TP1 : Familiarisation avec MATLAB : signaux et images digitales, DFT

Le langage interprété MATLAB se fonde, on le verra, comme son nom l'indique (MATrixLABoratory), sur le travail à partir de tableaux (matérialisés en termes mathématiques par des matrices). L'objectif de cette première séance de TP est de vous familiariser avec ce logiciel au travers précisément de ce travail sur les tableaux et de la transformation de Fourier discrète (dft) introduite en cours. On s'initiera aussi à la représentation visuelle avec les instructions plot (pour les signaux 1D), image, imagesc, imshow (visualisation des tableaux sous forme d'images, la luminance (ou encore *brilliance*) des pixels traduisant la taille des entrées, ce de manière inversement proportionnelle : blanc = grande entrée numérique, noir=petite entrée numérique). Vous serez aussi amenés avec les commandes du graphisme telles colormap.

Des aides à la prise en main de MATLAB (et Scilab) en pdf (malheureusement non interactives) sont disponibles sur ce site :

http://www.math.u-bordeaux1.fr/~ayger/MATLABSignal/Tutorials
http://www.math.u-bordeaux1.fr/~ayger/MATLABSignal/Tutorials

La première initMS.pdf a été rédigée par Philippe Thieullen (Bordeaux 1) pour l'ouvrage Mathématiques Appliquées L3 (A. Yger, J.A. Weil *ed.*), Pearson 2009 (disponible à la BU en plusieeurs exemplaires [1], tandis que la seconde (ne concernant que MATLAB) a été rédigée par Fanny Delebecque et Jean-Christophe Pesquet.

Téléchargez ces deux fichiers pdf depuis ces lien et sauvez les dans un répertoire TPSignal que vous aurez préalablement crée.

Ouvrez sur votre terminal MATLAB en tapant la commande matlab : vous devez disposer de quatre espaces (que vous pouvez visualiser ou non en utilisant les onglets Desktop et Window sur le bandeau) :

- *Command Window* (l'espace pour vous le plus important) où vous allez taper le plus souvent vos instructions (derrière de prompt >>) et voir s'afficher les résultats au terme de leur exécution [2]. Il est utile d'exploiter le fait que la machine garde en mémoire les instructions déjà effectuées (ceci est bien utile lorsque vous voulez taper une instruction similaire à une instruction

1. Le chapitre 10 de cet ouvrage (avec les tests et exercices correspondants) me servira très souvent dans ces TP.

2. Attention : mettre ; derrière une instruction vous évite de voir l'affichage (c'est exactement le contraire de ce qui se passe avec Maple12 !) ; ne rien mettre derrière vous fait au contraire courir le risque de voir s'afficher un résultat, ce qui n'est en règle générale pas souhaitable, surtout lorsque les variables en jeu sont de grands tableaux.

déjà effectuée : tapez les premières lettres et vous verrez apparaitre l'ancien modèle que vous pourrez alors corriger).
- *Command History* : c'est ici que s'affice l'historique des commandes. C'est une fenêtre que vous pouvez fermer pour ne pas encombrer votre écran de console.
- *Current Directory* : vous voyez dans cet espace le contenu du répertoire dans lequel vous vous êtes placé (qui devrait être toujours le répertoire `TPSignal` que vous venez de créer). C'est en particulier dans ce répertoire que se trouveront tous les fichiers `.m` correspondant à des `script` (par exemple commençant par `[.,., ...,.] = function (.,...,.)`) que vous aurez à écrire ou à utiliser, les fichiers `.mat` correspondant aux sessions ou aux variables que vous sauverez éventuellement, ainsi que les fichiers de données numériques (tableaux de nombres, images `jpeg`, *etc.*) préalablement téléchargés.
- *Workspace* : ici se trouvent recensées les variables actuellement affectées dans votre présente session de travail.

