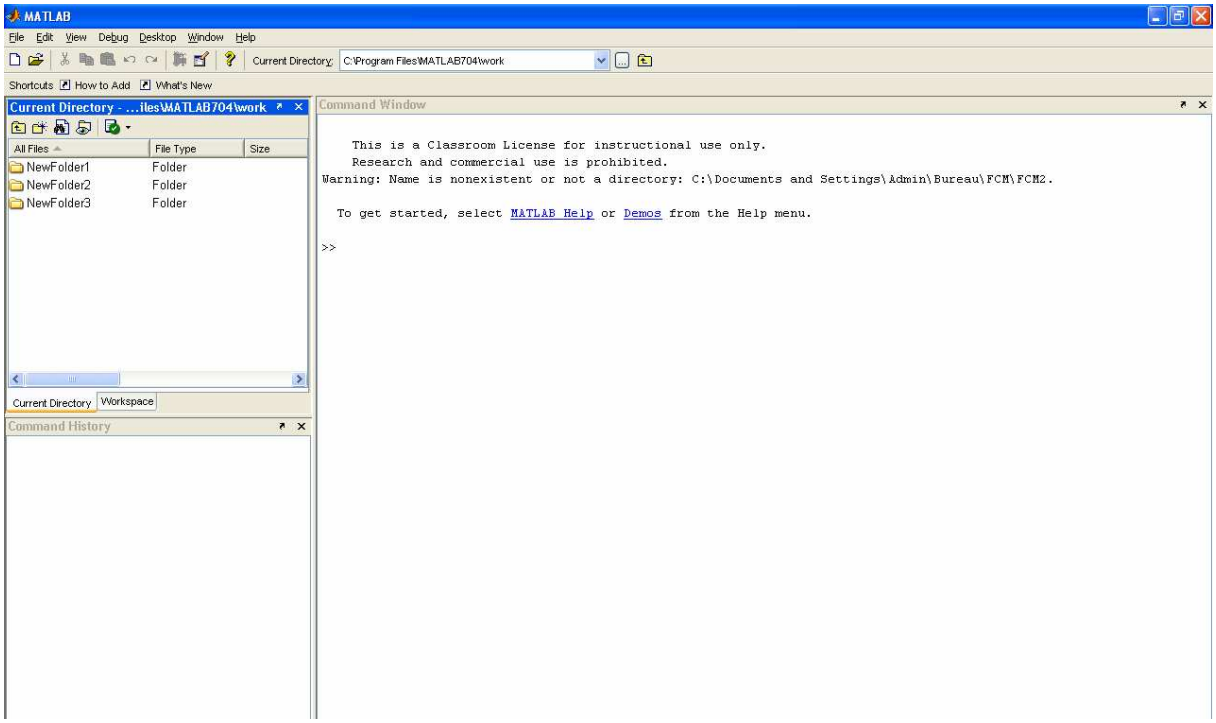
Traitement, Analyse d’Images

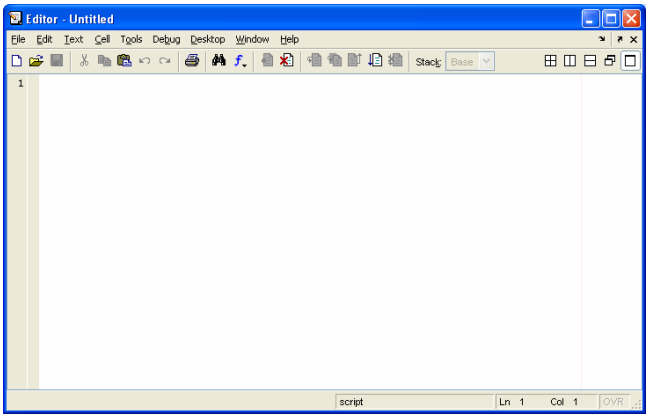
TP

Généralités

Matlab fonctionne de 2 façons soit en ligne de commande soit par l'intermédiaire de scripts ou de fonctions.
L'écran d'accueil est le suivant. On y distingue 3 fenêtres correspondant respectivement (à partir du haut gauche) au répertoire courant, puis en dessous à l'historique des commandes passées à Matlab et enfin à droite au shell Matlab dont le prompt est >>.



