# IMAGES NUMÉRIQUES MATRICIELLES EN SCILAB

Ce document, écrit par des animateurs de l'IREM de Besançon, a pour objectif de présenter quelques unes des fonctions du logiciel Scilab, celles qui sont spécifiques au traitement d'images numériques, à travers quelques exemples. Il s'agit ici de donner quelques éléments, « d'ouvrir quelques portes » pour qu'ensuite chacun puisse se promener librement, sans être nécessairement déjà un spécialiste de Scilab.
Nous ne parlerons ici que d'images numériques matricielles, c'est-à-dire d'images mémorisées sous la forme d'une matrice. C'est le cas des images au format .png ou au format .jpg.

## I - SCILAB - Module SIVP

### A - Scilab

Lorsque l'on ouvre le logiciel Scilab, une fenêtre nommée « **Console** » apparaît. Une instruction tapée dans cette fenêtre est exécutée de suite après la validation par la touche « Entrée » du clavier. Par exemple, la commande 4 + 3, suivie de « Entrée » donne « ans = 7 ».
Pour écrire un programme, et l'enregistrer, on ouvre en plus une fenêtre nommée « **SciNotes** » qui sert d'éditeur de texte : dans le menu, cliquer sur l'icône de gauche, représentant une feuille blanche .
Après enregistrement, le programme peut être exécuté à partir de Scinotes (menu Exécuter), ou bien en l'ayant au préalable copié dans la Console.

### B - Module SIVP (Scilab Image and Processing Video)

Le module SIVP permet d'utiliser les fonctions **imread**, **imshow** et **imwrite** dont nous parlons dans ce document. Il n'est pas installé systématiquement au moment où l'on télécharge Scilab, et pour l'installer, on peut :
– ou bien choisir le menu : Applications ; Gestionnaire de modules - ATOMS ; sélectionner SIVP et cliquer sur le bouton installer ;
– ou bien écrire dans la console : atomsinstall SIVP.
On redémarre ensuite Scilab, et le module sera alors définitivement chargé : à chaque démarrage, on verra écrit dans la console « Start SIVP . . . ».

La fonction « **imread** » affiche la matrice associée à une image donnée, la fonction « **imshow** » visualise l'image associée à une matrice donnée, et la fonction « **imwrite** » enregistre cette image. Des explications complémentaires sont donnes dans le paragraphe ci-dessous.

## II - LECTURE ET ÉCRITURE D'UNE IMAGE MATRICIELLE

### A - Images en niveaux de gris

1. **Lecture de la matrice d'une image : fonction « imread »**
   La fonction « **imread** » permet, étant donnée une image, de faire afficher la matrice qui lui est associée.
   La photo ci-contre a été prise avec un appareil photo numérique, puis transformée en niveaux de gris à l'aide du logiciel de traitement d'image GIMP.
   Elle est constituée de 21 × 32 pixels, chaque pixel (ou « picture element ») étant un petit carré gris plus ou moins foncé.

   Cette image (ou photo) est mémorisée sous la forme d'une matrice constituée de 21 lignes et de 32 colonnes, elle est composée de nombres entiers compris entre 0 et 255. Ces nombres correspondent aux nuances de gris de chacun des pixels, ils vont de 0 pour le noir à 255 pour le blanc, le nombre étant d'autant plus grand que le gris est clair. **Pour afficher la matrice associée à cette image**, il suffit de taper dans la console de Scilab, l'instruction : **imread('nom du fichier')**
   Le nom du fichier est entre apostrophes, on écrit tout le « chemin », sans oublier l'extension. Par exemple, si l'image s'appelle Musiciens21x32-gris, sauvegardée au format png, dans le dossier Photos, lui-même sur une clé USB, cette clé étant connectée au port E de l'ordinateur, en écrivant : G=imread('E :\Photos\Musiciens21x32-gris.png'), on obtient une matrice G de 21 lignes et 32 colonnes.