Les commandes `Unix` sont valables sous l'environnement `MATLAB` ; reportez vous au fascicule de TP 2011-2012 pour l'UE N1MA3003 (page 12), pour un rappel de ces commandes avec les touches `Ctrl + [...]`. Ce fascicule est téléchargeable ici :

`http://www.math.u-bordeaux1.fr/~ayger/initMS.pdf`

Le logiciel `MATLAB` installé sur les machines du CREMI est augmenté des cinq `toolboxes` : `Signal Processing`, `Image Processing`, `Wavelets`, `PDE`, `Statistique`. Les trois premiers pourront être mis à contribution dans ces TP. Est présent aussi le `toolbox` libre `Wavelab` rapatrié depuis ce site

`http://www-stat.stanford.edu/~wavelab/Wavelab_850/download.html`

Des routines prises dans ce package pourront s'avérer utiles. Pour le rendre opérationnel, ajouter sous l'onglet `Files/Set Path` :
`/opt/local/toolbox/Wavelab850`
Pour vous assurer de la présence de ce `package`, lancer les commandes suivantes :

```
>> WavePath
>> S1 = MakeSignal ('Piece-Regular',1024);
>> S2 = MakeSignal ('Blocks',1024) ;
>> S3 = MakeSignal ('LinChirp',1024);
>> S4 = MakeSignal ('TwoChirp',1024);
>> S5 = MakeSignal ('Chirps',1024);
>> S6 = MakeGabor ('Gabor',1024);
>> plot(S1) ; plot (S2) ;  .... ; plot(S6);
```

Comme vous le constatez, la routine `MakeSignal` nous servira à générer des signaux 1D.

Quelques tests préliminaires. Au contraire de `Maple12` (qui est un logiciel interprété de calcul symbolique ou formel), donc entrainé aux calculs *sans pertes* lorsque les variables d'entrée sont des entiers, des rationnels, ou des polynômes à coefficients entiers, rationnels, voire algébriques, `MATLAB` est un logiciel interprété de calcul scientifique, où les calculs numériques impliquent en général des nombres réels ou complexes déclarés en flottants (`floating`, soit -c'est le cas par défaut- en

double précision `double`, ou bien en simple précision `single`) et sont donc tributaires des arrondis inévitables liés à l'erreur machine (en accord toutefois avec la norme du standard `IEEE754`). Testez les commandes suivantes :

```
>> eps
>> realmax
>> realmin
```

A l'aide de `help` (exemple : tapez `help eps`), analysez le sens de ces variables modifiables ; comment interpréter le résultat de la commande

```
>> eps(x)
```

lorsque `x` est un nombre flottant (faites le test avec `eps(pi)`, avec `eps(1)`, `eps(1/2)`, `eps(`$10^{(20)}$ `+ pi)`, `eps(`10^7`+pi)`). Conclusion ? En quoi `eps(x)` peut il être considéré comme un indicateur de la précision au niveau du flottant `x` ? Le logiciel n'affiche par défaut que quatre chiffres représentatifs pour les nombres réels flottants (en double précision) ; pour en voir 16 (comme cela est théoriquement possible, aux arrondis près, *cf.* la valeur de `eps(x)`), il faut préalablement taper l'instruction :

```
>> format long
```

Testez ceci sur des exemples (par exemple sur les variables non effacables `i`, `j`, `sqrt(-1)`). Attention : `j` est le `i` des physiciens, ce n'est pas ici le `exp(2*i*pi/3)` des mathématiciens !

Exercice 1 (Manipulation de tableaux numériques sous `MATLAB`). Comme la terminologie utilisée le suggère, `MATLAB` est un logiciel à base de calcul matriciel. Les variables déclarées en entrée ou récupérées en sortie des routines sont des tableaux (soit de données numériques codées en simple ou double précision [`simple`, `double`], soit des données symboliques codées par exemple avec les formats `init8`, `unit8`, comme c'est par exemple le cas, on le verra, dans le codage des images basé sur le cube des couleurs (codage `RGB`). Le but de cet exercice est l'apprentissage de pareille manipulation de tableaux.

(1) Vérifiez que la matrice

$$A = \begin{pmatrix} 2 & -4 & -1 & -3 \\ 1 & 0 & -5 & 7 \\ -6 & -8 & 0 & 7 \end{pmatrix}$$

se déclare sous `MATLAB` ainsi :

```
>> A = [ 2 - 4  - 1  -3 ; 1  0 - 5  7 ; -6 - 8  0  7] ;
>> A
```

Déclarez, si vous avez compris, une matrice B cette fois à 4 lignes et 3 colonnes (mettez des entrées réelles arbitraires). Que se passe -il lorsque vous tentez les commandes :

```
>> C = A*B;
>> C = B*A;
>> BB = B';
>> C = BB*A;
>> C = A*BB;
>> C = A.*BB;
>> C = BB.*A;
```

Dans chaque cas où vous n'avez pas reçu un message d'erreur, affichez `BB` ou `C` en faisant

```
>> BB
>> C
```

(sans point virgule cette fois) et tentez d'analyser ce qui se passe. Les bases de calcul matriciel acquises en L1 et en L2 vous seront ici utiles.