Vous pouvez donc effectuer un certain nombre de commandes via ce shell, une ligne de code adaptée à ce que vous souhaitez réaliser.
Pour ne pas afficher les résultats du traitement par Matlab d'une ligne de commande, il conviendra de faire suivre la commande en question d'un point-virgule (';').
Ce type d'approche est restrictif car il convient généralement de tester un ensemble d'instructions adaptées au traitement souhaité. Le script est alors indispensable. Il permet de regrouper dans un fichier à l'extension .m (par exemple essai.m) un ensemble d'instructions. Pour exécuter le script, on utilisera la ligne de commande avec son nom (dans l'exemple précédent essai).
Pour insérer des commentaires dans un script ou une fonction, il faut utiliser le caractère % en début de ligne.
Avec Matlab est fourni un éditeur de script (Menu File puis New MFile ou ctrl+N).



Lorsque vous souhaitez créer une nouvelle fonction *fct* qui prend 2 paramètres et retourne une variable, vous devez créer un nouveau fichier (qui aura nécessairement le même nom que la fonction : *fct.m* dans notre exemple). On écrira alors :

```
function retour=fct(param1, param2)
```

La fonction définie retourne une valeur. Il n'y a pas de return comme dans d'autres langages mais le résultat en retour sera passé dans la variable (ici *retour*).

Exemple : Fonction Addition

```
function r=addition(param1, param2)
     m=param1+param2 ;
     retour=m ;
```

Pour exécuter cette fonction dans l'éditeur, dans un script ou dans une autre fonction, on utilisera son nom suivi des paramètres adaptés.

Exemples d'utilisation de la fonction addition en ligne de commande

Ex.1

```
>> m=3;
>> n=4;
>> addition(m,n)
ans=
7
```

Ex.2

```
>> m=3;
>> n=4;
>>ret=addition(m,n) ;
>>ret
ret=
7
```

Remarque 1 : Si vous utilisez *addition*(m,n) ; vous n'aurez pas d'affichage du tout. La variable *ans* (retour de la fonction) peut cependant être affichée par la commande *ans* dans le prompt.

Remarque 2 : Si vous omettez les ; dans l'implantation de la fonction vous obtiendrez un affichage de toutes les variables.

Remarque 3 : Les variables définies dans une fonction sont locales et les paramètres sont passés par recopie. Les variables de l'environnement ne sont pas visibles dans une fonction.

L'onglet Workspace de la 1ère fenêtre vous permet de visualiser l'ensemble des variables définies précédemment. Si c'est un tableau (une matrice) un double clic sur ce tableau permet de le visualiser.

Matlab et le path

Pour que les fonctions soient utilisables, il faut que les fichiers correspondants soient définis dans un répertoire du path de Matlab.
Pour ajouter un répertoire au path Matlab procédez comme suit :

- Menu *file* puis *Set Path* et ajoutez le répertoire que vous souhaitez.
- Sauvez votre modification en cliquant sur File puis Save Path.

Aide de Matlab :

L'aide s'obtient avec *help* plus le nom de la fonction ou par F1 (Menu Help puis Help Matlab) ou *doc* plus le nom de la fonction pour avoir le browser d'aide.

Base de MatLab : Les matrices

Déclaration d'une matrice

- Déclaration par les valeurs de la matrice

La matrice est définie ligne par ligne. Chaque ligne est séparée par un point virgule.

```
>> m=[ 1 2 3; 4 5 6 ; 7 8 9]
m =
1 2 3
4 5 6
7 8 9
```

- Allocation d'une matrice avec uniquement des valeurs prédéfinies (et pour laquelle les coefficients sont définis ultérieurement)

Cette fonction est ***zeros*** et permet d'allouer une matrice à la taille et la dimension désirées (ici 3x4) contenant uniquement des zéros. De même, les fonctions ***ones*** et ***eye*** ont les mêmes propriétés et construisent respectivement une matrice avec des 1 uniquement et la matrice identité de la taille voulue (première diagonale contenant des 1, le reste des coefficients à 0).

```
>> m=zeros(3,4)
m =
0 0 0 0
0 0 0 0
0 0 0 0
```

- Déclaration d'un vecteur

Le nombre minimum de paramètres est de 2. Ainsi, pour construire des vecteurs (matrice à une dimension) il faut mettre un paramètre à 1 selon que l'on souhaite un vecteur colonne ou un vecteur ligne).

```
>> m=zeros(1,3)
m =
0 0 0
```

- Suites de nombre

Il est aussi possible de définir des suites de nombre selon la syntaxe suivante:

m= nb_debut : nb_fin

ou m= nb_debut : pas_d_incrementation : nb_fin

Exemple :

```
>> m=1:10
m =
1 2 3 4 5 6 7 8 9 10
>> m=1:0.5:5
m =
1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
4.5000 5.0000
```

Taille d'une matrice : size()

La taille d'une matrice est donnée par la fonction *size(m)* qui renvoie une matrice des tailles (nombre de lignes et nombre de colonnes). Pour récupérer le nombre de lignes, on spécifiera *size(m,1)*, et pour le nombre de colonne *size(m,2)*.

