TP1 TIM : Initiation à Matlab

POUVARET Line, BENAOUN Hamdi

Représentations des scalaires et des tableaux (matrices, vecteurs)

a=5	MatLab nous affiche *a* = 5 et on remarque que dans l'espace de travail, la variable *a* a été créée avec la valeur 5 affectée.
b=6	MatLab nous affiche *b* = 6 et là aussi, la variable *b* a été créée avec 6 en valeur affectée.
b	On demande ici la valeur de la variable b. MatLab nous affiche donc b = 6 à ce moment de l'exécution.
a=[1 2 ; 3 4 ; 5 6]	Ici la variable *a* va contenir une matrice à 2 colonnes et 3 lignes et la valeur précédente (5) a été écrasée.
[lig, col]=size(a) ;	On récupère la taille de la matrice contenue dans *a* et on place les valeurs dans deux variables *lig* et *col* qui vont respectivement contenir le nombre de lignes et le nombre de colonnes de *a.*
lig	MatLab affiche le contenu de la variable *lig* : 3
col	MatLab affiche le contenu de la variable *col* : 2
a(1, 2)	MatLab nous affiche la valeur de la case de la matrice contenue dans *a* à la ligne 1, colonne 2 c'est-à-dire *a*(1,2) = 2
a(1, :)	MatLab nous affiche toutes les valeurs de la matrice de *a* sur la ligne 1 (et en ligne), c'est-à-dire *a*(1, :) = 1 2
a(:, 2)	MatLab nous affiche toutes les valeurs de la matrice de *a* sur la colonne 2 (et en colonne), c'est-à-dire *a*(:,2) = 2 4 6 (verticalement) On conserve donc le vecteur.
t=(0 :0.1 :10);	*t* est un tableau de flottants de 0 à 10 avec un pas de 0.1. (c'est une sorte de boucle for)
size(t)	MatLab nous affiche 1 101, c'est-à-dire qu'on a 1 ligne mais 101 colonnes.
length(t)	MatLab nous affiche directement 101 qui représente le nombre de flottants dans *t*.
t(1 :10)	MatLab nous affiche les 10 premières valeurs de *t* donc de 0 jusqu'à 0.9000
b=[10 ;20 ;30]	On stocke dans la variable *b* une matrice à une colonne et à 3 lignes.
b'	*b*' nous affiche la matrice sur 3 colonnes et 1 ligne correspondant aux valeurs de la matrice contenue dans *b.* Les lignes de *b* deviennent les colonnes de *b'* et les colonnes de *b* deviennent les lignes de *b'.*
a=a(1 : 2, 1 : 2)	*a* est modifiée et correspond à la matrice *a* de départ dont on a gardé les 2 premières lignes et les 2 premières colonnes (on l'a « rognée »)

b=10*ones(2, 2)	*b* va contenir une matrice carrée de 2 lignes, 2 colonnes avec uniquement des valeurs 10. ones(2,2) est une matrice carrée contenant uniquement des 1
c=a*b	*c* correspond au produit matriciel *a* x *b*
c=a.*b	*c* correspond au résultat de la multiplication par blocs de la matrice *a* avec la matrice *b*

Format d'image PGM

Image_a.pgm a une taille de 270 Ko.

Image_b.pgm a une taille de 78 Ko.

On remarque que quand on ouvre les fichiers dans un éditeur de textes (par exemple), image_a.pgm apparaît sous la forme d'un grand tableau de données (on suppose que chaque case correspond à la valeur de niveau de gris de chaque pixel) avec également les dimensions de l'image en première ligne et image_b.pgm apparaît sous une forme de fichier codé (sûrement compressé).

Lire, transformer et afficher une image

Instructions complétées et/ou écrites dans le programme :

```
valPix = ima(25,82);

oneLig = ima(numLig, :);
oneCol = ima( :, numCol);
```

Quelle est la taille de l'image, en ligne et en colonne ?

352 lignes, 224 colonnes.

Entre quelles valeurs minimale et maximale varient les niveaux de gris ?