2. **Visualisation et sauvegarde de l'image associée à une matrice : fonctions « imshow » et « imwrite »**
Il s'agit, étant donnée une matrice, de faire afficher l'image correspondante, ou/et de l'enregistrer.

(a) Création d'une matrice d'image

Nous cherchons à réaliser une image constituée de bandes verticales allant du noir (0) au blanc (255), en passant par les 256 nuances de gris possibles. Pour cela, nous allons écrire une matrice constituée de $\ell$ colonnes de 0, $\ell$ colonnes de 1, $\ell$ colonnes de 2, etc..., $\ell$ représentant la largeur (en pixels) d'une bande. De façon à pouvoir utiliser des paramètres (la hauteur $h$ de l'image, la largeur $\ell$ des bandes verticales), nous allons utiliser une fonction ; voici ci-dessous pour commencer un exemple de fonction élémentaire (qui n'a rien à voir avec le traitement d'images, c'est juste pour montrer comment on programme une fonction) :

*En écrivant et en exécutant le petit programme ci contre, puis en tapant ensuite la commande $f(4)$, on obtient « ans=−3 », c'est-à-dire l'image de 4 par $f$.*

```
function y=f(x)
y=x^2-5*x+1
endfunction
```

Voyons maintenant comment créer notre image de bandes verticales.

Écrire le programme ci-contre dans Scinotes :

« degrade » est le nom que nous avons décidé de donner à la fonction, $h$ désigne le nombre de lignes de la matrice, et $\ell$ la largeur (en pixels) d'une bande verticale. La matrice, que nous avons décidé de nommer res, aura donc $256 \times \ell$ colonnes ; res($i$,$j$) est le terme correspondant à la ligne $i$ et à la colonne $j$ de cette matrice.

```
function res=degrade(h,ℓ)
for i=1 :h
    for j = 1 :256*ℓ
    res(i,j)=floor((j-1)/ℓ))
    end
end
endfunction
```

Après avoir enregistré ce programme et l'avoir exécuté, (soit depuis SciNotes, soit après l'avoir « Copié/Collé » dans la console), en tapant (dans la console) : degrade(60,2), on obtient une matrice de hauteur 60, constituée de deux colonnes avec des 0, deux colonnes avec des 1, etc...

Les nombres entiers sont écrits avec un point, ce qui signifie qu'ils sont reconnus au format « double » (voir explications complémentaires sur les formats p.4), et écrits ainsi le logiciel ne reconnaît pas qu'il s'agit d'une matrice d'image. Pour que Scilab reconnaisse que la matrice est une matrice d'image, nous allons transformer les nombres en éléments de $\mathbb{Z}/256\mathbb{Z}$, en utilisant la fonction **uint8**. Pour faciliter la suite nous donnons un nom (Mire) à cette matrice : Mire = uint8(degrade(60,2)).

(b) Visualisation de l'image : fonction « **imshow** »

Pour visualiser l'image associée à cette matrice, on utilise la fonction **imshow**, en mettant en argument le nom de la matrice. En tapant (dans la console) l'instruction : imshow(Mire), on obtient l'image ci-dessous, qui apparaît dans une nouvelle fenêtre :

(c) Enregistrement de l'image : fonction « **imwrite** »

La fonction imshow permet de voir l'image mais ne l'enregistre pas ; pour la sauvegarder, on utilise la fonction **imwrite** : comme premier argument, le nom de la matrice, et comme deuxième argument, entre apostrophes, le nom que l'on souhaite donner à l'image, avec tout le chemin, sans oublier l'extension. Par exemple pour une sauvegarde sur une clé connectée au port E, sous le nom de fichier Essai, on tape : imwrite(Mire,'E :\Essai'). La réponse « ans=1 » dans la console signifie que la sauvegarde a été faite.