(2) Les commandes `rand (M,N)` et `randn (M,N)` génèrent des matrices à `M` lignes et `N` colonnes dont les entrées sont des réalisations de variables aléatoires réelles (mutuellement indépendantes) suivant respectivement :
   - en ce qui concerne la fonction `rand`, la loi uniforme sur $[0, 1]$ (chaque entrée est un nombre flottant aléatoire entre 0 et 1, tous les choix étant équiprobables) ;
   - en ce qui concerne la fonction `randn`, la loi normale (moyenne 0 et écart type égal à 1) sur $\mathbb{R}$.

   Générez des matrices suivant les instructions :

```
>> A = rand(10,8);
>> A
>> Adouble = [A A];
>> Adouble
>> B = randn(12,9);
>> B
>> Bdouble = [B;B];
>> Bdouble
>> AA = A(1:2:10,1:2:8);
>> AA
>> BB = B(1:3:8,2:2:9);
>> ONES = ones (7,5);
>> ONES
>> ZEROS = zeros (10,7);
>> ZEROS
>> EYE = eye (10,5);
>> EYE
>> AAA = flipud (A);
>> AAA
>> BBB = fliplr (B);
>> BBB
```

   Analysez l'opération qu'exécute chacune de ces instructions. Qu'exécute en particulier l'instruction `Mat1=Mat(1:pas1:M,1:pas2:N)` si `Mat` est un tableau numérique à `M` lignes et `N` colonnes, `pas1` et `pas2` étant des entiers respectivement entre 1 et `M` et 1 et `N` ?

(3) Construisez une matrice `Apad` à 16 lignes et 16 colonnes en bordant la matrice `A` générée à la question (1) de manière symétrique (des deux côtés) par des blocs de zéros [3]. Faites ensuite la même chose avec cette fois des blocs de 1.

3. Pareille opération, fort utile dans la pratique, soit pour compléter une matrice rectangulaire en une matrice carrée pour des calculs ultérieurs, soit pour s'accorder une ≪ marge de sécurité ≫, telle un ≪ cadre ≫, autour d'un tableau ou d'une image, s'appelle le *zeropadding*.

(4) Créez une matrice à 19 lignes et 15 colonnes dont les lignes et les colonnes sont celles de la matrice `A` générée à la question (1), mais séparées cette fois entre elles par des lignes et des colonnes de zéros.

(5) Quelle est la méthode la plus judicieuse (au niveau du temps d'exécution) pour déclarer sous `MATLAB` les matrices :

$$U = \begin{pmatrix} 0 & 0 & 3 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 8 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 8 & 0 \\ 1 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 1 & 3 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 8 \\ 1 & 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 8 & 1 \\ 1 & 1 & 1 & 1 & 1 & 5 \end{pmatrix} \quad ?$$

Déclarez ces deux matrices de la manière que vous jugerez la plus efficace possible (au niveau du temps d'exécution).

Sauvez votre session dans votre répertoire `TPSignal` (celui dans lequel vous êtes précisément en train de travailler) avec l'instruction `save TP1exo1`. Vérifiez que vous avez bien ainsi créé un fichier `TP1exo1.mat` dans votre répertoire `TPSignal`. Faites ensuite l'instruction `clear all` pour rendre à nouveau vierge tout votre *Workspace*. Vérifiez que c'est bien le cas. Que se passe t'il si vous donnez les instructions :

```
>> load TP1exo1
>> who
```

Faites `clear all` à nouveau.

EXERCICE 2 (signaux analogiques discrétisés et transformation de Fourier discrète (`DFT`)). Les signaux audio constituent des exemples de signaux analogiques (fonctions d'une variable continue, à savoir le temps) qui ont été discrétisés. Si le pas temporel d'échantillonnage est de `1/Fs` seconde, on dit que le signal analogique [4] est échantillonné à `Fs` Hertz. La partie entière de `Fs` constitue donc le nombre d'échantillons collectés par seconde. Dans un signal audio au format `.wav` (par exemple `toto.wav`) se trouvent encodés la fréquence d'échantillonnage `Fs` (en Hertz, c'est-à-dire qu'il y a `Fs` échantillons par seconde), les échantillons successifs du signal, ainsi que le nombre de bits `B` utilisé pour coder chaque échantillon. La commande

```
>> [s,Fs,B] = wavread('toto.wav');
```

génère, outre les valeurs de `Fs` et `B`, le signal audio (analogique) discrétisé (c'est-à-dire les échantillons du signal audio codés initialement sur `B` bits) et converti au format numérique, `s(k)` correspondant à un nombre réel flottant codé en double précision et normalisé de manière à appartenir au segment $[-1, 1]$ (16 décimales dans la mantisse suivant la virgule dans le système décimal, vérifiez le en vous mettant sous `format long`).

