TP1 : INITIATION À MATLAB

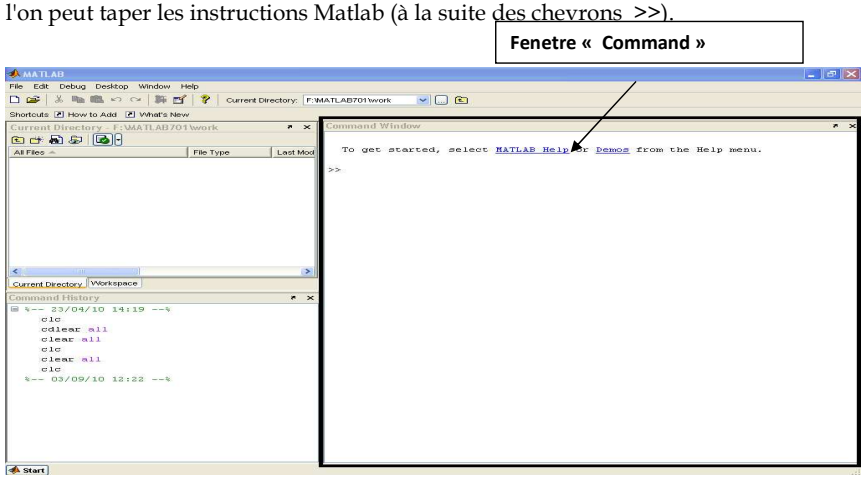
1. Introduction

Matlab (abréviation de " **MAT**rix **LAB**oratory ") est un puissant logiciel de calcul numérique. C'est un environnement informatique conçu pour le calcul matriciel. L'élément de base est une matrice dont la dimension n'a pas à être fixée. Matlab est un outil puissant qui permet la résolution de nombreux problèmes en beaucoup moins de temps qu'il n'en faudrait pour les formuler en C ou en Pascal. La programmation au même titre que C, Pascal ou Basic. Par contre, sa singularité est qu'il s'agit d'un langage interprété, c'est-à-dire que les instructions sont exécutées immédiatement après avoir été tapées.

Le but de ce TP est de vous familiariser avec l'utilisation de MATLAB.

1.1. Lancement

Au lancement de Matlab, la fenêtre "command" apparaît. C'est dans cette fenêtre que l'on peut taper les instructions Matlab (à la suite des chevrons >>).



1.2. Langage interprété

Puisque Matlab est un langage interprété. Il n'est pas nécessaire de compiler un programme avant de l'exécuter. Toute commande tapée dans la fenêtre de commande est immédiatement exécutée.

```
>> 2+2
ans = 4
>> 5^2
ans = 25
```

La réponse est affichée si on ne met pas de point-virgule en fin de ligne. Elle est de plus stockée dans une variable nommée **ans** (**ans**wer). La plupart des fonctions mathématiques usuelles sont définies dans Matlab, et ceci sous une forme naturelle (sin, cos, exp, ...). C'est également le cas de certaines constantes comme pi par exemple :

```
>> 2*sin(pi/4)
ans = 1.4142
```

1.3. Les variables

On peut évidemment indiquer le nom de la variable dans laquelle le résultat doit être stocké (ce nom doit commencer par une lettre et occuper moins de 19 caractères).

Remarque : Matlab distingue les minuscules des majuscules.

```
>> x = pi/4
x = 0.7854
```

Le nom de la variable ainsi que le résultat sont alors affichés. On peut taper plusieurs commandes par ligne si elles sont séparées par un point virgule.

```
>> x = pi/2; y = sin(x);
```


1.4. Les variables complexes

Matlab travaille indifféremment avec des nombres réels et complexes. Par défaut les variables i et j sont initialisées à la valeur complexe. Naturellement si vous redéfinissez la variable i ou j avec une autre valeur elle n'aura plus la même signification.

```
>> z = 3 + 2*i
z = 3.0000 + 2.0000i
```

Les fonctions usuelles de manipulation des nombres complexes sont prédéfinies dans Matlab : real, imag, abs, angle (en radian), conj.

```
>> r = abs(z);
>> theta = angle(z);
>> y = r*exp(i*theta);
```

1.5. Les vecteurs, les matrices et leur manipulation

En fait, toute variable de Matlab est une matrice (scalaire : matrice 1x1, vecteur : matrice 1xN ou Nx1). On peut spécifier directement une matrice sous la forme d'un tableau avec des crochets, l'espace ou la virgule sépare deux éléments d'une même ligne, les points virgules séparent les éléments de lignes distinctes.

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

Les éléments d'une matrice peuvent être n'importe quelle expression de Matlab :

```
>> x = [ -1.3, sqrt(3), (1+2+3)*4/5 ]
x = -1.3000 1.7321 4.8000
```

Les éléments d'une matrice peuvent ensuite être référencés par leurs indices, on utilise alors des parenthèses et non des crochets. Le mot-clé **end** peut être utilisé en indice pour signifier le dernier élément.

```
>>x(2)
ans = 1.7321
>>x(5) = abs(x(1))
x = -1.3000 1.7321 4.8000 0.0000 1.3000
>> x(end-1)
ans =0
```

On peut remarquer que la taille du vecteur x a été ajustée en remplissant les éléments non précisés par 0. On peut aussi créer des matrices avec les fonctions zeros, ones et eye, ces fonctions créent des matrices de la taille précisée, respectivement remplies de zéros, de un, et de un sur la diagonale et de zéros ailleurs (eye =prononciation anglaise de I comme identité).

```
>> eye(2,3)
ans = 1 0 0
 0 1 0
>> ones(1,5)
ans = 1 1 1 1 1
```

On peut avoir des informations sur la taille d'une matrice :

```
>> size(x)
ans = 1 5
>> length(x)
ans = 5
```

On peut ajouter des lignes et des colonnes à des matrices déjà existantes (attention, les dimensions doivent toutefois être compatibles ...).


```
>> r1 = [10, 11, 12];
>> A = [A ; r1]
A =  1  2  3
     4  5  6
     7  8  9
     10 11 12
>> r2 = zeros(4,1);
>> A = [A, r2]
A =   1   2  3  0
      4   5  6  0
      7   8  9  0
      10 11 12  0
```

1.6. L'opérateur " : "

L'opérateur " : ", sous Matlab, peut être considéré comme l'opérateur d'énumération. Sa syntaxe usuelle est :« deb:pas:fin ». Il construit un vecteur dont le premier élément est deb puis deb+pas, deb+2*pas... jusqu'à deb+n*pas tel que deb+n*pas < fin < deb+(n+1)*pas. « : » est très utile pour construire des signaux.

```
>> x = 0.5:0.1:0.85
x = 0.5000 0.6000 0.7000 0.8000
```

Le pas d'incrémentation peut être omis, un pas de 1 est alors pris par défaut :

```
>> x =1:5
x = 1 2 3 4 5
```