## B - Images en couleur

Lorque l'on prend une photo avec un appareil numérique, ayant par exemple deux millions de pixels, cela signifie qu'il comporte deux millions de capteurs ; chaque capteur mesure les quantités de lumière rouge, de lumière verte et de lumière bleue reçues, et les enregistre sous forme de nombres : ainsi à chaque capteur est

associé un triplet (R,V,B) de nombres entiers compris entre 0 et 255, et à une image en couleur sont associées trois matrices (la première pour le rouge, la deuxième pour le vert et la dernière pour le bleu). Chaque matrice comporte, dans ce cas, deux millions de termes. Nous appellerons hypermatrice cet ensemble de trois matrices.
Les fonctions imread, imshow et imwrite s'utilisent avec les images en couleur comme avec les images en niveaux de gris.

1. **Lecture de l'hypermatrice d'une image**

   La « photo des Musiciens », avant sa transformation en niveaux de gris, était en couleur..., constituée de $21 \times 32$ pixels, chaque pixel étant un petit carré de couleur.

   Avec l'instruction imread, on obtient 3 matrices, chacune comportant 21 lignes et 32 colonnes, notées respectivement ( :, :,1), ( :, :,2) et ( :, :,3) dans la console. Ces matrices donnent, pour l'ensemble des pixels, les valeurs de Rouge (matrice ( :, :,1)) , de Vert (matrice ( :, :,2)) et de Bleu (matrice ( :, :,3)).

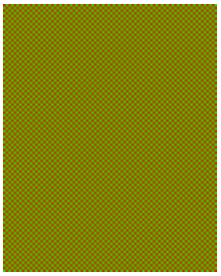
2. **Écriture de l'hypermatrice d'une image**

   On cherche ici à réaliser un damier alternant des pixels rouges, ayant pour coordonnées RVB : (255,0,0), et des pixels verts, ayant pour coordonnées RVB : (0,255,0). La matrice du rouge est donc une alternance de 255 et de 0, celle du vert de 0 et de 255, celle du bleu ne contient que des 0.

   Les paramètres $m$ et $n$ représentent respectivement le nombre de lignes et le nombre de colonnes du damier.

```
function res=damier(m,n)
for i=1 :m
    for j = 1 :n
        if floor((i+j)/2)=(i+j)/2    then
          res(i,j,1)=255
          res(i,j,2)=0
        else res(i,j,1)=0
          res(i,j,2)=255
        end
      res(i,j,3)=0
    end
end
res=uint8(res)
endfunction
```

Avec damier(100,80), et l'utilisation de la fonction imshow, on obtient le damier ci-dessous :



Remarque : en affichant ce document en pdf à une échelle proche de 100%, mais différente de 100%, on fait apparaître des effets de moirés assez jolis.

# III - TRANSFORMATIONS D'IMAGES

Nous allons maintenant transformer des images en travaillant directement sur les matrices associées. On peut intervenir sur la valeur numérique des pixels (pour éclaircir l'image, accentuer les contrastes, etc...), ou bien sur la localisation des pixels (pour effectuer des transformations comme des symétries, des rotations, ou encore la transformation du « Photomaton »).
Le temps mis pour effectuer ces programmes peut être assez long, c'est pourquoi les exemples que nous présentons sont avec des images en niveaux de gris.

## A - Intervenir sur la valeur numérique des pixels

1. **Format « uint8 » et format « double »**

   Par défaut, Scilab écrit les nombres au format « double »(sur 64 bits). Pour écrire les éléments des matrices d'images qui sont des nombres entiers compris entre 0 et 255, c'est-à-dire les représentants compris entre 0 et 255 des éléments de $\mathbb{Z}/256\mathbb{Z}$, on a utilisé le format « uint8 » (sur 8 bits).

   En réalité, la matrice est également reconnue par Scilab comme une matrice d'images si ses éléments sont des nombres compris entre 0 et 1, écrits au format « double ». Nous avons fait ce choix du format « uint8 » d'une part parce que lorsque l'on lit (avec la fonction « imread ») la matrice d'une image, c'est dans le format « uint8 » qu'elle est affichée ; et d'autre part parce que cela nous paraît préférable pour comprendre l'aspect discret du codage des couleurs.