Exemple 1 :

```
>>
m=zeros(6,3);
>> size(m,2)
ans =
3
```

Exemple 2 :

```
>> m=zeros(6,3);
>> [nl,nc]=size(m);
>> nl
nl =
6
>> nc
nc =
3
```

Éléments de la matrice

ATTENTION : Tous les index de matrice de Matlab commencent à 1!
La syntaxe pour extraire un élément en ligne i et colonne j d'une matrice utilise des parenthèses : m(i,j)

Exemple :

```
>> m=[ 1 2 3; 4 5 6 ; 7 8 9]
m =
1 2 3
4 5 6
7 8 9
>> m(1,2)
ans =
2
>> m(2,3)
ans =
6
```

Pour extraire simplement une ligne ou une colonne de la matrice, on utilisera la notation ‘:’ en lieu et place de la colonne ou de la ligne que l’on souhaite extraire et on indiquera le numéro de cette ligne ou de cette colonne:
Exemples :

```
>> m(:,1)
ans =
1
4
7
```

```
>> m(2,:)
ans =
4 5 6
```

Pour extraire une sous-matrice, on utilisera la syntaxe suivante : *m(indice_de_debut : indice_de_fin, indice_de_debut : indice_de_fin)* respectivement pour désigner les lignes extraites et les colonnes dans ces lignes.

Exemple :

```
>> m(1:2,1:3)
ans =
1 2 3
4 5 6
```

Dans cet exemple, on a extrait de la matrice m les lignes de 1 à 2 (1:2) et les colonnes de 1 à 3 (1:3).

Opérations entre matrices

m1*m2	Produit matriciel de m1 et m2. Il faut size(m1,1)=size(m2,2).
m1/m2	Division matricielle de m1 et m2. Il faut size(m1,1)=size(m2,2).
m1+m2	Somme terme à terme de m1 et m2 (doivent avoir la même taille).
m1-m2	Soustraction terme à terme de m1 et m2 (doivent avoir la même taille).
m1.*m2	Multiplication terme à terme de m1 et m2 (doivent avoir la même taille).
m1./m2	Division terme à terme de m1 et m2 (doivent avoir la même taille).
-m1	Opposé de la matrice m1.
m1'	Transposée de m1.

ATTENTION : Ne pas confondre le produit terme à terme avec le produit matriciel.

Opération sur une matrice

Pour toutes les fonctions usuelles, celles-ci s'effectuent terme à terme sur la matrice:

Exemple :

```
m=[ pi 0 ; -pi pi/2];
>> cos(m)
ans =
-1.0000 1.0000
-1.0000 0.0000
```

En particulier, toutes les opérations avec un scalaire (un nombre) se font terme à terme (addition, multiplication, division de toute une matrice par une valeur)

Exemple:

```
>> m*2
ans =
6.2832 0
-6.2832 3.1416
```

Instructions de base pour la programmation Matlab

Instructions classiques

- Boucle for

```
for(i=indice_debut : indice_fin)
      % instructions...
end
```

- If

```
if( condition )
      % instructions...
[else
      % instructions facultatives…]
end
```

La comparaison entre deux entiers s'effectue avec `==` et la différence avec `~=`

Utilisation d'images

- Chargement d'une image : imread

La fonction imread prend en paramètre le nom d'un fichier, localisé par son chemin à partir du répertoire de travail courant. Les formats supportés sont : BMP, JPEG, GIF, PNG, TIFF, ...

Dans le cas d'une image en niveaux de gris, la matrice est une matrice d'entiers de dimension 2 (entre 0 et 255). Dans le cas d'une image couleur, le format de base de Matlab est une matrice de dimension 3 (cf. doc Matlab pour plus d'info).

Exemple : Chargement d'une image noir et blanc.

```
>>I=imread('lenna.gif');
>> size(I)
ans =
256 256
```

- Enregistrement d'une image : imwrite

La fonction imwrite prend en paramètres la matrice à enregistrer comme image, puis le nom du fichier image (avec un format d'enregistrement choisi par l'extension fournie). On peut également spécifier le format choisi.

Exemple : Enregistrement de l'image précédente au format JPEG.

```
>> imwrite(I,'lennaNG.jpg');
```

- Visualisation d'une image : Image

La fonction *image(I)* de Matlab permet de visualiser des images couleur : l'affichage n'est pas correct avec des images en niveaux de gris si on ne passe pas par une autre fonction modifiant la palette de couleur. On exécutera donc ensuite *colormap gray*. Vérifier dans l'aide de Matlab quel doit être le type de paramètre fourni à *image* et faites les conversions nécessaires.