On peut aussi utiliser le " : " pour sélectionner des éléments d'un vecteur ou d'une matrice :

```
>> A(1,3)   % Troisième élément de la première ligne de A
>> A(1,1:3) % Premier, deuxième et troisième éléments de
            % la première ligne de A
>> A(1,:)   % Tous les éléments de la première ligne
>> A(:,3)   % Tous les éléments de la troisième colonne
>> A(:)     % Vecteur colonne contenant tous les éléments
            % de A lus colonne par colonne.
```

1.7. Opérations matricielles

Les opérations usuelles sont définies de façon naturelle pour les matrices :

```
>> 2*A % Produit par un scalaire
>> A*B % Produit de deux matrices (de dimensions cohérentes)
>> A^p % Elève la matrice carrée A à la puissance p
>> inv(A) % Inversion d'une matrice carrée inversible (message d'alerte éventuel)
>> A.*B % Produit élément par élément de deux matrices / IMPORTANT
>> X = A\B % Donne la solution de A*X = B (équivalent à X = inv(A)*B)
>> X = B/A % Donne la solution de X*A = B (équivalent à X = B*inv(A))
>> X = A./B % Division éléments par éléments
```

Autres fonctions utiles pour les opérations matricielles

```
>> max(A)
```

renverra un vecteur ligne dont chaque élément sera le maximum de la colonne correspondante. De la même façon est définie la fonction min().

```
>> poly(A) % Polynôme caractéristique de A
>> det(A) % Déterminant de A
>> trace(A) % Trace de A
>> [V, D] = eig(A)%Renvoie les valeurs propres et les vecteurs propres de A
```

2. Affichages graphiques et alphanumériques

2.1. Affichage alphanumérique

On peut afficher des chaînes de caractères dans la fenêtre de commande :

```
>> message = 'bienvenue sur Matlab';
>> disp(message)
bienvenue sur Matlab
```

Les fonctions sprintf et fprintf existent également


```
>> fprintf('pi vaut %f\n',pi)
pi vaut 3.141593
```

On peut aussi demander des valeurs à l'utilisateur :

```
>> rep = input('Nombre d''itération de l''algorithme : ');
```

Matlab affichera la chaîne de caractère entrée en paramètre et attendra une réponse de l'utilisateur.

2.2. Affichages graphiques de courbes 1D

Matlab permet un grand nombre de types d'affichage 1D et 2D, seuls les plus courant seront décrits ici. La commande plot permet l'affichage d'une courbe 1D :

```
>> x = 0:0.1:2; y = sin(x*pi);
>> plot(x*pi,y) % plot(abscisse,ordonnée)
```

On peut ajouter un titre aux figures ainsi que des labels aux axes avec les commandes title, xlabel, ylabel:

```
>> title('Courbe y = sinus(pi*x)')
>> xlabel('x'); ylabel('y')
```

2.3. Affichages graphiques de courbes 2D

Avec la commande mesh on peut aisément avoir une représentation d'une fonction 2D. Cependant il faut construire la grille des coordonnées des points en lesquels on a les valeurs de la fonction à l'aide de la fonction meshgrid.

```
>> x = 0:0.1:2;y = -1:0.1:1;
>> [X,Y] = meshgrid(x,y);
>> mesh(X,Y,cos(pi*X).*sin(pi*Y)) % Coordonnées en x et en y et
                                  % valeurs de la fonction : Z
>> xlabel('x');ylabel('y');zlabel('z');
>> title('z=cos(pi*x)*sin(pi*y)')
```

2.4. Affichage de plusieurs courbes

On peut bien évidemment vouloir afficher plusieurs courbes à l'écran. Pour cela deux solutions s'offrent à nous :On peut effectuer plusieurs affichages sur une même figure en utilisant la commande subplot qui subdivise la enêtre graphique en plusieurs sous figures. Sa syntaxe est :

«subplot(nombre_lignes,nombre_colonnes,numéro_subdivision)».Les subdivisions sont numérotés de 1 à nombre_lignes*nombre_colonnes, de la gauche vers la droite puis de haut en bas.

```
>> subplot(3,2,1)
>> plot(x,y)
>> subplot(3,2,2)
>> plot(x,y.^2)
```

On peut aussi ouvrir une deuxième fenêtre graphique à l'aide de la commande figure. Le passage d'une fenêtre graphique à une autre pourra alors se faire à la souris ou en précisant le numéro correspondant dans la commande figure(n). **NB** : on peut également superposer plusieurs courbes sur le même référentiel, en utilisant la commande hold on (resp. hold off).

3. Environnement de travail, scripts et fonctions

3.1. Répertoire de travail

Il est indispensable de travailler dans votre propre répertoire et non dans le répertoire de Matlab. De même dans votre répertoire vous pouvez organiser vos fichiers dans des sous répertoires. Pour que vos fonctions et scripts soient accessibles à partir de la ligne de commande Matlab il faut le répertoire dans lequel vous allez travailler, en utilisant les commandes cd et dir.

3.2. Sauvegarde et chargement de variables

Il peut être utile, lors d'une session de travail, de sauvegarder des variables dans des fichiers du répertoire de travail. Cela peut être réalisé à l'aide de la fonction save dont la syntaxe est :

« save nom_fichier nom_variables ».

```
>> save test.mat A, x, y
```

Si le nom des variables est omis, tous l'espace de travail est sauvé; si l'extension du fichier est omise elle sera automatiquement .mat, si de plus, le nom du fichier est omis, la sauvegarde se fera dans le fichier matlab.mat. Pour recharger des variables sauvées dans un fichier, il suffit de taper :

```
>> load nom_fichier
```

Si le nom du fichier est omis, Matlab chargera le fichier matlab.mat. On peut à un moment donné vouloir connaître toutes les variables disponibles dans l'espace de travail. Cela est possible grâce aux commandes who et whos. La commande who nous donne le nom de toutes les variables existantes, tandis que whos nous donne leur nom et l'espace mémoire qu'elles occupent.

3.3. Scripts

Il est parfois (souvent) souhaitable, pour ne pas avoir à taper plusieurs fois une même séquence d'instructions, de la stocker dans un fichier. Ainsi on pourra réutiliser cette séquence dans une autre session de travail. Un tel fichier est dénommé script. Sous windows, il suffit d'ouvrir un fichier avec le menu file, new, de taper la séquence de commande et de sauver le fichier avec une extension " .m " (nom_fich.m). En tapant le nom du fichier sous Matlab, la suite d'instructions s'exécute. Les variables définies dans l'espace de travail sont accessibles pour le script. De même les variables définies (ou modifiées) dans le script sont accessibles dans l'espace de travail. On peut (doit) mettre des commentaires à l'aide du caractère

pour-cent " % ". Toute commande située après " % " n'est pas prise en compte par Matlab, jusqu'à la ligne suivante.