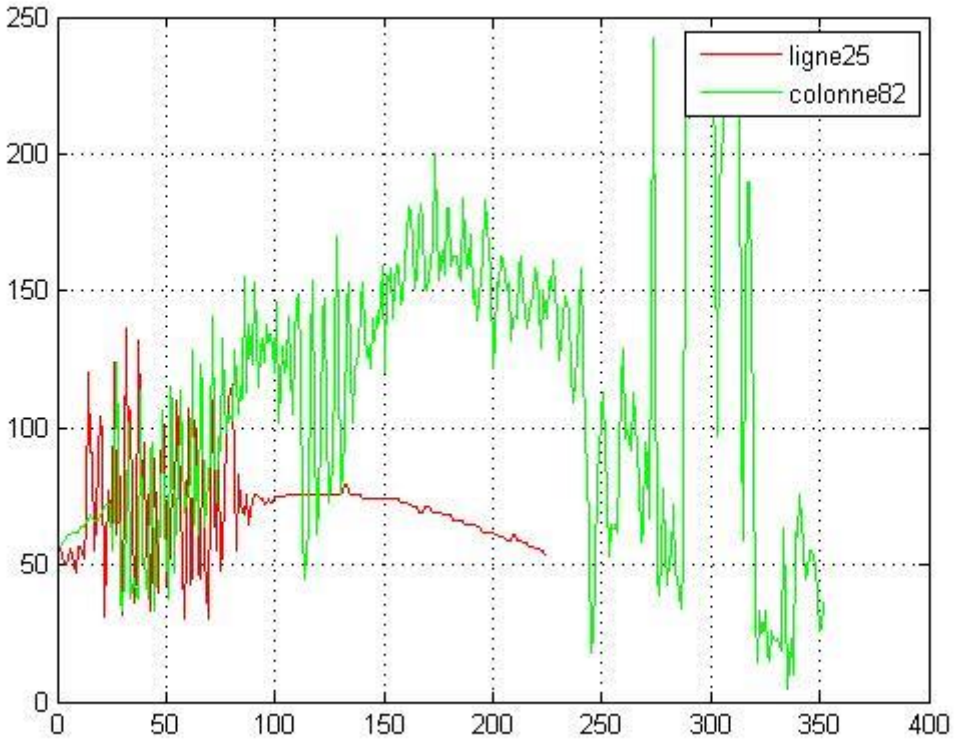
Minimum : 0, Maximum : 255

Quelle est la valeur du niveau de gris du pixel à la ligne 25 et la colonne 82 ?

55

Relever les profils de niveaux de gris de la ligne 25 et de la colonne 82 et affichez-les.

Les profils s'affichent sur l'image sous la forme d'une ligne en rouge et une colonne en verte et celles-ci se croisent au pixel (25,82)



Comment a été réalisée l'image appelé imaOut ? Que représente-t-elle ?

C'est l'image originale codée sur 8 bits et pour décider quel pixel sera blanc ou noir, on considère que le seuil est à 200 (par rapport à l'image originale). Donc un pixel qui avait un niveau de gris en dessous de 200 sera représenté en noir sur l'image transformée et à partir de 201, il sera représenté en blanc.

Transformation d'une image couleur en niveaux de gris

Instructions complétées et/ou écrites dans le programme :

```
imaGrisInverse = 255-imaGris;
imaInverse     = 255-ima;
```

Quelle est la taille de l'image renvoyée par l'instruction size(ima) ? Comment comprenez-vous ce résultat ?

On obtient : 384 256 3

Notre image est représentée sous la forme d'une matrice à 3 dimensions dont le nombre de ligne est égal à 384, le nombre de colonnes à 256 et 3 est la troisième dimension correspondant au nombre de couleurs (RVB).

Expliquez l'instruction pour créer l'image en niveau de gris à partir de l'image couleur.

```
imaGris = 0.299*ima(:,:,1)+0.587*ima(:,:,2)+0.114*ima(:,:,3);
```

Notre image en gris est représentée sous la forme d'une matrice à 2 dimensions, donc on fait une projection orthogonale d'une matrice à 3 dimensions vers une matrice à 2 dimensions et on applique les coefficients de l'équation de la luminance à chaque pixel d'une couleur en particulier. On supprime donc la dimension des couleurs.

Transformer ces 2 images (en couleur et en niveau de gris) par le procédé de « vidéo inversé ». Une image en vidéo inversée se construit en affectant une forte valeur (niveau clair) à un niveau sombre et inversement et cela selon une loi linéaire telle que illustrée ci-dessous. Expliquer à quoi s'applique cette fonction. A quoi est égal OutMax ? Même question pour InMax ? Donner l'équation de cette droite. Ecrire les instructions nécessaires dans le programme pour effectuer les transformations demandées. Afficher et commenter le résultat obtenu.

OutMax est égal à 255 (valeur maximale qu'on peut trouver pour un pixel)

InMax est égal à 255 également.

L'équation de la droite est de type Out(In) = OutMax – In (si on se base sur la courbe représentée sur la fiche du TP).

Les instructions nécessaires sont

```
imaGrisInverse = 255-imaGris;
imaInverse     = 255-ima;
```

Pour l'image grise inversée, on obtient un négatif en nuances de gris.