   Dans ce qui suit, nous allons utiliser des fonctions faisant intervenir des nombres autres que des entiers compris entre 0 et 255. Par exemple, supposons que l'on veuille multiplier la valeur de tous les pixels par 0,7. Le nombre 0,7 est au format « double », les nombres de la matrice au format « uint8 » ; dans ce cas, Scilab effectue les calculs en mettant tous les nombres au format « uint8 », et alors 0,7 est remplacé par 0. Par exemple, 0.7*78 donne ans=54.6, mais 0.7*uint8(78) donne ans=0.

   Il faudra donc être attentif au type de nombre sur lesquels on travaille, et pour calculer $k$*M, on commencera par convertir la matrice en une matrice de nombres au format « double », et on reviendra ensuite à une matrice d'image avec la fonction uint8.

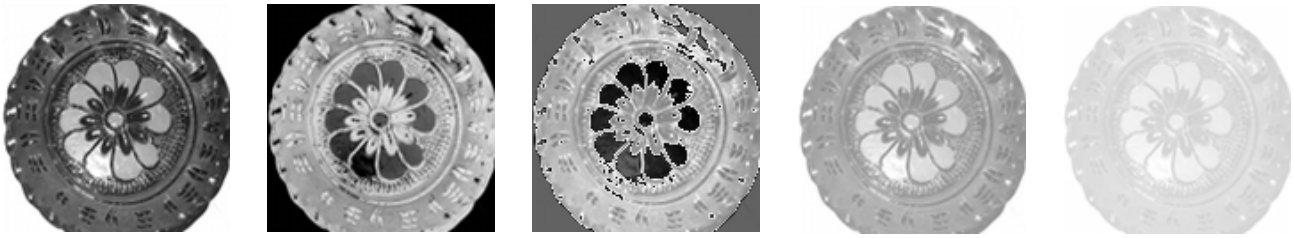
2. **Transformations affines**

   L'instruction « M+p » ajoute p à chacun des éléments de la matrice, et de même « $k$*M » multiplie chaque élément de la matrice par $k$.

```
function res=affine(M,m,p)
res=m*double(M)+p
res=iunt8(res)
imshow(res)
endfunction
```

   Un premier cas particulier est celui où l'image est transfomée en son négatif : pour obtenir un tel effet, on transforme le blanc en noir, le noir en blanc, et tout niveau de gris $x$ en $255 - x$. L'instruction « affine(M,-1,255) » permet d'obtenir ce résultat. Voici quelques exemples d'images transformées par des fonctions affines :

M | 255-M | M+100 | 0,5*M+127 | 0,2*M+204



3. **Éclaircir ou foncer une image**

   Nous cherchons à éclaircir une image, et donc à rapprocher les valeurs des niveaux de gris de 255.

   Une première idée consisterait à ajouter par exemple 100, mais alors toute valeur supérieure à 156, deviendrait dans $\mathbb{Z}/256\mathbb{Z}$, une valeur comprise entre 0 et 100, et ainsi les parties les plus claires de l'image deviendraient foncées (voir ci-dessus l'image correpondant à ce cas). D'autres fonctions affines conviennent, voir les exemples ci-dessus.

Essayons maintenant avec une fonction autre qu'une fonction affine de foncer une image. Pour foncer une image (et donc pour que les valeurs des pixels de l'image transformée soit plus petites que celles de l'image initiale), on va utiliser la fonction $f$ du second degré : $f(x) = \frac{x^2}{255}$ ; $f$ transforme 0 en 0, 255 en 255, et pour les autres valeurs $x$ de [0 ;255], on a bien $f(x) < x$.

L'instruction « uint8((double(M).^2)/255 » permet d'obtenir la matrice souhaitée et le résultat ci-contre

Dans ce cas, il faut faire attention à ne pas confondre le produit matriciel avec un produit élément par élément : l'instruction A*B correspond à un produit matriciel, tandis que A.*B (avec un point avant le symbole d'opération) est la matrice des produits des coefficients A(i,j)*B(i,j).

