit :

- un bruit uniforme : les valeurs de $B(i,j)$ sont répartis uniformément autour d'un niveau moyen (1), ce type de bruit simule très bien le bruit thermique des capteurs CCD et CMOS. Pour créer un bruit uniforme de p% avec scilab et l'ajouter à `image0` :

```
-->bruitunif=(1+p*grand(image0,'unf',-1,1));
-->image=image0.*bruitunif;
-->image=max(image,0);image=min(image,255); //intervalle ramené à [0;255]
-->image=imnoise(image0,'speckle',p)//avec sivp
```

- un bruit binaire : les valeurs de $B(i,j)$ valent 0 ou 1 suivant que le pixel est affecté d'un bruit ou pas, ce type de bruit simule très bien le bruit lié aux pixels défectueux[1] d'un capteur CCD ou CMOS. Pour créer un bruit binaire de p% avec scilab (cas des pixels blancs seulement) et l'ajouter à `image0` :

```
-->bruitbin=grand(image0,'bin',1,p);
-->image=max(image0,255*bruitbin);
-->image=imnoise(image, 'salt & pepper', p);//avec sivp
```

1. créer une image `image1` bruité uniformément à 20%, calculer le niveau de gris moyen, l'écart-type et le SNR de `image1`, commenter.
2. de même créer une image `image2` bruité uniformément à 50%, calculer le niveau de gris moyen, l'écart-type et le SNR de `image2`, comparer avec les résultats obtenus pour `image1`.
3. créer une image `image3` avec un bruit binaire de 10%, calculer le niveau de gris moyen, l'écart-type et le SNR de `image2`, comparer avec les résultats obtenus pour `image1` et `image2`.

4. Créer 10 images `imageb` bruitées uniformément à 20% et en faire la moyenne `image4` (faire la somme et diviser par 10). Comparer le SNR de `imageb` avec ceux de `image1` et `image2`.

1. pixels chauds, bloqués au niveau de gris maximum, ou pixels froids, bloqués au niveau de gris minimum (car obstrué par une poussière par exemple)

Exercice 5

Multiplier une image par un masque

Pour masquer certaines zones d'une image il suffit de la multiplier (terme à terme) par une image de même taille ayant deux niveaux : 0 (pour les zones à masquer) et 1 (pour les zones qui restent visibles). Pour cela nous aurons besoin des normes vues en cours comme $\|(x,y)\|_2 = \sqrt{x^2+y^2}$ et de la fonction `bool2s` :

```
function z=n2(x,y)
z=sqrt((x).^2+(y).^2);//|| ||_2
endfunction
//taille de l' image
[nx,ny]=size(image0);
```

```
x=[1:nx];//listes des coordonnées x
y=[1:ny];//listes des coordonnées y
xc=256;yc=256;//centre de l'image
//masque de rayon 100 autour de (xc,yc)
masque=bool2s(feval(x-xc,y-yc,n2)>100);
imshow(uint8(255*masque));//affichage
```

1. masquer un disque, de 100 pixels de rayon, au centre `(xc,yc)` de l'image `image0` :

   ```
   image=image0.*masque;
   ```

2. masquer un voisinage, de 100 pixels de rayon, du centre de l'image `image0` en utilisant les normes $\|\cdot\|_\infty$ et $\|\cdot\|_1$ vues en cours.
3. masquer une zone ovale de l'image `image0` centrée sur le visage (voir l'image plus bas)

   $$\frac{(x-xc)^2}{a^2}+\frac{(y-yc)^2}{b^2}<1 \quad 2a = \text{largeur, } 2b = \text{hauteur, } (xc,yc) \text{ centre de l'ovale}$$

   Indication : *Utiliser le menu* `IN_maths->coordonnées dans l'image` *pour calculer la largeur (2a) et la longueur (2b) et la position du centre* (xc,yc) *de l'ovale à placer sur le visage.*
4. masquer les yeux par une bande noire (voir l'image plus bas)
   Indication : *s'inspirer de* $\|\cdots\|_1$ *et de la « norme ovale »*
5. créer un dégradé, comme l'image ci-dessous, en multipliant `image0` par le masque suivant :
   $masque(x,y) = \exp\left(-\frac{(x-xc)^2+(y-yc)^2}{100^2}\right)$ avec (xc,yc) centre de l'image