(1) Depuis le répertoire

`http://www.math.u-bordeaux1.fr/~yger/MATLABSignal/Audio`

téléchargez les 10 fichiers `saxo-NOTE-wav`, `violon-NOTE-wav`, où `NOTE` est à prendre dans la liste : `do, mi, la, ladiapason, si` des notes de la gamme. Convertissez ces 10 fichiers en 10 fichiers digitaux `NOTE-saxo`,

4. Par exemple la position de la membrane du haut-parleur s'il s'agit d'un signal `audio`.

NOTE-violon (notés s) et vérifiez que les signaux analogiques correspondant ont été tous échantillonnés à Fs=48000 Hz. Affichez avec la commande plot l'un de ces signaux digitaux.

(2) Soit $S : t \mapsto S(t)$ un signal analogique défini sur un intervalle temporel d'une durée N/Fs (en seconde), où Fs figure un nombre entier et N=2M un nombre entier pair, à compter de $t = 0$, c'est-à-dire sur l'intervalle $[0, \mathtt{N/Fs}]$. On souhaite déterminer une fonction trigonométrique « balancée »

$$s : t \longmapsto \sum_{j=-M+1}^{M} c_{j+M}\, e^{\dfrac{2i\pi(j-1)\mathtt{Fs}}{\mathtt{N}}\, t}$$

de manière à ce que

$$\forall\, k = 1, ..., \mathtt{N}, \quad S((k-1)/\mathtt{Fs}) = s((k-1)/\mathtt{Fs}).$$

Vérifiez que le système de N équations à N inconnues que doivent satisfaire les nombres c_j pour qu'il en soit ainsi s'exprime :

$$\sum_{j=-\mathtt{M}+1}^{\mathtt{M}} c_{j+\mathtt{M}}\, e^{2i\pi(j-1)(k-1)/\mathtt{N}} = S((k-1)/\mathtt{Fs}), \quad k = 1, ..., \mathtt{N}$$

ou encore (en utilisant le changement d'indexation $j' = j + M$)

$$\sum_{j'=1}^{\mathtt{N}} c_{j'} W_{\mathtt{N}}^{-(j'-1)(k-1)} = (-1)^{k-1} S((k-1)/\mathtt{Fs}), \quad k = 0, ..., \mathtt{N}-1,$$

avec $W_{\mathtt{N}} = \exp(-2i\pi/\mathtt{N})$. L'inverse de la matrice $[W_{\mathtt{N}}^{-(j-1)(k-1)}]_{1\le j,k\le \mathtt{N}}$ étant

$$1/\mathtt{N} \times [W_{\mathtt{N}}^{(j-1)(k-1)}]_{1\le j,k\le \mathtt{N}}$$

et la transformation d'une variable X de longueur N (représentée en ligne ou en colonne) par l'application linéaire de matrice $[W_{\mathtt{N}}^{(j-1)(k-1)}]_{1\le j,k\le \mathtt{N}}$ étant assurée par l'une ou l'autre des commandes

```
>> U = fft(X);
>> U = fft(X,N);
```

vérifiez que le calcul de $c_{j'}$, $j' = 1, ..., \mathtt{N}$, s'obtient *via* :

```
>> C=fft(ss(1:N),N)/N;
```

si ss est le signal digital de longueur N dont la k-ième entrée ($k = 1, ..., \mathtt{N}$) est $(-1)^{k-1} * S((k-1)/\mathtt{Fs})$. Quels sont les indices j' tels que le coefficient $c_{j'}$ apparaisse comme le coefficient d'un signal oscillant de basse freǫuence (en valeur absolue) dans la décomposition du signal polynôme trigonométrique périodique s interpolant S ? Où se trouvent les indices j' tels que le coefficient $c_{j'}$ apparaisse comme le coefficient d'un signal oscillant de haute fréquence (en valeur absolue) dans la décomposition de ce même signal polynôme trigonométrique s ? Quelle est la période de S ? Observez l'effet de la commande fftshift :

```
>> C= fftshift(fft(ss,Fs))/Fs;
```