## B - INTERVENIR SUR LA POSITION DES PIXELS

### 1. Symétries axiales

Le programme ci-contre correspond à une symétrie d'axe horizontal : la fonction SH envoie la première ligne à la dernière, etc....

La fonction size renvoie un vecteur qui donne le nombre de lignes et de colonnes de la matrice, T(1) est donc ici le nombre de lignes de M, et T(2) le nombre de colonnes.

Le programme s'adapte facilement pour une symétrie d'axe vertical. On peut aussi aisément avoir une symétrie à 45° en utilisant la matrice transposée (avec Scilab, la transposée de M se note M').

```
function res=SH(M)
T=size(M)
for i=1 :T(1)
    for j = 1 :T(2)
   res(i,j)=M(T(1)-i+1,j)
   end
end
res=uint8(res)
imshow(res)
endfunction
```

Image initiale

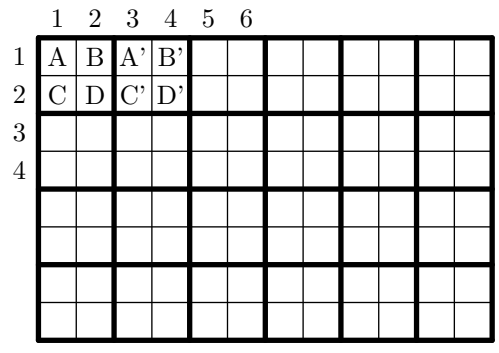
Symétrie horizontale

Symétrie verticale

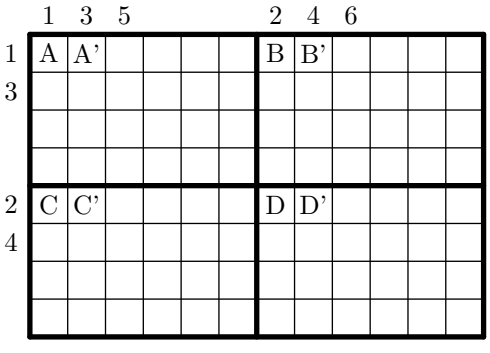
Transposition

### 2. Transformation « Photomaton »

Cette transformation a la particularité magique qu'en la réitérant le nombre de fois nécessaires, on retrouve l'image initiale. Elle est expliquée plus en détail dans le diaporama intitulé , consultable en ligne sur le site de l'IREM de BESANÇON, et en particulier la « magie » y est révélée ! Rappelons ici brièvement le principe : on découpe la matrice initiale en petits carrés de 2 × 2 (on travaille sur des matrices ayant un nombre pair de lignes ($n$) et un nombre pair de colonnes ($m$)), et la matrice d'arrivée en quatre matrices ayant $n/2$ lignes et $m/2$ colonnes. Puis, on range les 4 éléments de chaque petit carré ainsi :




Matrice initiale




Matrice photomaton

Les lignes impaires de la matrice initiale se retrouvent donc dans la moitié supérieure de la matrice photomaton, les lignes paires dans la moitié inférieure. De même les colonnes impaires se retrouvent dans la partie gauche, et les colonnes paires dans la partie droite.

Ainsi, si on appelle $x$ un élément se situant à la ligne $i$ et à la ligne $j$ de la matrice initiale.

– Si $i$ est impair, $x$ se retrouvera à la ligne $i' = \frac{i+1}{2}$.

– Si $i$ est pair, $x$ se retrouvera à la ligne $i' = \frac{i+n}{2}$.

On obtient des formules identiques pour exprimer la colonne où se retrouvera l'élément $x$.

Voici ci-dessous un programme permettant d'appliquer la tranformation « Photomaton » à une matrice M, ainsi qu'un exemple de ce que l'on peut obtenir.