3.4. Fonctions

On a parfois (souvent) besoin de fonctions qui ne sont pas fournies par Matlab. On peut alors créer de telles fonctions dans un fichier séparé et les appeler de la même façon que les fonctions préexistantes. La première ligne (hormis les lignes de commentaires) d'une fonction doit impérativement avoir la syntaxe suivante :

```
function [ var de sorties, ...] = nom_fonction( var d'entrée, ...)
```

Exemple de fonction :

```
function y = sinuscardinal(x)
z = sin(x); % Variable de stockage
y = z./x; % Résultat de la fonction % Il faudra se méfier de la division pour x=0
```

Cette fonction pourra être appelée par :

```
>> sincpi = sinuscardinal(pi);
```

Exemple de fonction à plusieurs variables de sortie :

```
function [mini, maxi] = minetmax(x)
mini = min(x); % Première variable de sortie
maxi = max(x); % Deuxième variable de sortie
```

Cette fonction pourra être appelée par :

```
>> [miny, maxy] = minetmax(y);
```

Le nom de la fonction doit impérativement être le même que le nom du fichier dans lequel elle est stockée (sinon Matlab ne prendra pas en compte ce nom mais uniquement celui du fichier). Les nombres d'arguments en entrée et en sortie ne sont pas fixes et peuvent être récupérés par nargin et nargout. Cela nous permet donc de

faire des fonctions Matlab pouvant dépendre d'un nombre variable de paramètres. Les variables de l'espace de travail ne sont pas accessibles à la fonction sauf si elles sont entrées comme variables d'entrée. De même les variables définies dans une fonction ne sont pas accessibles dans l'espace de travail.

4. Boucles et contrôles

Comme de nombreux autres langages de programmation, Matlab possède trois types d'instructions de contrôles et de boucles : for, if et while.

4.1. Contrôle : " if "

```
« if (expression logique) »
sute d'instructions 1;
else
suite d'instructions 2;
end
```

La condition est exprimée au travers d'une expression logique. Les expressions logiques sont des expressions quelconques pour lesquelles on considère uniquement le fait qu'elles soient nulles (expression fausse) ou non (expression vraie). On peut aussi construire ces expressions à l'aide des opérateurs logiques et (&), ou (|), égal (==), supérieur (>,>=), inférieur(<,<=), non(~), ou exclusif (xor) ou à l'aide de fonctions logiques prédéfinies exist, any, find , isinf, isnan...

4.2. Boucle : « while »

```
while (expression logique)
suite d'instructions
end
```

La suite d'instructions est répétée un nombre indéfini de fois jusqu'à ce que l'expression logique soit fausse.

4.3. Boucle : "for"

La boucle for a pour syntaxe :

```
for i=1:n
suite d'instructions;
end
```

Cette structure de contrôle est très utile, mais il est recommandé d'éviter au maximum les boucles dans Matlab, car elles sont très lentes par rapport à un traitement matriciel. Par exemple, la boucle suivante :

```
for i=1:N, x(i) = i; end
```

est équivalente à

```
x = 1:N;
```

Mais s'exécutera beaucoup plus lentement. Essayez avec N = 10, 100, 1000, etc. et comparez les temps. Malheureusement, on ne rencontre pas toujours des cas aussi simples et il est difficile de penser matriciel, c'est à dire penser en Matlab. Les fonctions zeros et ones, permettent parfois d'éviter de mettre en œuvre une boucle.

Par exemple, au lieu de faire la boucle :

```
r = 1:10;
A =[]; % initialisation de A à vide
for i=1:5, A = [A ; r]; end
```

On peut écrire :

```
r = 1:10;
A = ones(5,1)*r;
```

De plus certaines fonctions Matlab agissent sur une matrice en tant que suite de vecteurs.

Ainsi

```
maximum = max(A);
```


nous donne le même résultat (mais en moins de temps) que :

```
n = length(A(1,:));
for i=1:n, maximum(i) = max(A(:,i)); end
```

5. Help / Aide en ligne

Matlab est pourvu d'une fonction d'aide très utile : help. Ainsi si vous tapez help sur la ligne de commande apparaissent tous les répertoires accessibles depuis Matlab ainsi qu'une explication concernant ces répertoires. Si vous tapez help suivi du nom d'un répertoire, Matlab affiche une explication brève sur toutes les fonctions et scripts de ce répertoire. Enfin si vous tapez help suivi du nom d'un fichier (script ou fonction), apparaît une explication détaillée sur l'utilisation de la fonction ou du script. De plus vous pouvez créer une aide sur les fonctions et scripts que vous créez et sur vos répertoires. Pour une fonction, Matlab prendra comme help toutes les lignes de commentaires suivant l'instruction function (en première ligne). Pour un script, Matlab prendra toutes les lignes de commentaires placés en début de fichier. Ces aides que vous créez pour vos fonctions et scripts sont intéressantes pour un utilisateur éventuel, mais aussi pour vous, pour vous rappeler le but de telle fonction ou pour vous souvenir de sa syntaxe.

Exemple simple :

```
function [mini, maxi] = minetmax(x)
% [mini, maxi] = minetmax(x)
% Cette fonction renvoie les éléments minimum et maximum d'un vecteur
```

6. Exercices d'application

Exercice 1 :

Ecrire une fonction « traitement_image_2D(x) » qui prend comme paramètre d'entrée le nom d'une image et qui renvoie en sortie le nombre de pixels ayant la couleur blanche et le nombre de pixels ayant la couleur noire.

La réalisation d'une telle fonction se fait comme suit :

1. Lecture de l'image dont le nom est le paramètre d'entrée de la fonction (en utilisant la fonction imread 'voir Help').

2. Affichage de l'image (en utilisant la fonction imshow).
3. Balayer l'image et chercher le nombre de pixel correspondant à la couleur blanche et ceux correspondant à la couleur noire.

Exercice 2 :

Ecrire une fonction « traitement _surface_3D(x) » qui prend comme paramètre d'entrée le nom d'un fichier contenant les coordonnées x,y et z des points d'une surface 3D et qui renvoie en sortie le point parmi tous ces points qui soit le plus proche au centre de gravité de cette surface

La structure du fichier est la suivante :

V x_1 y_1 z_1

V x_2 y_2 z_2

. . . . .

. . . . .

. . . . .

V x_n y_n z_n

La réalisation de cette fonction se fait comme suit :

1. Ouverture en lecture du fichier dont le nom est le paramètre d'entrée de la fonction (utiliser la commande 'fopen').
2. Lecture du fichier et stockage des coordonnées des points dans un vecteur X selon l'axe des x, dans un vecteur Y selon l'axe des y et dans un vecteur Z selon l'axe des z. (utiliser la fonction fscanf pour la lecture. Voir 'Help').

