Traitement de l'information et vision artificielle **Guillaume Obozinski**
Ecole des Ponts ParisTech, 2ème année guillaume.obozinski@imagine.enpc.fr

TP 3 ET 4 : FILTRE DE WIENER ET ACP

La séance de TP se fait sous environnement Windows, sauf si vous avez une nette préférence pour Linux.

Pour commencer la séance

1. Lancer un navigateur, par exemple Mozilla, et aller sur la page web suivante :
   `http://imagine.enpc.fr/~marletr/enseignement/mpi/index.html`
2. Télécharger l'archive. Un fois décompressée on obtient un dossier `TIVA_TP3`.
3. Lancer ensuite `Matlab` et modifier le répertoire de travail en choisissant le répertoire `Bureau/TIVA_TP3` que vous avez créé.
4. Pour les différentes questions, vous pouvez utiliser un «copier-coller» à partir de ce document. Il est fortement recommandé de saisir toutes les commandes dans la fenêtre de l'éditeur que vous avez ouverte. Pour exécuter les commandes saisies, il suffit de les sélectionner avec la souris et d'appuyer sur la touche `F9`.
5. Pour inclure des commentaires dans le programme, ce qui est fortement recommandé, vous devez utiliser le caractère `%`. Tout ce qui suit ce caractère sera négligé lors de l'exécution.

1. Commandes pour l'affichage des images et autre

1. La commande `imdisp` permet d'afficher plusieurs format d'images, dont un certain nombre d'images en niveau de gris, qui ne s'affichent pas correctement avec `image` ou `imagesc`. Pour pouvoir vous en servir il faut rajouter la commande au "path" de Matlab car elle est dans un sous-dossier qui s'appelle lui-même `imdisp`. Il suffit d'executer la commande `addpath('imdisp');`
2. Pour les images `.png`, elles ont souvent leur propre colormap. Il faut donc les importer le colormap avec l'image et demander à Matlab de l'utiliser, ce qui s'obtient avec
   ```
   [I,map]=imread('mon_image.png');
   image(I);
   colormap(map);
   ```
3. La commande `image` permet de visualiser une image RGB ou NB dont les couleurs sont soit codés sur 8 bits par un entier au format `uint8`, soit codés par des doubles dans l'intervalle $[0, 1]$. `imagesc` permet d'afficher des images en niveaux de gris dont les valeurs ne satisfont pas ces restrictions.

4. Pour "imprimer" une image ou un graphe au format `.jpg`, `.png`, `.eps` ou autre on utilisera la commande `print`. Elle imprime le contenu de la figure courante. Par exemple, `print('-depsc','mon_image.eps')` pour une image EPS couleur. Lorsqu'il s'agit d'images PNG et JPEG sont plus appropriées. On on utilisera plutôt `print('-djpg','mon_image.jpg')`
5. Si vous voulez sauver les variables que vous avez en mémoire vous pouvez à tout moment taper `save ma_session` dans la ligne de commande et les variables de votre session seront sauvés dans un fichier `ma_session.mat` dans le répertoire courant.

3. Plaque floue...

Le but ici est d'utiliser le filtre de Wiener. Une voiture a été prise en photo [1] roulant a bonne allure devant l'Université d'Oxford. La plaque est tellement floue qu'elle est illisible... Mais le flou étant dû à la translation rectiligne de la voiture, il y a bon espoir de pouvoir faire de la déconvolution... Il est recommandé de garder les commandes que vous exécuterez au fur et à mesure dans un script afin de pouvoir assembler facilement les différent morceaux vers la fin de cette partie. Allez dans le dossier `Wiener` avec Matlab pour cette partie.