Exercice 6

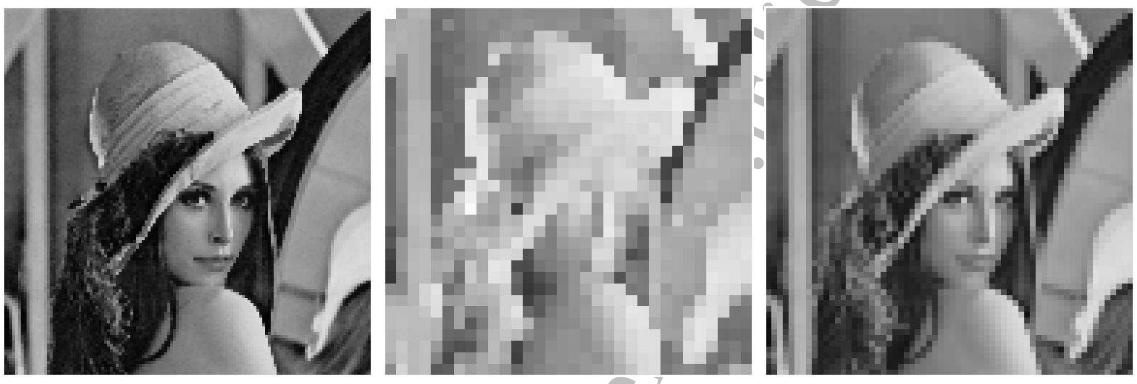
Pixeliser une image

Un autre moyen de "réduire" une image consiste à en diminuer le nombre de pixels, en les regroupant et en leur affectant un niveau de gris commum (le minimum des niveaux de gris du groupement ou le maximum ou la moyenne).

1. Écrire une fonction scilab : `function image2=pixeliser(image1,f,l,k)` qui découpe l'image en pixels de taille $l \times k$ et affecte à tous les pixels d'un même groupe G la valeur $m = f(G)$ comme ci-dessous :

$$G = \begin{array}{|c|c|c|}\hline I_0(i,j) & \dots & I_0(i,j+k) \\ \hline \vdots & & \vdots \\ \hline I_0(i+l,j) & \dots & I_0(i+l,j+k) \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|}\hline m & \dots & m \\ \hline \vdots & & \vdots \\ \hline m & \dots & m \\ \hline \end{array} \text{ avec } m = f(G)$$

2. Tester la fonction précédente avec l'image `image0` et :
   f=min, l=4=k, puis f=max, l=16=k, puis f=mean, l=8=k
   Vous devriez obtenir un résultat comparable aux images ci-dessous :



   Comparer le SNR des différentes images (par rapport à l'image de départ `image0`).
   Indication : *parcourir* `image1` *avec un pas l pour les lignes et un pas k pour les colonnes, pour sélectionner une région de taille $l \times k$ à partir du pixel (i,j) dans* `image1` *faire* `image1(i:i+l-1,j:j+k-1)`. *On testera avec les fonctions $f =$* `min,max,mean`.
3. À partir de la fonction `pixeliser` écrire un script pour "flouter" le visage de Lena comme ci-dessous :

Exercice 7

Binarisation d'une image

Dans une image en niveaux de gris codée sur n bits, l'intensité de chaque pixel peut varier entre 0 et $2^n - 1 = 255$ pour des images sur 8 bits comme dans ce TP. L'espace mémoire nécessaire pour la stocker est donc (au moins) égal au *nombre de pixels* $\times 2^n$. Dans cet exercice nous allons voir comment on peut diminuer la profondeur n d'une image en limitant la perte au niveau visuel.

1. Ramener la profondeur de `image0` à 1 bit, en utilisant l'algorithmes ci-dessous :

$$I(i,j) = \begin{cases} I_{max} & si \quad I_0(i,j) > seuil \\ 0 & sinon \end{cases}$$

   Créer les images correspondantes pour $seuil = 127$ (stockée dans `image1`) $seuil =$ niveau de gris moyen de `image0` (stockée dans `image2`)

2. Créer `image3` obtenue en ramenant la profondeur de `image0` à 2 bits, en utilisant :

$$I(i,j) = \begin{cases} 255 & si \quad I_0(i,j) > 190 \\ 170 & si \quad 190 \geq I_0(i,j) > 128 \\ 85 & si \quad 127 \geq I_0(i,j) > 63 \\ 0 & sinon \end{cases}$$

3. Pour être imprimée dans un journal une image en niveaux de gris doit être transformée en une image binaire (comme `image1` et `image2`), l'impression de niveau de gris étant donnée par la densité des pixels (blanc ou noir). Pour effectuer cette conversion on utilise un algorithme de "diffusion d'erreur" :

   fonction $I =$ diffuse_erreur(I_0)
   $I = I_0$, $p \times m =$ de taille l'image
   pour $i = 1$ **jusqu'à** p **faire**
   pour $j = 1$ **jusqu'à** m **faire**

$$I(i,j) = \begin{cases} I_{max} & si \quad I_0(i,j) > seuil (= 127) \\ 0 & sinon \end{cases}$$