   NB : Le nombre de ligne du fichier cad le nombre de points n'est pas connu en avance donc il faut trouver une condition d'arrêt de la lecture.
3. fermeture du fichier (la fonction fclose).
4. Affichage de la surface 3D en utilisant le commande plot3 (voir 'help').
5. Calcul du centre de gravité de cette surface.
6. chercher parmi les points de la surface 3D le point qui soit le plus proche à ce centre de gravité.

TP2 : GÉNÉRATION DES SIGNAUX

Ce TP a pour but de familiariser l'étudiant avec l'utilisation du logiciel MATLAB pour la génération des différents types de formes d'ondes.

The Signal Processing Toolbox de MATLAB contient des fonctions pour la génération des formes d'ondes couramment utilisées périodiques et non périodiques, séquences (impulsion, échelon, rampe), train d'impulsions…

1. Génération des signaux : Impulsion Unité et Echelon Unité

Deux signaux élémentaires de base sont : l'impulsion Unité et l'échelon Unité.

L'impulsion unité u[n] de longueur N peut être générée en utilisant la commande MATLAB suivante :

```
u = [1   zeros(1,N-1)] ;
```

De même, l'impulsion unité ud[n] de longueur N et décalées de M échantillons, tel que M<N, peut être générée par la commande MATLAB suivante :

```
ud = [zeros(1,M)     1        zeros(1,N-M-1)];
```

L'échelon unité s[n] de longueur N peut être généré en utilisant la commande MATLAB suivante :

```
s=[ones(1,N)]
```

La version décalée de l'échelon unité s'obtient de la même manière que dans le cas de l'impulsion unité.

Le programme P1 peut être utilisé pour générer et tracer un signal impulsion unité :

```
**************************************************************************

%Programme P1

% Génération d'une impulsion unité
```

```
% Génération d'un vecteur de -10 à 20
 n=-10:20;
% Génération de l'impulsion unité
u = [zeros(1,10)  1  zeros(1,20)];
% Tracer le signal généré
stem(n,u);
xlabel('Temps indexé en n'); ylabel('Amplitude');
title('Impulsion Unité');
axis([-10 20 0 1.2]);
```

****

Application :

1- Modifier le programme précédent pour générer une impulsion unité décalée de 11 échantillons. Exécuter le programme et tracer le signal généré.
2- Modifier le programme précédent pour générer un échelon unité décalé de 20 échantillons.

2. Génération des signaux périodiques

Une autre classe de signaux très utile en traitement du signal et en automatique sont les signaux sinusoïdaux de la forme :

$$x[n]=A.\cos(\omega_0 n+\phi)$$

Avec MATLAB, ce type de signaux peut être généré en utilisant les opérateurs trigonométriques cos et sin.

Le programme P2 est un exemple qui génère un signal sinusoïdal.

****

```
% Programme P2
% Génération d'un signal sinusoïdal
n = 0:100;
f = 0.1;
phase = 0;
A = 1.5;
arg = 2*pi*f*n - phase;
x = A*cos(arg);
clf;   %efface l'ancienne figure
stem(n,x);
axis([0 100 –2 2]);
grid;
title('Signal Sinusoidal');
xlabel('Temps indexé en n');
ylabel('Amplitude') ;
axis ;
```

****

Application :

1- Modifier le programme précédent pour générer et tracer un signal sinusoïdal de longueur 50, fréquence 0.08, amplitude 2.5 et phase décalée de 90 degrés.
2- Générer et tracer les séquences définies ci dessus :

$x_1(n) = \sin\left(\frac{\pi}{17}n\right)$ $0 \le n \le 35$

$x_2(n) = \sin\left(\frac{\pi}{17}n\right)$ $-15 \le n \le 25$

$x_3(n) = \sin\left(3\pi n + \frac{\pi}{2}\right)$ $-15 \le n \le 15$

$x_4(n) = \sin\left(\frac{\pi}{\sqrt{23}}n\right)$ $0 \le n \le 50$

En plus des fonctions sin et cos, le toolbox de MATLAB offre d'autres fonctions qui produisent des signaux périodiques comme sawtooth et square.

La fonction sawtooth génère une onde triangulaire avec des maximums en +-1 et une période de 2*pi.

La fonction square génère une onde carrée avec une période de 2*pi. Un paramètre optionnel spécifie le cycle, la période en pourcent pour laquelle le signal est positif.

Le programme P3 génère 1.5 secondes d'onde triangulaire (respectivement carré), de fréquence 50Hz et avec une fréquence d'échantillonnage de 10KHz.

****

```
%Programme P3

fs = 10000;

t = 0:1/fs/1.5;

x1 = sawtooth(2*pi*50*t);

x2 = square(2*pi*50*t);

subplot(211), plot(t,x1), xlabel('Time (sec)'); ylabel('Amplitude'); title('Sawtooth Periodic Wave')
```



```
subplot(212), plot(t,x2), xlabel('Time (sec)'); ylabel('Amplitude'); title('Square Periodic Wave')
```

****

3. Génération des signaux exponentiels

Un autre type de signaux de base sont les signaux exponentiels. Ce type de signaux peut être généré par les opérateurs de MATLAB « .^ » et « exp ».

Le programme P4 est employé pour générer un signal exponentiel x[n] = A exp(jωn) à valeurs complexes.

****

```
% Programme P4

% Génération d'un signal exponentiel complexe

clf ;

c = -(1/12)+(pi/6)*i;

k = 2;

n = 0:100;

x = k*exp(c*n);

subplot(2,1,1);

stem(n,real(x));

xlabel ('Temps indexé en n'); ylabel('Amplitude');

title('Partie réelle') ;

subplot(2,1,2);

stem(n,imag(x));
```

xlabel (‘Temps indexé en n’) ; ylabel(‘Amplitude’) ;

title (‘Partie imaginaire’) ;

****

Le programme P5 génère lui une séquence exponentielle à valeurs réelles à partir de l’expression suivante :

$X[n]=A.\alpha^n$

****

```
% Programme P5
% Génération d'une séquence exponentielle à valeurs réelles
n = 0 : 35;  a = 1.2;  k = 0.2;
x = k*a.^n;
stem(n,x)
xlabel (‘Temps indexé en n’); ylabel(‘Amplitude’);
```

****

4. Génération des signaux aléatoires

Un signal aléatoire de longueur N avec une distribution uniforme dans l’intervalle [0 1] peut être généré par la commande de MATLAB suivante :

```
x = rand(1,N) ;
```

De même, u signal aléatoire x[n] de longueur N avec une distribution normalisée à moyenne nulle et variance unité peu être généré en utilisant la commande suivante de MATLAB :

```
x = randn (1,N) ;
```


Application :

Générer et tracer un signal aléatoire de longueur 100 tels que ces éléments sont uniformément distribués dans l’intervalle [-2,2].