1. Chargez l'image `car.tiff` et visualisez la avec la commande `imdisp`. La voiture est floue du fait de son déplacement.
2. Si la voiture était prise exactement de profil, le flou correspondrait exactement à la convolution de l'image nette avec un segment horizontal. Comme la plaque n'est pas très grande, on va considérer que son image floue est obtenue par convolution avec un segment qui n'est pas horizontal, et puisque la convolution avec un segment diagonal est compliqué, on commence par faire une rotation de l'image et extraire une région d'intérêt autour de la plaque. Pour simplifier les choses, cette image est déjà extraite. Elle est sauvée dans `plaque.mat`. Charger l'image avec `load('plaque');`
   La variable plaque devrait être définie dans l'environnement de travail.
3. On construit d'abord le noyau de convolution h : un segment horizontal de longueur L qui permet par convolution de faire la moyenne sur L pixels. On pourra estimer L à l'oeil en zoomant sur l'imagette et essayer plusieurs valeurs différentes. On construit une image qui a les mêmes dimensions que `I`.
   ```
   [N1,N2]=size(I);
   seg=zeros(size(I));
   seg(1,mod(round(L/2:L/2),N2)=1/L;
   ```
   Pourquoi a-t-on construit `seg` comme cela plutôt que de le centrer sur l'image ? Justifiez avec la formule de la convolution. Notez qu'on peut obtenir une version centrée avec `fftshift`.
4. Calculez la TFD H de `seg` et affichez son module avec `imagesc` en recentrant la TFD de façon que l'origine soit au centre de l'image grâce à `fftshift`. Avec la commande `plot`, faites une représentation du profil longitudinal du module de H.

1. Cette photo est gracieusement mise à notre disposition par Andrew Zisserman, professeur en vision par ordinateur à l'Université d'Oxford.

5. Calculez la TFD de l'image de la plaque, après l'avoir converti au format double.
6. Calculez une matrice W qui correspond au filtre de Wiener. On considérera que $K(u, v)$ est une constante qu'il faudra trouver par essai et erreur, disons $K = 0.001$ pour commencer. La commande `conj` pourra servir.
7. Faites une fonction `deblurr` qui prend en entrée K, L et l'imagette, qui calcule le filtre de Wiener pour ces paramètres, qui l'applique à l'imagette dans le domaine de Fourier et qui retourne la transformée de Fourier inverse. Affichez le résultat avec `imagesc`. On pourra afficher côte à côte l'image originale et l'image défloutée avec la commande `subplot` de la façon suivante

```
figure(2)
subplot(1,2,1);
imagesc(I)
subplot(1,2,2);
imagesc(DeblurredImage);
```

8. Changez L et K, pour obtenir le meilleur résultat possible. Est-ce qu'on arrive à lire la plaque ?
9. Représentez avec la fonction `plot` le profil longitudinal du module de W. Que se passe-t-il quand K est trop faible ? Trop grand ?
10. Calculez la transformée de Fourier inverse de W. Appelons-la w. Représentez avec la fonction `plot` la section longitudinale de w (Il pourra être judicieux de visualiser d'abord w avec `imagesc`). Interprétez l'application du filtre de Wiener en terme de convolution avec w. Que se passe-t-il quand K est trop faible ? Trop grand ?

4. Visages et ACP Le but de cette partie est de comprendre ce que l'ACP permet d'obtenir lorsqu'on l'applique sur une base de données de visages alignés, soit du même individu, soit de plusieurs individus avec des conditions d'éclairages changeantes. On utilisera la base d'image de visage de Yale qui a été utilisé dans le cadre de la recherche sur la reconnaissance automatique de visage.
`http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html`
(Ce n'est pas la peine de la télécharger à partir de cette page web).

Nous ne ferons malheureusement pas de reconnaissance automatique de visages aujourd'hui, même si des méthodes utilisant des idées de sous-espaces (mais pas juste de l'ACP) existent bien. Si vous voulez sauver les variables que vous avez en mémoire vous pouvez taper `save ma_session` dans la ligne de commande et les variables de votre session seront sauvés dans un fichier `ma_session.mat` dans le répertoire courant. Vous pouvez faire place nette en executant la commande
`clear all` pour effacer de la mémoire de Matlab toutes les variables de la partie précédentes.