Implantation d'algorithmes de Traitement d'Images

S. Cormier – Master 1 Informatique 2010-2011

Remarque préliminaire très importante : L'ordre de réalisation n'est pas forcément à respecter. Il convient de faire tout d'abord la binarisation puis le calcul du gradient et du Laplacien puis de l'histogramme et du recadrage.

Compte rendu à réaliser et à remettre : en dehors de l'implantation, il faut tester ces fonctions sur les images *house, femme, fille, couloir et bureau* etc. et faire varier les paramètres de seuil pour la binarisation et dégager des conclusions adaptées.

1. Utilisation de Matlab

Vous devrez utiliser Matlab pour visualiser et manipuler les images fournies.
Matlab permet très facilement de manipuler des matrices. Associé avec une toolbox comme Image Processing ce peut être un outil de test intéressant pour définir de nouveaux algorithmes ou utiliser des algorithmes « classiques » du T.I.

Rappels succincts Matlab :

- Pour créer une matrice de n lignes et p colonnes initialisée avec des valeurs a_{ij} (i=1.. n j=1..p) connues, on procédera de la façon suivante :

M= [a11 a12 a13 … a1p ; a21 a22 …. a2p ; …. ; an1 an2 … anp] ;

N'oubliez pas le point virgule à la fin...
Vous pouvez stocker votre programme Matlab dans un fichier .m

- *Help* nom_de_la_comande vous donne l'aide sur la fonction. Si vous remplacez *help* par *doc* vous avec la doc en HTML.
- *Close all* et *clear* sont 2 instructions que vous pouvez inclure dans chaque programme pour fermer toutes les figures et effacer toutes les variables
- Imread permet de charger une image à un format particulier (confère l'aide de Matlab)
  Ex. *I = imread('toto.jpg') ;*
  Cette instruction permet de stocker dans la matrice I les valeurs des pixels de l'image. Selon le format vous obtiendrez une valeur adaptée pour votre matrice I.

- Création de boucles : *For*
  For i =indice_debut : indice_fin

- Test: if
  if expression1
  …
  else
  …
  End

- Opérateurs: *&* | ~

2. Application des LUT : Binarisation

Rappel : Une LUT est une fonction de transformation de l'ensemble des niveaux de gris d'une image, indépendante de la position des pixels. Elle est définie par :

F : ndg → ndg
ndg$_i$ → F(ndg$_i$)

Un niveau de gris appartient à l'ensemble {0, 1, ... , 255} pour une image en 8 bits et plus généralement à l'ensemble {0, 1, ... , 2^n-1} pour n bits.
Pour plus de détails sur les LUT (cf. cours).

La binarisation

Pour les détails de ce type d'approche cf. cours.
Nous sommes en 256 niveaux de gris. Nous allons implanter 1 fonction de binarisation dont le but est de pouvoir binariser de façon simple.
Il s'agit d'abord de faire saisir à l'utilisateur 2 valeurs de seuils, seuil_bas et seuil_haut.
Ensuite, il faut créer une LUT (cf. cours) qui va être constituée de la façon suivante :

- Créez un tableau à 256 (2^8) entrées appelé *lut_bin*
- Pour toutes les valeurs de niveaux de gris comprises entre seuil_bas et seuil_haut, on affecte la valeur 0 puis on affecte 255 dans le cas contraire.

Vous devez donc obtenir quelque chose de la forme :

- *lut_bin*[i] = 0 si i ≤ seuil_bas ou si i ≥ seuil_haut
- *lut_bin*[i] = 255 (ou 1) dans le cas contraire

A partir de là, la procédure consiste à charger une image puis à modifier les données qui la compose.
Testez la binarisation pour les images femme et couloir avec des seuils adaptés.

Histogramme

Le calcul de l'histogramme des niveaux de gris permet de déterminer la fréquence d'apparition de chaque valeur de niveau de gris *ndg$_i$* présent dans l'image.
Pour déterminer cet histogramme, il convient donc de :

- créer un tableau avec 256 entrées *histo[]*, initialisé à zéro
- les valeurs stockées sont des valeurs comprises entre 0 (il n'y a aucun pixel d'un tel niveau de gris dans l'image) et dimX*dimY au maximum (cas d'une image complètement noire ou complètement blanche par exemple !)
- En parcourant le tableau Image, si la valeur du niveau de gris vaut *ndg$_i$* alors incrémenter *histo[ndg$_i$]*.