5. Génération des signaux complexes

Des signaux complexes peuvent être générés en utilisant les opérations faites pour les signaux élémentaires.

Par exemple un signal de modulation d’amplitude peut être généré par la modulation d’un signal sinusoïdal de haute fréquence $x_H[n] = \cos(\omega_H n)$ avec un signal sinusoïdal de basse fréquence $x_B[n] = \cos(\omega_B n)$. Le signal résultant y[n] a la forme suivante :

$y[n] = A(1+m\, x_B[n])\, x_H[n] = A(1 + m \cos(\omega_B n)) \cos(\omega_H n)$

avec m, appelé facteur de modulation, est choisi pour s’assurer que l’expression $(1+m\, x_B[n])$ est positive pour toutes les valeurs de n.

Le programme P6 est utilisé pour générer un signal modulé en amplitude.

****

```
% Programme P6
% Génération d’un signal modulé en amplitude
clf ;
n = 0 :100 ;
m = 0.4 ; fH = 0.1 ; fL = 0.01 ;
xH = sin(2*pi*fH*n);
xL = sin(2*pi*fL*n);
y = (1+m*xL).*xH;
```

stem(n,y); grid;

xlabel('Temps indexé en n'); ylabel('Amplitude');

****

Application :

Générer et tracer un signal de modulation d'amplitude de fréquence fH=0.08 et fL=0.04 et de facteur de modulation m=0.5. Interpréter.

TP2 (SUITE) : CONVOLUTION DES SIGNAUX

Il s'agit dans cette manipulation d'étudier la convolution sur quelques exemples de signaux. On va e situer dans le cas d'un système linéaire et invariant dans le temps, c'es-à-dire que le signal de sortie de ce système se présente comme le résultat du produit de convolution entre le signal d'entrée x(t) et sa réponse impulsionnelle h(t).

Manipulation

1. Lancer MATLAB.
2. Exécuter le programme 'convolution-demo'.
3. A l'aide de cette interface générer le signal x(t)=u(t)-u(t-2), avec u(t) le signal échelon unité. Pour cela utiliser le bouton 'choisir x(t)'.
4. Générer le signal h(t)=x(t). Pour cela utiliser le bouton 'choisir h(t)'.
5. Sélectionner ensuite le signal à faire déplacer, soit 'déplacer x(t)', soit 'déplacer h(t)'.
6. Utiliser la sourie pour faire déplacer l'instant de translation 't' sur l'axe 'τ'.
7. Interpréter les signaux obtenus au fur et à mesure.
8. Vérifier le résultat obtenu graphiquement par rapport à celui calculé théoriquement.
9. Changer maintenant le signal x(t) par x(t)=u(t)-u(t-5).
10. Interpréter le résultat de la convolution obtenu.
11. Refaire la même manipe ave les signaux suivants :

$x(t) = u(t) - u(t-5) eth(t) = e^{-1/2}\left[u(t) - u(t-10)\right]$

$x(t) = \delta(t) eth(t) = \cos(0,2\pi t) w(t) avec w(t) = [u(t) - u(t-10)]$

$x(t) = \delta(t-5) eth(t) = \cos(0,2\pi t) w(t) avec w(t) = [u(t) - u(t-10)]$

$x(t) = \sin(\pi t) w(t) eth(t) = \delta(t-6)$ et cette fois-ci faire déplacer x(t).

TP3 : INTRODUCTION À L'ANALYSE DE FOURIER

1. Série de Fourier

Il s'agit dans cette première manipe de voire comment calculer l'amplitude et la phase des 20 premiers harmoniques du signal carré x(t) périodique de période T=1s. Ce signal vaut 1 sur 25% de la période et 0 sur le reste de la période.

- Utiliser le programme suivant pour générer le signal x(t) :

```
**************************************************************************
T0=1/1024 ; % période d'échantillonnage
t=0 :T0 :1-T0 ; % discrétisation de l'axe temporel
uns=ones(1,256) ; %25 pour cent de la période vaut 1
zer=zeros(1,1024-256) ;% le reste de la période vaut 0
x=[uns,zer] ; % construire le signal x(t)
plot(t,x) % tracer le signal
axis([0 1 -1 2])
**************************************************************************
```

- Maintenant que nous avons généré le signal temporel x(t), nous allons générer les fonctions exponentielles : e^{-jkw0t} qui se trouve dans l'intégrale de Fourier. Dans notre cas la période T est égale à 1s de sorte que la pulsation fondamentale $\omega_0=2\pi$.

Le programme suivant montre comment générer avec MATLAB les 20 premières fonctions exponentielles :

```
**************************************************************************
w0=2*pi ;
for k=1 :20
        u(k, : )=exp(-j*k*w0*t) ;
end
**************************************************************************
```


- La variable « u » est une matrice de 20 lignes (nombre d'harmoniques) et 1024 colonnes (nombre d'échantillons dans le temps). On peut vérifier les tracés des parties réelles de la première et seconde exponentielle par la syntaxe suivante :

```
**************************************************************************
plot(t,real(u(1, : )))
plot(t,real(u(2, : )))
**************************************************************************
```

- On a maintenant tous les éléments pour calculer les coefficients de Fourier du signal x(t). Puisque l'intégrale est la limite d'une somme.

```
**************************************************************************
C0=sum(x)*T0 ;
for k=1 :20
        X(k)=sum(x.*u(k, : )*T0 ;
end
**************************************************************************
```

La 1ère ligne calcule la composante continue C0. Il s'agit simplement d'intégrer x(t) sur une période. La boucle k=1 :20 calcule successivement les coefficients X(k) (20 premières harmoniques). La somme « sum » est l'approximation de l'intégrale, le symbole « .* » en MATLAB signifie la multiplication point par point de deux fonctions, et l'élément T0 est l'élément différentiel de l'intégration.

- Pour afficher les spectres d'amplitude et de phase, on utilise les fonctions « abs » et « angle » de MATLAB :

```
**************************************************************************
A=abs(X) ;
subplot(2,1,1) ; stem(0 :20,[C0 A])
P=angle(X)
subplot(2,1,2); stem(0:20,[angle(C0) P])
**************************************************************************
```

Sur la 1ère ligne, on calcule le spectre d'amplitude à partir du spectre complexe X. Sur la 2ème ligne, on affiche le spectre d'amplitude (incluant la composante continue C0 à k=0) avec la fonction « stem ». Cette fonction permet un affichage en bâtonnets, contrairement à la fonction « plot » qui trace une ligne reliant les points du signal. La fonction « subplot » permet de tracer plusieurs courbes avec différents axes sur la même figure. Les crochets ici [C0 A] indique que l'on forme un vecteur en concaténant la composante C0 aux amplitudes de « A ». La même opération est faite pour le spectre de phase.

