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

TP 2 : Transformée de Fourier et aliasing

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_TP2`.
3. Lancer ensuite `Matlab` et modifier le répertoire de travail en choisissant le répertoire `Bureau/TIVA_TP2` 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. Rappel des commandes de base de Matlab.

Exécuter ligne par ligne les commandes suivantes et chercher à comprendre ce qu'elles produisent.

1. Créer une variable ou un tableau :

```
% this is a comment
a = 1; a = 2+1i % real and complex numbers
b = [1 2 3 4] % row vector
c = [1; 2; 3; 4] % column vector
d = 1:2:7 % here one has d=[1 3 5 7]
A = eye(4) % identity matrix
B = ones(4) % matrix with all the entries =1
C = zeros(4) % matrix with all the entries =0
D = rand(4) % random matrix with Unif[0,1] entries
E = reshape(b,2,2) % transforms the 4-vector b into a 2 x 2 matrix D
c = b' % transpose
```

2. Modifications des vecteurs et des matrices :

```
A(2,2) = B(1,1) + b(1) % to access an entry in a vector or matrix
b(1:3) = 0 % to access a set of indices in a matrix
b(end-2:end) = 1 % to access the last entries
b = b(end:-1:1) % to reverse a vector
b = sort(b) % to sort values
b = sort(b,'descend') % to sort values in descending order
b = b .* (b>2) % to set to zeros (threshold) the values below 2
b(3) = [] % to suppress the 3rd entry of a vector
B = [b; b] % to create a matrix of size 2 x 4
c = B(:,2) % to access 2nd column
```

3. Instructions avancées :

```
a = cos(b); a = sqrt(b) % usual function
help perform_wavelet_transform; % print the help
a = abs(b); a = angle(b); % norm and argument of a complex
a = real(b); a = imag(b); % real and imaginary part of a complex
disp('Hello'); % display a text
disp( sprintf('Valeur de x=%.2f', x) ); % print a values with 2 digits
A(A==Inf) = 3; % replace Inf values by 3
A(:); % flatten a matrix into a column vector
max(A(:)); % max of a matrix
M = M .* (abs(M)>T); % threshold to 0 values below T.
```

4. Affichage des graphiques :

```
plot( 1:10, (1:10).^2 ); % display a 1D function
title('Mon titre'); % title
xlabel('variable x'); ylabel('variable y'); % axis
subplot(2, 2, 1); % divide the screen in 2x2 and select 1st quadrant
```

5. Programmation :

```
for i=1:4 % repeat the loop for i=1, i=2, i=3 et i=4
   disp(i); % make here something
end
i = 4;
while i>0 % while syntax
   disp(i); % do smth
   i = i-1;
end
```

2. Charger et visualiser des signaux et des images.

1. Trouvez une image sur internet et sauvez-la dans votre répertoire courant sous le nom `test_image.xxx` (ici, `xxx` est l'extension de l'image : `jpg, gif, png, tiff,` ...).

```
I=imread('test_image.xxx');
figure('color','cyan')
image(I)
axis off
axis image
```

2. Vous pouvez aussi visualiser les composants R,G et B de l'image :

```
figure('color','cyan')
J = I;
J(:,:,2:3) = 0;
image(J)
axis off
axis image
```

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.

3. Transformée de Fourier d'une image

1. Chargez l'image `lena.png`
2. Calculez et affichez les différentes parties de la FFT de cette image : sa partie réelle, sa partie imaginaire, son module, sa phase. On pourra utiliser les commandes `fft2, ifft2, ifftshift, ifftshift, imagesc, real, imag, abs`. Pour fixer l'affichage en niveaux de gris on utilisera `colormap('gray')` avec `imagesc`. Il sera peut-être nécessaire de prendre le logarithme d'une quantité pour que l'image obtenue soit lisible.
3. Recalculer l'image à partir de sa transformée de Fourier. Que se passe-t-il si on ne prend que le module de la TFD pour reconstruire l'image ? ... que la phase ?

4. Aliasing

1. Chargez l'image `brickwall2.jpg` et affichez la.
2. Redimensionnez l'image en changeant sa taille à l'écran et son *aspect ratio*. Qu'observe-t-on ?
3. Ecrivez une fonction qui sous-échantillonne l'image en prenant 1 pixel sur T. Visualisez l'image obtenue pour différentes valeurs de T de 2 à 10. La fonction `pause` pourra être utile.

4. On se propose maintenant d'étudier l'aliasing sur une image synthétique. Le script `alias_circle.m` calcule la fonction $(x, y) \mapsto \sin(200(x^2 + y^2))$ et permet de la visualiser comme un image. Redimensionnez l'image à la main pour observer les artefacts. Le script permet de sous-échantillonner l'image par des facteurs de 6 à 120 par pas de 6, (ou par des facteurs qui sont des diviseurs de la taille de l'image en éditant le script). (Le script se met en pause après l'affichage de chaque image et il suffit d'appuyer sur la barre d'espacement pour passer à l'image suivante.)