La fonction « photomaton1 » applique une fois la transformation à la matrice M, et puis affiche l'image obtenue après transformation ; la fonction diaporama1 réitère cette même transformation k fois en affichant à chaque étape les images obtenues (si l'on choisit bien k, on revient à l'image initiale).

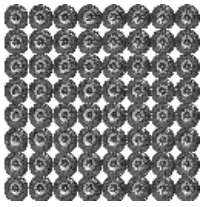
L'image utilisée ci-dessous est constituée de 128×128 pixels, on revient à l'image initiale en 7 itérations. Les images ci-dessous correspondent à ces itérations successives.

Image initiale

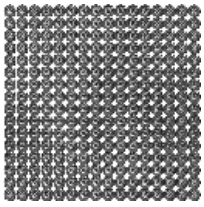
Étape 1

Étape 2

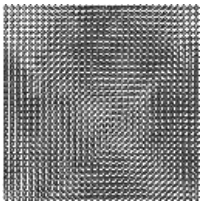
Étape 3



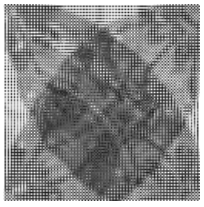
Étape 4



Étape 5



Étape 6



Étape 7

Programme pour **une** itération :

```
function res=photomaton1(M)
T=size(M)
for i=1 :T(1)/2
    for j = 1 :T(2)/2
     if floor(i+1)/2==(i+1)/2    then
       l=(i+1)/2
       else l=(i+T(1))/2
     end
     if floor(j+1)/2==(j+1)/2    then
       c=(j+1)/2
       else c=(j+T(2))/2
     end
   res(l,c)=M(i,j)
   end
end
res=uint8(res)
imshow(res)
endfunction
```

Programme pour **k** itérations :

```
function res=diaporama1(M,k)
for i = 1 :k
M=photomaton1(M)
end
endfunction
```

Nous proposons un autre programme, plus long dans l'écriture du programme, mais plus rapide dans l'exécution, ce qui peut être intéressant lorsque le nombre d'itérations pour revenir à l'image initiale est élevé, ou bien si l'on veut adapter le programme pour une image en couleur.

Le principe est le suivant : Un élément $x$ qui se trouve à la ligne $i'$ de la matrice photomaton provient de la ligne $i$ telle que :
– Si $i' \leqslant \frac{n}{2}$, $i = 2i' - 1$.
– Si $i' > \frac{n}{2}$, $i = 2i' - n$.

On a des formules identiques pour exprimer la colonne d'où provient l'élément $x$.

La matrice « res » est remplie quart par quart dans l'ordre suivant :
– quart en haut à gauche ;
– quart en bas à gauche ;
– quart en haut à droite ;
– quart en bas à droite.

Le programme ci-contre est donné pour une itération. Pour obtenir k itérations, reprendre le programme précédent.

Remarque : la distinction des 4 cas n'est pas indispensable, on peut utiliser la formule plus synthétique :
res(*i*,*j*)=M(2**i*-1-(T(1)-1)*floor(2*i*/(T(1)+1)),2**j*-1)-(T(2)-1)*floor(2*i*/(T(2)+1)),
mais l'exécution du programme est plus longue.

Programme pour **une** itération :

```
function res=photomaton2(M)
T=size(M)
for i=1 :T(1)/2
    for j = 1 :T(2)/2
   res(i,j)=M(2*i-1,2*j-1)
   end
end
for i=T(1)/2+1 :T(1)
    for j = 1 :T(2)/2
   res(i,j)=M(2*(i-T(1)/2,2*j-1)
   end
end
for i=1 :T(1)/2
    for j = 1+T(2)/2 :T(2)
   res(i,j)=M(2*i-1,2*(j-T(2)/2))
   end
end
for i=T(1)/2+1 :T(1)
    for j = 1+T(2)/2 :T(2)
   res(i,j)=M(2*(i-T(1)/2,2*(j-T(2)/2))
   end
end
res=uint8(res)
imshow(res)
endfunction
```