- Finalement, on montre ici comment faire pour reconstituer le signal d'origine x(t) à partir des 20 premières harmoniques (incluant C0) :

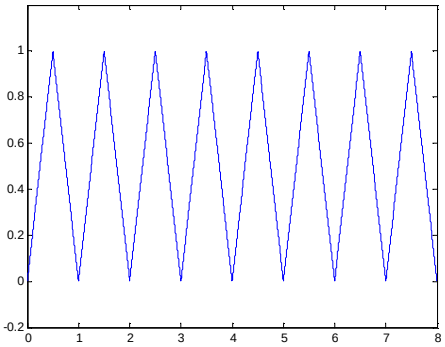
```
**************************************************************************
y=C0 ;
for k=1 :20
        y=y+2*A(k)*cos(k*w0*t+P(k)) ;
end
plot(t,y,'b',t,x,'r')
**************************************************************************
```

Dans ce code, on a utilisé la propriété des séries de Fourier pour les signaux réels qui remplace les exponentielles en cosinus. La dernière ligne trace le signal « y » reconstruit à partir des harmoniques en bleu et le signal d'origine x(t) en rouge.

- On peut vérifier l'apport du nombre des harmoniques sur l'approximation du signal d'origine x(t) en faisant varier le nombre des harmoniques entre 5 et 30 par pas de 5 par exemple.

Application :

Calculer et représenter les spectres d'amplitude et de phase (20 premières harmoniques) du signal triangulaire ci-dessous. Afficher la reconstruction de ce signal à partir de ces harmoniques en faisant varier le nombre des harmoniques entre 5 et 30 par pas de 5.



2. Transformée de Fourier

On considère le signal carré de l'exemple précédent. La période fondamentale est T=1s. Les coefficients de Fourier X(k) sont calculés par la formule suivante :

$$X(k) = \frac{1}{T}\int_0^T x(t)e^{-jk\omega_0 t}\,dt$$

- Représenter le module des 20 premiers coefficients spectraux de ce signal.
- Que se passe-t-il maintenant si on double la période du signal qui devient T=2 ?
- Même questions pour T=4 et T=8.
- Que se passe-t-il si on fait tendre la période du signal vers l'infini ? interpréter ;

TP 3(suite) : ANALYSE ET SYNTHÈSE DE SIGNAUX PÉRIODIQUES

Le but de ce TP est d'établir la correspondance entre le spectre en amplitude d'un signal périodique et son développement en série trigonométrique. Ceci doit permettre dans un second temps de synthétiser différents signaux périodiques à partir du relevé de leurs spectres. Nous allons à présent rappeler quelques généralités sur l'analyse de Fourier. La décomposition en série de Fourier d'un signal périodique de période T0 (ou à durée limitée sur un intervalle T0) s'écrit :

$$x(t) = \sum_{n=-\infty}^{+\infty} x_n \exp(2i\pi n \frac{t}{T_0})$$

x(t) : signal périodique de période T0.

xn : coefficients de Fourier de x(t).

Ces coefficients donnent une représentation en fréquence ou spectrale du signal. Pour passer de la représentation temporelle à la représentation spectrale, on utilise la formule suivante :

$$x_n = \frac{1}{T_0}\int_{T_0} x(t)\exp(-2i\pi n \frac{t}{T_0})dt$$

1/T0 : représente une fréquence appelée fondamentale, et de manière plus générale on nomme $n^{ième}$ harmonique la fréquence correspondant à n/T0.

Prenons une fonction cosinus comme exemple pour calculer le spectre d'un signal périodique à partir des séries de Fourier :

$$x(t) = A\cos(2\pi \frac{t}{T_0})$$

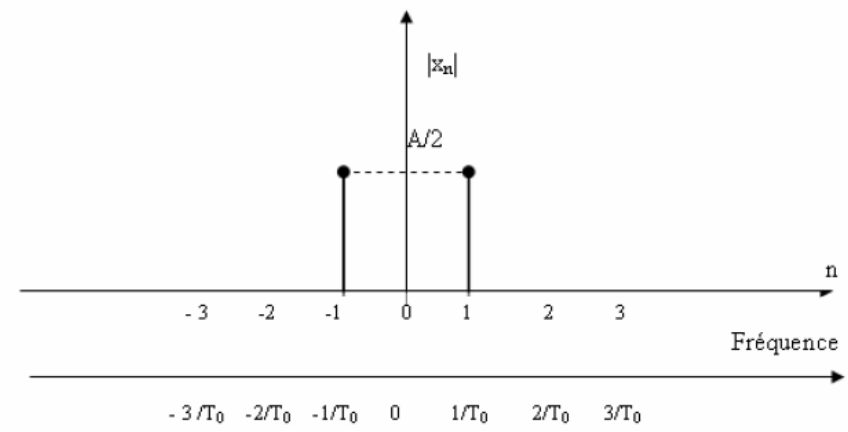
T0 : période

A : amplitude

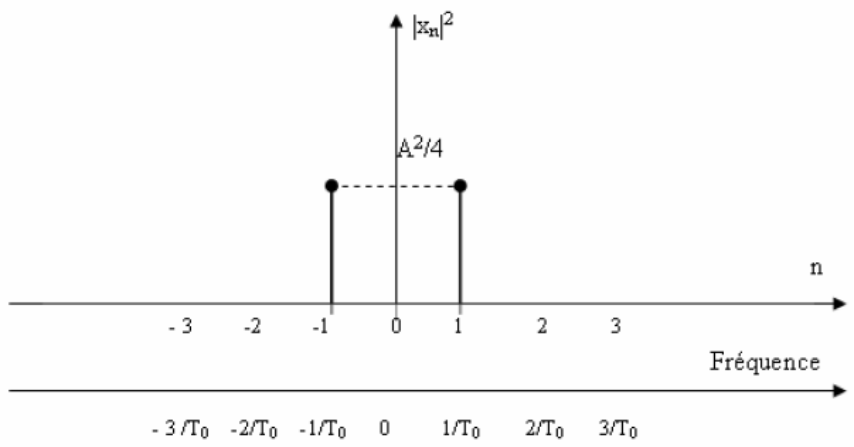
En utilisant l'équation (2) pour calculer les coefficients de Fourier, nous obtenons :

x_n=A/2 pour n=±1 et x_n=0 pour n différent de ± 1

Le spectre en amplitude est donné par la norme de x_n et peut être tracé de la façon suivante :



On remarque que ce signal périodique n'est constitué que d'une fondamentale, il est monochromatique. Le domaine des fréquences négatives n'a aucune signification physique, il est dû aux fonctions exponentielles complexes introduites dans la définition des séries de Fourier. Pour les signaux réels, le spectre en amplitude sera toujours une fonction paire. Les analyseurs de spectre délivrent plus généralement le spectre en puissance qui est représenté ci-dessous dans le cas de la fonction cosinus



Cette représentation permet de faire un calcul très simple de la puissance moyenne totale par le théorème de Parseval :

$$P = \frac{1}{T_0}\int_0^{T_0} x^2(t)dt = \sum_{n=-\infty}^{+\infty} \left|x_n^2\right|$$

Le résultat $P=A^2/2$ est alors immédiat à partir du spectre représenté précédemment.