(si le signal s est trop court, il se trouve de force prolongé par des zéros). Comparez avec

```
>> CC = fft(s,Fs)/Fs;
```

Quel est le champ fréquenciel (en Hertz) exploré ? Prenez pour N des valeurs diverses (512, 1024, 2048, 4096) et choisissez au hasard une fenêtre d'observation sN de longueur N pour l'un de vos signaux digitaux NOTE-saxo ou Note-violon (plutôt au cœur du signal, aux alentours de 20000). Calculez le module et l'argument (phase) des entrées de

```
>> dft_sN = fftshift(fft(sN,N))/N);
```

Pourquoi le champ fréquenciel exploré reste-t'il $[-\mathtt{Fs}/2, \mathtt{Fs}/2]$, mais cette fois discrétisé avec un pas de Fs/N et non plus de 1 ?

(3) En privilégiant la valeur de N=2048 et en choissant une fenêtre d'observation judicieusement, précisez (en Hertz) les valeurs des fréquences correspondant aux divers harmoniques de la note enregistrée. Examinez le cas de diverses notes (en particulier le ladiapason), sur les deux instruments que sont le saxo et le violon.

(4) Après avoir étudié la fonction de la routine hamming, explicitez ce qu'effectue le code MATLAB suivant :

```
function [Omega,PSD] = welch1(s,Fs,fenetre,recouvrement);

 % function [Omega,PSD] = welch1(s,Fs,fenetre,recouvrement);
 % completer ici en decrivant la fonction de cette routine.

NN = length(s);
N  = length(fenetre);
Omega = - fix(Fs/2) : Fs/N : fix(Fs/2)-1;
PAS = N-recouvrement;
PSD=zeros(N,1);
q=0;
   for j= N+1: PAS : NN,
       y= fenetre.* s(j-N:j-1);
       PSD=PSD+(abs(fft(y))).^2;
       q=q+1;
   end
PSD=PSD/(q*(norm(fenetre))^2);
PSD1=PSD(1:fix(N/2));
PSD2=PSD(fix(N/2)+1:N);
PSD=[PSD2;PSD1];
plot(Omega,PSD);
```

Ce processus d'inspiration statistique correspond à l'implémentation de ce que l'on appelle la *transformation de Welch*. Cette transformation correspond à un calcul statistique du carré du module du spectre (sur une gamme de N fréquences) par simple moyennisation des résultats obtenus sur des fenêtres d'exploration du signal (chacune de longueur N). Ce calcul correspond à la version discrètisée du calcul de la *densité spectrale de puissance* (psd) dont on reparlera.

(5) Générez des signaux se présentant comme des fonctions présentant des discontinuités, par exemple :

```
>> S1 = MakeSignal ('Blocks',2048);
>> S2 = MakeSignal ('Piece-Regular',2048);
```

Calculez leur `fft` :

```
>> spectreS1 = fft (S1);
>> spectreS2 = fft (S2);
```

Introduisez une fenêtre `fenetre` de longueur `2048` mettant de manière brutale les deux spectres à 0 hors des bandes [0 300] et [2048-300 2048] ; ces deux bandes concernent-elles la gamme des hautes ou des basses fréquences ? Calculez ensuite les deux signaux

```
>> S1tronc = real (ifft (spectreS1.*fenetre);
>> S2tronc = real (ifft (spectreS2.*fenetre);
```

Qu'observez vous en comparant l'affichage de ces signaux avec ceux des signaux originels :

```
>> plot(S1)  hold plot(S1tronc,'r');
>> figure
>> plot(S2) hold plot(S2tronc,'r');
```

Pourquoi fallait-il impérativement prendre la partie réelle ? Que se passe-t'il si l'on remplace 300 par une valeur plus grande ? plus petite ? Le phénomène que vous observez ici est le *phénomène de Gibbs*, on y reviendra. Pensez juste pour l'instant aux problèmes que peut poser le « repiquage » de vieux enregistrements (pleins de « cracks »). Examinez ce qui se passe lorsqu'au lieu de couper « brutalement » les hautes fréquences, on les coupe « en douceur ». Pour couper ainsi, utilisez cette fois soit la moitié gauche, soit la moitié droite du graphe de la fonction fenêtre déclarée en ligne par :

```
>> H = hamming (600)';
```

(affichez cette fonction avec `plot` au préalable pour avoir une idée de l'allure du graphe, symétrique par rapport à l'axe médian). Refaites les manipulations précédentes à partir des deux signaux `S1` et `S2`. Qu'observez vous cette fois relativement au phénomène de Gibbs ?