5. **Zoom en Fourier**. Pour les images sous-échantillonnées de la question précédente, nous allons essayer de reconstruire l'image d'origine grâce au théorème d'échantillonnage : ce sera un zoom en Fourier ! Pour un sous-échantillonnage sur un grille de pas T_1 et T_2, la représentation de l'image sous-échantillonnée sur la grille de l'image initiale est la multiplication de l'image initiale par un peigne de Dirac bidimensionnel de pas T_1 et T_2 multipliée par un facteur $T_1\,T_2$, soit pour une image f de dimensions initiales $N_1 \times N_2$,

$$f_d[n_1, n_2] = T_1 T_2\, f[n_1, n_2] \left[\sum_{p_1=1}^{\frac{N_1}{T_1}-1} \sum_{p_2=1}^{\frac{N_2}{T_2}-1} \delta[n_1 - p_1 T_1]\, \delta[n_2 - p_2 T_2] \right].$$

   Ecrire une fonction qui recalcule cette image.

6. Le théorème d'échantillonnage suggère d'appliquer un filtre passe-bas qui supprime toutes les fréquences supérieures à la fréquence de Nyquist, c'est-à-dire supérieures à la fréquence d'échantillonnage divisée par 2. En une dimension un, dans la TFD, si on échantillonne avec un pas T qui est un diviseur de la longueur N du signal initial f , la fréquence d'échantillonnage $1/T$ correspond au coefficient $\widehat{f}[N/T]$. En déduire la forme du filtre passe bas en Fourier à appliquer à $\widehat{f_d}$. Attention au fait que la FFT de Matlab renvoie le vecteur $(\widehat{f}[0], \widehat{f}[1], \ldots, \widehat{f}[N])$ dont le premier coefficient est le coefficient de fréquence 0. Dans le cas d'une image, les coefficients correspondants aux ondes planes selon les axes sont dans la première colonne et première ligne de la TFD. Implémentez une fonction `fourier_interp.m` qui tente de reconstruire l'image initiale à partir de l'image sous-échantillonnée et appliquez la au images sous-échantillonnées des ondes concentriques. Est-ce que l'on évite l'aliasing ? Imprimer une image qui illustre ceci ainsi que l'image sous-échantillonnée correspondante.

7. **Anti-aliasing**. On explore maintenant le principe du filtre anti-aliasing. Modifiez le script `alias_circle.m` pour représenter la fonction $(x, y) \mapsto \sin\left(200 \cdot ((2x)^3 + (2y)^3)\right)$. et sauvez le nouveau script sous un nom de votre choix. Pour attenuer les fréquences au delà d'un seuil θ on se propose de flouter l'image par convolution avec une gaussienne bidimensionelle isotrope d'écart-type θ (On pourrait faire un choix plus conservateur...). La fonction `blurr.m` calcule la convolution d'une gaussienne, dont l'écart-type σ est à spécifier en pixels, avec l'image. Utilisez cette convolution comme filtre anti-aliasing avant de sous-échantillonner l'image et réutilisez le zoom en Fourier de

la question précédente pour reconstruire une grande image. Quel valeur choisirez-vous pour σ en fonction de $T = T_1 = T_2$? Qu'obtient-t'on ? Est-ce qu'on a supprimé de l'aliasing ? Imprimer une image qui illustre ceci ainsi que l'image sans filtre anti-aliasing correspondante.

8. (*Optionel*) Appliquer un très léger flou à l'image `brickwall2.jpg`. Qu'observez-vous en terme d'aliasing en reformatant la figure ?

5. Version discrete du théorème d'échantillonnage

On se propose ici de retrouver la fréquence de coupure de Nyquist qu'on a utilisé dans la section précédente.

1. Soit $f \in \mathbf{R}^N$ un signal discret. Quelle relation existe–il entre $f[n]$ et $f[N-n]$?
2. **Formule sommatoire de Poisson discrète**. On considère, pour $f \in \mathbf{R}^N$, sa version échantillonnée f_d avec un pas $T \in \{1, \dots, N\}$ qui peut s'écrire $f_d = Tfc$ avec $c = \sum_{p=0}^{N/T-1} \delta[\cdot - Tp]$ un peigne de Dirac. Calculer la transformée de Fourier discrète du peigne de Dirac et montrer que c'est un autre peigne de Dirac.
3. En déduire la forme de $\widehat{f_d}$ et quel filtre passe-bas il faut lui appliquer pour retrouver le signal initial sous l'hypothèse où le spectre de f est concentré sur les basses fréquences. En particulier en déduire la valeur de coupure de Nyquist.

Le compte-rendu est à envoyer sous format `pdf` à `guillaume.obozinski@imagine.enpc.fr` dans un délai de deux semaines.