1. Relations entre la série de Fourier et la décomposition en série trigonométrique

Pour obtenir la relation entre les coefficients x_n et les coefficients en cosinus (a_n) et les coefficients en sinus (b_n), on remplace la fonction exponentielle complexe dans la formule (1) par cosinus + i sinus. Ensuite, on identifie le résultat obtenu avec le développement en série trigonométrique donné ci-dessous :

$$x(t) = a_0 + \sum_{n=-\infty}^{+\infty} a_n \cos(2\pi n\frac{t}{T_0}) + \sum_{n=-\infty}^{+\infty} b_n \sin(2\pi n\frac{t}{T_0})$$

On obtient alors la relation suivante entre les coefficients x_n et les valeurs de a_n et b_n :

$$x_n = \frac{a_n - ib_n}{2} \qquad \text{pour n différent de 0}$$

$$x_0 = a_0 = \frac{1}{T_0}\int_{T_0} x(t)dt$$

Les relations donnant les valeurs de a_n et b_n sont obtenues à partir des équations (6) et (2) :

$$a_n = \frac{2}{T_0}\int_{T_0} x(t)\cos(2\pi n\frac{t}{T_0})dt$$

$$b_n = \frac{2}{T_0}\int_{T_0} x(t)\sin(2\pi n\frac{t}{T_0})dt$$

Les coefficients a_n et b_n ne sont définis que pour des valeurs de n positives. Dans le cas des signaux réels (à partie imaginaire nulle) une relation simple provenant de la propriété de symétrie hermitienne permet de calculer les coefficients x_n pour des indices n négatifs à partir des valeurs de x_n pour n positif : $x_{-n} = x_n^*$

En appliquant ces relations à notre signal sinusoïdal, le seul coefficient différent de 0 est $a_1=A$. Ce résultat s'obtient également de façon très simple dans ce cas particulier en identifiant les équations (3) et (5).

2. Relation entre la transformée de Fourier et les coefficients de Fourier pour les signaux périodiques

Il est également possible de calculer la transformée de Fourier pour un signal périodique de période T0, le résultat obtenu donne en réalité les mêmes informations que les coefficients de Fourier, ceci va être démontré à présent :

$$X(f) = \int_{-\infty}^{+\infty} x(t)\exp(-2i\pi ft)$$
$$= \int_{-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} x_n \exp\left(2i\pi n \frac{t}{T_0}\right)\exp(-2i\pi ft)dt$$
$$= \sum_{n=-\infty}^{+\infty} x_n \int_{-\infty}^{+\infty} \exp\left(-2i\pi\left(f - \frac{n}{T_0}\right)t\right)dt$$
$$= \sum_{n=-\infty}^{+\infty} x_n \delta\left(f - \frac{n}{T_0}\right)$$

3. Propriétés de la série trigonométrique

Il existe de nombreux cas où les coefficients a_n et b_n se calculent très rapidement :

a) pour un signal réel et pair tous les coefficients b_n sont nuls seuls les coefficients an sont différents de zéro.

b) pour un signal réel et impair tous les coefficients a_n sont nuls seuls les coefficients b_n sont non nuls.

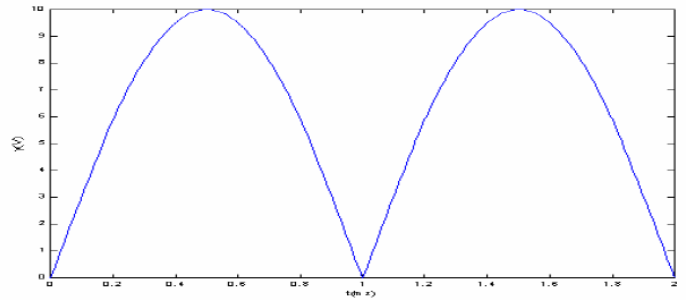
Les valeurs de a_0 ne suivent pas les conditions précédentes, a_0 représente la valeur moyenne du signal.

4. Travail demandé

4.1. Calcul des coefficients en série trigonométrique

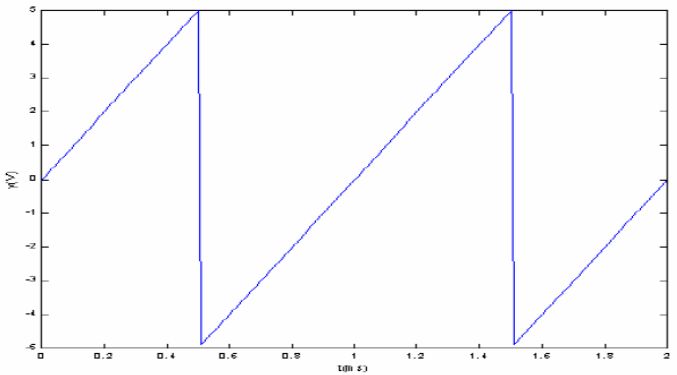
Calculer les coefficients en série trigonométrique des signaux suivants, prenez soins à bien relever les périodes des signaux (attention l'axe des abscisses est donné en millisecondes):

a) Sinusoïde redressée



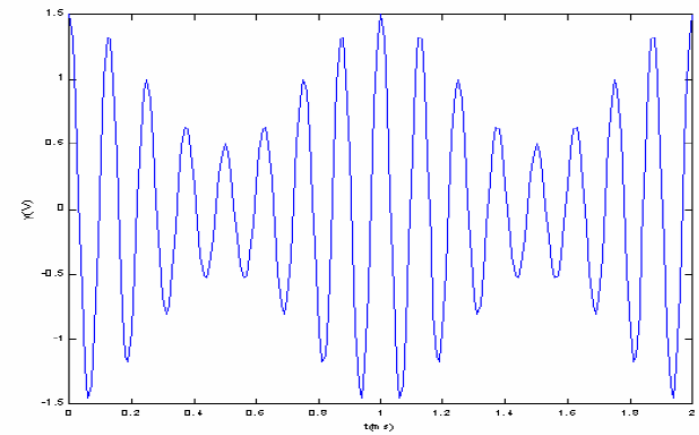
Montrer que : $a_n = \frac{40}{\pi\left(1-(2n)^2\right)}$ $a_0 = \frac{20}{\pi}$ et b_n=0 pour tout n

b) Rampe



Montrer que : $b_n = -10\frac{\cos(\pi n)}{\pi n}$ et a_n=0 pour tout n

c) Signal modulé en amplitude



Ce dernier signal peut être modélisé par l'équation:

$$y(t) = (1 + m\cos(2\pi f_m t))\cos(2\pi f_p t))$$

avec f_m=1 kHz: fréquence du signal modulant, f_p=8kHz: fréquence de la porteuse, m indice de modulation fixé arbitrairement à 0.5 dans le cas présent. Pour trouver la décomposition en série trigonométrique dece signal, il n'est pas nécessaire d'utiliser les formules 7 et 8 pour calculer les coefficients an et b_n, il suffit de développer le produit des fonctions cosinus apparaissant dans l'équation précédente. Le signal y(t) peut alors s'exprimer comme la somme de fonctions cosinus, on réalise donc aussi la décomposition de ce signal en somme de plusieurs sinusoïdes que l'on peut appeler série trigonométrique.