Allez dans le dossier `ACP` avec Matlab pour cette partie.

1. Chargez le fichier `visages.mat` avec la fonction `load`. Il contient deux variables `visages_train` et `visages_test`. Ces deux objets sont des *cell array*. Tapez `whos visages_train` dans la ligne de commande. Chaque *cell array* est composé de 10 cellules et chaque cellule contient une matrice, contenant des images du visage d'une personne. La commande
   ```
   I=visages_train{7};
   ```
   vous permet de récupérer la matrice contenant les images du sujet numéro 7. Attention, pour obtenir le contenu d'une cellule il faut mettre des accolades et pas des parenthèses. Tapez `whos I`. On voit que I est une matrice avec 54 colonnes et beaucoup de lignes. Chaque ligne est une image de 192×168 pixels. On peut reconstruire la matrice correspondant à une image particulière d'indice k avec la commande :
   ```
   reshape(I(:,k),[192 168]);
   ```
   Pour résumer, `visages_train` contient pour 10 sujets différents 54 images de taille 192×168 pixels. Les images de chaque sujet sont dans une des cellules de `visages_train`. De la même façon, `visages_test` contient pour les 10 même sujets 10 images de taille 192×168 pixels que l'on garde en réserve. On se servira surtout de `visages_train`.

2. La fonction `view_faces` fournie prend en entrée une matrice correspondant aux images d'une personne et permet de les visualiser. Elle s'arrête après chaque image (avec la commande `pause();`) et il suffit d'appuyer sur la barre d'espace pour aller d'une image à la suivante. Utilisez la pour visualiser les visages d'une ou plusieurs personnes.

3. La fonction classique de Matlab pour calculer la SVD est la fonction `svd`. Tapez `help svd` dans la ligne de commande pour voir quels sont les arguments d'entrée et de sortie de la fonction. En particulier, notez que l'on peut obtenir la décomposition fine $X = USV^\top$ en tapant
   ```
   [U,S,V]=svd(X);
   ```
   Quand la matrice X est de grande taille, calculer la SVD complète (même la version compacte) est souvent trop couteux. On peut ne calculer que les K premier vecteurs singuliers à droite et à gauche avec la commande `svds`. Comme nous manipulons des images, la matrice X a autant de colonnes que de pixels. Elle est donc très large et les implémentations de SVD de Matlab traînent la patte. On utilisera donc la fonction `mysvd` fournie, qui commence par calculer une décomposition QR de la matrice. Le code de `mysvd` est le suivant
   ```
   function [U,S,V]=mysvd(X,K)
   %
   % Cette fonction calcule les K premières composantes
   % de la SVD pour
   % des matrices avec beaucoup de colonnes
   %
   ```

```
[Q,R]=qr(X',0); % Decomposition QR
[U,S,V0]=svds(R',K); % calcul des K premieres composantes de la SVD de R'
V=Q*V0;
```

Expliquez pourquoi la décomposition QR aide et pourquoi le calcul qui est fait dans cette fonction renvoie bien la SVD qu'on veut. (On pourra taper `help qr` dans la ligne de commande si nécessaire.).

4. On choisit l'un des sujets et on calcule les 20 premières composantes de sa SVD, avec

```
I=visage_train{7};
X=double(I');
 muX=mean(X);
X=bsxfun(@minus,X,mean(X));
K=10;
[U,S,V]=mysvd(X,K);
```

Représentez la diagonale de S avec la fonction plot

```
plot(diag(S),'.-');
```

Qu'en concluez-vous ? Combien de directions principale sont nécessaires pour expliquer la plus grande partie de la variance des données ?

5. Dans la suite on utilisera 10 composantes. On commence par visualiser les axes principaux qu'on a trouvé avec

```
for k=1:10,
figure(3)
imagesc(reshape(S(k,k)*V(:,k)'+muX,[192,168]));
axis off
axis image
colormap gray
pause();
end
```

Peut-on donner une interprétation aux directions principales ? Si vous n'avez pas trop d'idées, peut-être pourrez vous y répondre avec la question 9...