Pour l'image en couleurs inversées, on obtient également un négatif mais en couleurs. (Le jaune devient bleu par exemple)

image couleur

image inverse vidéo

image gris

image inverse vidéo

Table des niveaux de gris

Instructions complétées et/ou écrites dans le programme :

```
%Afficher l'image en niveau de gris avec 32 niveaux de gris
nbGris = 32;
pause

figure(n2)
colormap(gray(nbGris));
map = colormap
image(imaGris/(255/nbGris-1))
```

1. La table de couleurs gray contient toutes les nuances de gris allant de 0 à 1 (en flottants) et le paramètre de la fonction gray correspond au nombre de valeurs dans la table. Plus on aura un grand nombre de valeurs, plus on aura des nuances de gris. 0 correspond au noir et 1 au blanc.

2.

L'équation de la transformation d'image permettant d'obtenir une image en niveaux de gris (selon la courbe de la fiche de TP) est Out(In) = In

Il nous faut donc une bijection entre l'ensemble des couleurs de l'image originale de gris vers l'ensemble des 32 niveaux de gris de la table gray.

Quantification et Indexation des couleurs

1) couleur contient les 32 couleurs prototypes pour notre indexation. L'algorithme des K-means permet de trouver les 32 couleurs les plus représentatives pour toutes les couleurs de l'image d'origine. couleur est une table de trois colonnes (RVB) et 32 lignes.

2)

```
d                  = dist2(data, couleur);
```

On calcule ici la distance entre chaque pixel de couleur de l'image originale avec chaque couleur de la table couleur. On stocke toutes ces distances dans d.

```
[temp,indexCoul] = min(d,[],2);
```

On récupère la distance minimale d'une couleur avec une couleur prototype (donc la couleur la plus proche de celle originale). On stocke les distances minimales dans deux matrices (temps et indexCoul) d'une colonne

```
imaIndexCoul       = reshape(indexCoul, lig, col);
```

On modifie la forme de la matrice indexCoul en lui indiquant le nombre de lignes (le nombre de lignes de la matrice de l'image originale) et le nombre de colonnes (idem). On stocke le résultat dans la matrice imaIndexCoul de taille lig x col.

3) Plus on va réduire le nombre de couleurs représentatives plus l'image originale va perdre de la couleur et de l'information. (On voit d'ailleurs que les nuages ressemblent de moins en moins à des nuages)

La figure 3 représente l'image en index couleur, c'est-à-dire que les pixels d'une même couleur seront représentés par la même couleur prototype une fois la quantification faite. Les couleurs utilisées dans cette figure représentent l'index dans le LUT, elles ne représentent pas la couleur qu'on obtiendra à la fin. Pour cela il nous faut le LUT pour traduire cette image.

Instructions complétées et/ou écrites dans le programme :

```
%On crée la matrice qui correspondra à notre image quantifiée
imaQuantifRGB = ones(lig, col, 3);

%Pour chaque pixel de la nouvelle image, on retrouve la composante de
chaque couleur
%correspondante dans le LUT
for i=1:lig
    for j=1:col
        for k=1   :3
            imaQuantifRGB(i,j,k) = couleurLut(imaIndexCoul(i,j),k);
        end
    end
end

%-------------------------------------------------------%

%Affichage
n5 = figure;
%--Insérer le code pour l'affichage de imaQuantifRGB----%
image(imaQuantifRGB)
colormap('default')
```

4) Cette image sera affichée avec la table de couleur par défaut. Chaque index correspond à trois composantes RGB, il suffit de relier les index de l'image imaIndexCoul avec la table de couleur couleurLut pour retrouver les composantes RGB que l'on entrera dans notre nouvelle image imaQuantifRGB. Il faut donc parcourir les trois dimensions de la matrice correspondante à cette image et pour chaque pixel lui attribuer sa composante RGB que l'on trouve dans la table couleurLut.
   Quand on enregistre les figures 4 et 5, on constate que la figure 4 a une taille de 177 Ko alors que la figure 5 a une taille de 540 Ko. Ce qui est normal puisqu'on a récupéré les trois dimensions RGB dans la figure 5. On constate également que l'image originale avait une taille de 834 Ko. On l'a donc compressé mais en perdant de l'information et de la couleur. Les figures 4 et 5 ont un rendu équivalent, leur seule différence réside dans le fait que l'image de la figure 4 n'a pas 3 dimensions RGB contrairement à l'image de la figure 5.

5) Plus on diminue le nombre de couleurs de l'index, plus on perd de la couleur sur notre image originale mais on augmente sa compression. Ainsi il faut trouver un compromis entre le nombre de couleurs d'indexation et la compression de l'image. Si l'image est trop dégradée c'est que l'on n'a probablement pas mis suffisamment de couleurs d'indexation.