EXERCICE 3 (décomposition de Haar d'une image). Voici encore un exercice prétexte à la manipulation de tableaux sous `MATLAB`. Vous y verrez aussi l'étroite relation entre tableaux de nombres (donc matrices) et images. Dans un premier temps, on propose une initiation au chargement des images, exactement comme nous avions vu dans l'exercice précédent comment charger des fichiers audio.

Téléchargez depuis le site

`http://www.math.u-bordeaux1.fr/~yger/MATLABSignal/Images`

le fichier `venise.jpg` et enregistrez ce fichier dans votre répertoire `TPSignal`. Il s'agit ici d'une imge au format `jpg` que j'ai préalablement téléchargé depuis le web. Chargez cette fonction sous MATLAB en utilisant la commande :

```
>> I= imread('venise.jpg');
```

Déterminez ensuite la taille de cette image :

```
>> size(I)
```

Constatez en faisant des tests que `I` est un tableau à trois entrées : les deux premières entrées figurent la position du pixel (ligne, colonne) ; une fois ces entrées fixées (par exemple `i` et `j`, les trois sorties `I(i,j,1), I(i,j,2), I(i,j,3)` sont des nombres entiers codés entre `0` et `255` (8 bits car $256 = 2^8$), correspondant au codage `R,G,B` dans le cube des couleurs. Pour travailler du point de vue numérique sur cette image, on la convertit d'abord au format Noir et Blanc (`gray`) :

```
>> II = rgb2gray(I);
```

Cette fois `II` est devenu un tableau à deux entrées, mais toujours au format `uint8` (entrées entre `0` et `255`, du noir [luminance minimale] au blanc [luminance maximale]. Pour convertir ce tableau en un tableau de nombres flottants entre 0 et 1 (en double précision) entre 0 et 1 (et pouvoir ultérieurement traiter cette image du point de vue nnumérique), il reste à effectuer la commande :

```
>> Inumer = double(II);
```

Testez les commandes :

```
>> image(I);
>> imagesc(I);
>> image(II);
>> imagesc(II);
>> image(Inumer);
>> imagesc(Inumer);
```

Téléchargez toujours depuis
`http://www.math.u-bordeaux1.fr/~yger/MATLABSignal/Images`
cette fois le fichier (de données) `venise`. Une fois ceci fait, « chargez » ce fichier dans votre *Workspace* en tapant sous la *Command Window* les instructions :

```
>> load venise
>> I=venise;
```

Grâce à la commande `size`, vérifiez que la variable ainsi déclarée `I` est un tableau à 256 lignes et 256 colonnes [5]. Testez les valeurs de quelques entrées `I(i,j)` pour voir qu'il s'agit en fait d'une matrice dont les entrées sont des nombres entiers positifs (mais considérés ici comme des nombres flottants en double précision, comme la réponse `1` à l'instruction `isfloat(I)` vous le prouvera, vérifiez le).

(1) Visualisez la matrice `I` de deux manières :
   - en utilisant la visualisation en 3D suivant `mesh(I)` ;
   - en utilisant la visualisation en 2D suivant `image(I)`, `imagesc(I)` (voire aussi `imshow(I,[])` si vous êtes au CREMI, *cf.* plus loin).

   Quelle est, pour ce type de matrice `I`, de ces deux visualisations, celle qui est la plus adaptée ? Quel est l'effet de la commande

   ```
   >> II = imrotate(I,angle);
   ```

   (où `angle` désigne la valeur en flottant d'un angle entre 0 et 360 degrés exprimé en flottant) ? [6]. On utilisera l'une des routines du type `image` mentionnées plus haut pour visualiser l'image `II` ainsi qu'une instruction convenable (faire `help axis`) pour que l'affichage respecte la taille (ici

5. Notez que 256 est une puissance de 2, ce qui n'est, on le verra plus tard, nullement un hasard en ce qui concerne les formats d'images `jpeg`.