6. Le but de cette question est de voir si on arrive à reconstruire une image visuellement correcte à partir de ses composantes principales. On se servira du code ci-dessous dans lequel il faut calculer `Proj_Ik` qui est la projection **affine** de l'image sur les axes principaux dans l'espace de départ. Attention au fait qu'on fait une projection **affine** et non pas une projection linéaire, puisque le sous-espace principal est un sous-espace affine passant par `muX`...

```
 for k=1:size(I,2)
figure(3)
subplot(1,2,1)
imdisp(reshape(I(:,k),[192,168]));
subplot(1,2,2)
```

```
Proj_Ik=zeros(192*168,1);
% A vous de jouer ici et de calculer Proj_Ik !
imdisp(reshape(Proj_Ik,[192,168]));
pause();
end
```

Etes-vous satisfaits du résultat ?

7. **Représentation dans le plan des directions principales**
   On souhaite finalement voir les composantes principales tant espérées. Calculez la matrice `Y` des composantes principales qui permettra d'afficher les deux premières composantes avec
   ```
    figure(4)
   plot(Y(:,1),Y(:,2),'o');
   ```

8. On fait une analyse en composante principale de l'ensemble des images de visages. On peut obtenir toutes images dans une matrice avec la commande
   `Iall=[visages_train{:}];`
   Après avoir calculé la matrice de design et en n'oubliant pas de centrer les données, calculez les 3 directions principales de ce nuage de points. Si vous souhaitez vous débarrasser des grosses matrices en mémoire, faites
   ```
   clear Iall;
   clear X;
   ```
   une fois la SVD calculée. Visualisez les composantes principales en recalculant Y et avec
   ```
    figure(4)
   plot(Y(:,1),Y(:,2),'o');
   ```
   Que remarque-t-on sur la structure du nuage de points ainsi visualisé ?
   Faites la même chose en 3D avec les commandes
   ```
    figure(5)
   plot3(Y(:,1),Y(:,2),Y(:,3),'o');
   ```
   En appuyant sur le bouton "3D Rotate" du panneau de commande de la figure vous pouvez faire tourner le nuage de points.

9. **Question bonus :** Pour essayer de mieux comprendre la structure du nuage de points en donut écrasé et ce que les axes principaux représentent, on se propose de parcourir le nuage de point dans le sens des aiguilles d'une montre autour du centre du donut (pas le centre du nuage de points) et d'afficher simultanément le point courant dans le nuage de points et l'image correspondante. C'est ce que fait le code suivant si on lui donne un vecteur `idx` qui contient les indices des images dans l'ordre qu'on veut.
   ```
   for k=1:length(idx),
   k_idx=idx(k);
   ```

```
figure(6),
subplot(1,2,1)
plot(Y(:,1),Y(:,2),'o');
hold on;
plot(Y(k_idx,1),Y(k_idx,2),'r+');
hold off;
subplot(1,2,2)
imagesc(reshape(Iall(:,k_idx),[192,168]));
colormap gray
axis off
axis image
pause();
end
```

Pour construire le vecteur `idx`, on pourra par exemple projeter les données sur un cercle bien choisi et calculer l'angle que fait la projection avec l'axe des abscisses. Pour calculer l'angle on pourra utiliser la fonction `cart2pol` qui permet de passer en coordonnées polaires, et la fonction `sort` qui permet de trier un vecteur par ordre croissant ou décroissant et renvoie la permutation correspondante. Utilisez `help sort` dans la ligne de commande pour savoir comment utiliser la fonction.

Le compte-rendu sera à envoyer au format `pdf` à `guillaume.obozinski@imagine.enpc.fr` avant le cours du 2 décembre. Il est plus que vivement encouragé d'essayer de finir le TP pour le cours du 25 novembre ou nous aurons une courte session TP et où vous pourrez poser les questions concernant les difficultés rencontrées.