Comparez avec la fonction Matlab. La Création d'un histogramme sous Matlab se fait aisément avec hist.
Vous pouvez créer votre histogramme directement avec la fonction imhist selon la syntaxe suivante :
[n,x] = imhist(Nom_du_tableau_stockant_l_image, 256)
n contiendra le nombre de pixels dans chacun des niveaux et x une numérotation des niveaux

Visualisez le résultat pour l'image femme.

Le traçage de l'histogramme est également possible : *figure, imhist(i,256))* par exemple.

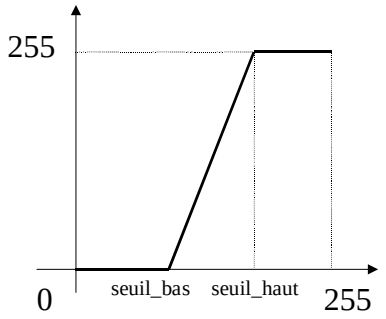
Visualisez et sauvegardez VOTRE HISTOGRAMME.

3. Recadrage dynamique

Cette transformation consiste à redistribuer sur l'intervalle complet [0, 255] les valeurs de niveaux de gris des pixels de l'image traitée.

Il s'agit de définir 2 seuils comme précédemment seuil_bas et seuil_haut.
La transformation à appliquer aux niveaux de gris est alors définie par :

- $F(ndg_i)=0$ si ndg_i appartient à [0, seuil_bas]
  ou =255 si ndg_i appartient à [seuil_haut , 2^n-1]
- $F(ndg_i)=\alpha * ndg_i + \beta$ sinon



Transformation par une fonction linéaire

Vous devez donc calculer les coefficients α et β à partir de cette définition.
Une fois ces coefficients déterminés (ils le sont de façon formelle d'après la définition de la fonction), il faut déterminer les seuil_bas et seuil_haut.
Pour cela, il est nécessaire de déterminer l'histogramme cumulé de l'image.
Cet histogramme sera donc :

$$histo\_cumule[i] = \sum_{k=0}^{i} histo[k]$$

Ensuite, seuil_bas et seuil_haut sont tels que :
histo_cumule [seuil_bas] ≤ 2,5% du nombre total de pixels
et histo_cumule[2^n-1]-histo_cumule[seuil_haut]≤ 2,5% du nombre total de pixels

Ce nombre total de pixels est bien sûr égal à N=dimX*dimY.

L'implantation est donc réalisée en deux temps :

1. Détermination de l'histogramme cumulé
2. Déterminer l'indice *a* tel que histo_cumule[*a*] >2.5%*N (seuil_bas vaut alors *a*-1)
3. Déterminer l'indice *b* tel que histo_cumule[2^n-1] - histo_cumule[*b*] > 2.5%*n (seuil_haut vaut alors *b*-1)

Le calcul est ensuite effectué en utilisant la transformation définie précédemment.
Pour toute valeur entre seuil_bas et seuil_haut on calcule F et on affecte cette valeur à une LUT particulière, *lut_t* (initialisée correctement pour les valeurs n'appartenant pas à l'intervalle): *lut_t*[k]=T(k).
On applique ensuite la *lut_t* à l'image comme précédemment et on la sauvegarde.

4. Détection de contours

Cette détection de contours sera effectuée par deux algorithmes, Gradient et Laplacien et dans le cadre du filtrage linéaire, caractérisé par un noyau de convolution (cf. cours).
Pour implanter un opérateur, il faut considérer son noyau de convolution (fenêtre 3*3) (cf. cours).

$$t(i, j) = \sum_{i=0}^{9} h(i, j).I(i, j)$$

Traditionnellement, on écrit les filtres sous la forme de coefficients entiers. Pour le gradient et le Laplacien, les noyaux (cf. cours) ont pour nom respectif g_x et g_y (en X et en Y) et Laplacien

Pour pouvoir calculer la valeur de la convolution, il est utile de disposer d'une ligne et une colonne supplémentaire contenant des 0 ou autre selon le prolongement. Cette convolution prend en paramètre le noyau du filtre. Réalisez cette opération avec Matlab.
Comparez avec la fonction Matlab (cf. aide de Matlab).

Gradient : Appliquer cet opérateur avec les 2 noyaux fournis.

Amplitude du gradient : L'amplitude du gradient en un point p(x,y) vaut $[G_x^2(x,y)+G_y^2(x,y)]^{1/2}$
A partir des images du gradient en X et en Y calculé avec les noyaux respectifs, écrire une fonction amplitude_g qui crée une image qui contient l'amplitude du gradient.

Laplacien : L'application est immédiate, il suffit de calculer la convolution avec un autre opérateur !