6. Cette instruction `imrotate` bien utile n'est malheureusement pas présente dans la bibliothèque de `Scilab`.

carrée) de l'image. Quelle est la difficulté liée à la réalisation mathématique d'une telle transformation de matrice (pensez au passage entre le repérage cartésien dans une grille de pixels et le repérage polaire) ? Que se passe t-il en particulier lorsqu'un pixel (i, j) tourne d'un certain angle ? Tombe t'on encore sur un pixel de la grille cartésienne ?

(2) Ouvrez un fichier `.m` vierge en utilisant l'onglet *File* du bandeau. Vous allez essayer dans ce fichier de rédiger un code, dont la première instruction (déclaration des entrées et sorties de la fonction) sera

```
function [RR,DH,DV,DO] = Haar (ipt)
```

qui est censé calculer, étant donnée une matrice de flottants `ipt` de taille $(2^N, 2^N)$, les quatre matrices, de taille $(2^{N-1}, 2^{N-1})$, dont les entrées sont respectivement les nombres

$$\begin{aligned}
\texttt{RR}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}\\
\texttt{DH}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DV}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DO}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}
\end{aligned}$$

Testez votre code en l'exécutant avec comme matrice `ipt` la matrice `I` :

```
>> [RR, DH, DV, DO] = Haar(I);
```

Affichez avec `image` (ou mieux `imagesc`) les quatre matrices `RR, DH, DV, DO`. Pour le rendu des couleurs et du graphisme, vous pouvez jouer avec la commande `colormap` (exemples les instructions `colormap hot`,`colormap jet`, `colormap pink`, `colormap jet`, *etc.*, faire `help colormap` pour décider). Pour un meilleur rendu graphique, vous pouvez aussi faire appel à l'instruction

```
>> imshow(MATRICE,[ ])
```

Cette commande est dans la bibliothèque du *Toolbox* `Image Processing` (disponible sous `MATLAB` au CREMI). Justifiez la terminologie : `RR` = *Résumé*, `DH` = *Détails Horizontaux*, `DV` = *Détails Verticaux*, `DO` = *Détails Obliques*. Recommencez le traitement de la matrice `I` cette fois sur la matrice `RR`. Qu'observez vous sur les quatre images de taille $(64, 64)$ ainsi obtenues ? Poursuivez si vous voulez pour obtenir quatre images de taille (32,32) en décomposant le nouveau résumé. La décomposition que vous êtes en train d'effectuer ici s'appelle *décomposition de Haar* : elle permet d'≪ isoler ≫ les ≪ structures ≫ cohérentes des détails (ou ≪ accidents ≫) d'une image digitalisée. Sauvez votre fichier `.m` sous le nom `Haar` si vous ne l'avez pas encore fait et fermez le.

(3) Ouvrez un nouveau fichier vierge `.m` depuis l'onglet *File* du bandeau. Rédigez dans ce fichier un code (correspondant encore à une fonction) dont la première instruction sera

```
function II = HaarInverseAux (I1,I2,I3,I4)
```

qui, étant donnée quatre matrices `I1,I2,I3,I4` de taille $(2^{N-1}, 2^{N-1})$, fabrique une matrice `II` de taille $(2^N, 2^N)$ telle que

$$\texttt{II}(2i-1, 2j-1) = \texttt{I1}(i,j),\ \texttt{II}(2i-1, 2j) = \texttt{I2}(i,j)$$
$$\texttt{II}(2i, 2j-1) = \texttt{I3}(i,j),\ \texttt{II}(2i, 2j) = \texttt{I4}(i,j)$$

pour toute paire d'entiers (i, j) entre 1 et 2^{N-1}. Sauvez ce nouveau fichier `.m` sous le nom `HaarInverseAux`. Vous pourrez être amenés à utiliser ce code comme code auxiliaire par la suite.

(4) Ouvrez un nouveau fichier `.m` où vous réaliserez un code (toujours une fonction) dont la première instruction sera cette fois

```
function I=HaarInverse (RR,DH,DV,DO)
```

qui, étant données quatre matrices `RR, DH, DV, DO`, toutes de même taille $(2^{N-1}, 2^{N-1})$, reconstitue une matrice I de taille $(2^N, 2^N)$ de manière à ce que ce code exécute l'opération inverse de celle exécutée par la fonction `Haar`. Validez votre résultat en utilisant la matrice `I` de la question 1. Sauvez votre fichier `.m` comme `HaarInverse`.