   $erreur = I_0(i,j) - I(i,j)$
   répartir l'erreur sur les pixels voisins de (i,j) dans I_0
   comme ci-contre :

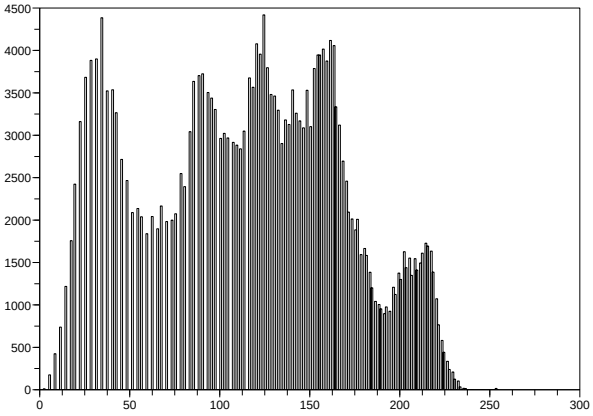
pixel (i,j)	$erreur \times \frac{1}{4}$
$erreur \times \frac{1}{4}$	$erreur \times \frac{1}{2}$

   fin faire
   fin faire

   Créer la fonction scilab `diffuse_erreur` et l'appliquer à `image0` et comparer le résultat `image4` aux précédents. Les résultats des images 1 à 4 doivent être comparables aux images ci-dessous :

Exercice 8

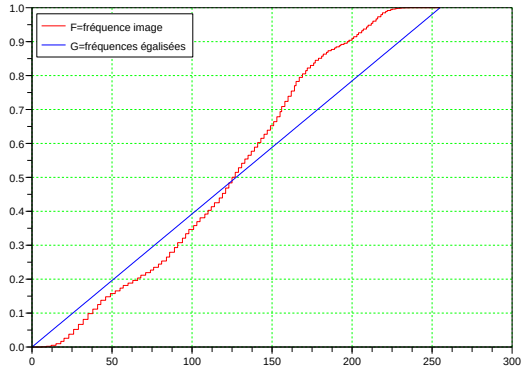
On peut interpréter l'histogramme d'une image comme le graphe de la fonction de densité $f(t)$ d'une variable aléatoire X (prenant ses valeurs entre 0 et 255 pour une image 8 bits) :



Sa fonction de répartition est une fonction continue, dérivable et croissante, donc une bijection de $[0, 255]$ vers $[0, 1]$ et est une primitive de f :

$$F(x) = \mathbb{P}(X \leq x) = \int_{-\infty}^{x} f(t)dt = \int_{0}^{x} f(t)dt$$

Elle se calcule numériquement en faisant la somme cumulée des fréquences de l'histogramme et en divisant par le nombre de données (des pixels ici) :



$$G(y) = \begin{cases} 0 & si \quad x < 0 \\ \displaystyle\int_0^y \frac{1}{255}dt = \frac{y}{255} & si \quad 0 \leq x \leq 255 \\ 1 & si \quad x > 255 \end{cases}$$

Un histogramme parfaitement égalisé correspondrait à une densité constante ou, en utilisant notre cours de probabilités, à une fonction de répartition linéaire (la courbe en bleu). On peut justement ramener la répartition de départ $F(x)$ à la répartition idéale $G(y)$ en faisant le changement de variable $t = H(s) = 255F(s)$ puisque :

$$G(y) = \int_0^y \frac{1}{255}dt = \int_0^x \frac{1}{255} \times 255F'(s)ds = \int_0^x f(s)ds = F(x) \qquad (\text{où } y = H(x))$$

Égaliser l'histogramme d'une image permet d'améliorer le contraste des zones trop uniforme d'une image, cela s'obtient en appliquant la fonction $H(s) = 255F(s)$ à l'image.

1. À partir de l'aide en ligne de la fonction `imhist` et des commandes ci-dessous :

```
[counts, cells] = imhist(uint8(image))
plot2d(cells,counts,5)
//à comparer avec
//histplot([0:255],double(image),normalization=%f)
```

   expliquer ce que contiennent les variables *counts* et *cells* ?

2. À partir des fonctions scilab `cumsum` et `imhist` tracer le graphe de la fonction de répartition associée à `image`.

3. Écrire et tester la fonction `egalise_histo` qui égalise l'histogramme d'une image :

   fonction $image2 =$ egalise_histo$(image1)$
   $image2 = image1$,
   $p \times n =$ taille de $image1$
   calculer la matrice des valeurs $y = H(x)$ dans l'intervalle $[0; 255]$
   pour tout $x \in [1; 2; \ldots 256]$ **faire**
   trouver les n° des pixels tel que $image1(ind) = x$
   modifier le niveau de gris de ces pixels dans le résultat $image2(ind) = H(x)$
   fin faire