- Déterminer la période du signal modulé ainsi que les coefficients de Fourier.

d) Peigne de Dirac

Le peigne de Dirac a été étudié en cours et nous avons déterminé la transformée de Fourier de ce signal et donc ses différents coefficients de Fourier.

- Calculer les coefficients de Fourier d'un peigne de Dirac de période égale à 1 seconde.

- En déduire les valeurs des coefficients a_0, a_n et b_n.

4.2. Synthèse de signaux périodiques à partir des valeurs de an et bn précédemment obtenues.

Réalisez des fonctions que vous nommerez sinred.m, rampe.m, modam.m et dirac.m pour réaliser la synthèse par série trigonométrique des fonctions sinus redressée, rampe, cosinus modulé en amplitude par un cosinus, et peigne de Dirac. Les arguments d'entrée de ces fonctions seront :

- N : le nombre de sinusoïdes à sommer
- T0 : la période de la fonction.

Il n'y a pas d'argument de sortie, il faut simplement tracer les représentations temporelles et fréquentielles : x(t) et |X(f)|

Pour cela, les signaux seront donc générés à partir d'un vecteur temps t de 1000 points répartis sur l'intervalle $[0, 2T_0]$ grâce à la formule suivante :

$$x(t) = a_0 + \sum_{n=-\infty}^{+\infty} a_n \cos(2\pi n \frac{t}{T_0}) + \sum_{n=-\infty}^{+\infty} b_n \sin(2\pi n \frac{t}{T_0})$$

Pour chaque fonction il faudra calculer les valeurs pour a_n, b_n et a_0. C'est ainsi que l'on pourra synthétiser de façon exacte ou de manière approchée les signaux a),b),c),d) à partir de leurs décompositions en série trigonométrique.

- Pour les signaux a) et b), relevez et commentez l'évolution des ondulations présentes sur les signaux synthétisés x(t) au fur et à mesure que vous augmentez le nombre N de sinusoïdes. En déduire l'origine de ces ondulations qui constituent ce qu'on appelle l'effet Gibbs. Comment expliquez-vous que ces ondulations soient plus importantes sur la rampe que sur le sinus redressé. Pour vos explications, basez-vous sur les spectres des signaux générés.

- Pour le signal c), vérifiez que le signal que vous obtenez par synthèse de plusieurs sinusoïdes correspond exactement à celui représenté dans l'énoncé.

- Pour bien visualiser l'effet Gibbs et son évolution avec le nombre N de sinusoïdes ajoutées, créer dans la fonction Dirac une animation obtenue en ré initialisant dans une boucle while le graphe obtenu pour le tracer de x(t) au fur et à mesure que N augmente. Expliquer en conclusion pour quelles raisons il est impossible de générer un peigne de Dirac ou une impulsion de Dirac.

TP 4 : UTILISATIO N DE LA TFD POUR L'ANALYSE SPECTRALE DE SIGNAUX

1. Introduction

Le but de ce TP est d'interpréter correctement les résultats obtenus par le calcul d'une transformée de Fourier discrète dans le cas de signaux périodiques. Il faudra pouvoir lier ces résultats aux coefficients de Fourier et à la transformée de Fourier du signal échantillonné. A titre d'illustration on traitera des cas simples de signaux périodiques et on verra tout l'intérêt de l'analyse spectrale sur des signaux périodiques bruités. Nous verrons comment le choix de la fréquence d'échantillonnage et du nombre d'échantillons influe sur les spectres obtenus. Pour des rappels sur l'analyse de Fourier de signaux périodiques, reportez vous à l'introduction du TP 2. On peut facilement exprimer la transformée de Fourier X(f) à partir des coefficients de Fourier x_n pour un signal périodique x(t) :

$$X(f) = \sum_{n=-\infty}^{+\infty} x_n \delta\left(f - \frac{n}{T_0}\right)$$

avec T0 : la période de x(t)

et δ : l'impulsion de Dirac ou impulsion unité

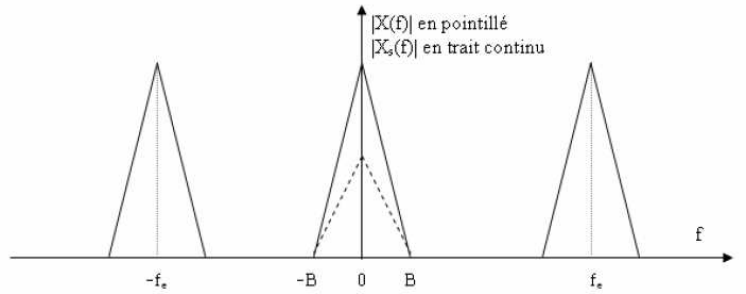
En théorie, échantillonner le signal x(t) revient à le multiplier par un peigne de Dirac $\delta_{Te}(t)$ de période égale à la période d'échantillonnage T_e. On notera $x_s(t)$ le signal échantillonné, il s'écrira en fonction de x(t) sous la forme :

$$x_s(t) = x(t)\delta_{Te}(t) = x(t) \sum_{n=-\infty}^{+\infty} \delta(t - nT_e)$$

La transformée de Fourier de xs(t) (Xs(f)) s'écrira alors comme le produit de convolution de X(f) avec la transformée de Fourier du peigne de Dirac qui est également un peigne de Dirac :

$$X_s(f) = X(f) * f_e \delta_{f_e}(f)$$
$$= f_e \sum_{n=-\infty}^{+\infty} X(f - nf_e)$$

La simplification opérée pour passer de la première à la deuxième ligne de l'équation (3) est due à la propriété d'élément neutre de l'impulsion de Dirac pour la convolution. La figure suivante illustre la relation existant entre X(f) et $X_s(f)$. Attention la fonction Xs(f) étant périodique de période f_e, il est impossible de la représenter dans son intégralité (de -∞ à +∞).



Cette figure montre bien qu'il est nécessaire d'échantillonner le signal à une fréquence f_e supérieure ou égale à 2 fois la bande limitée du signal B de manière à éviter un recouvrement de spectre (théorème de Shannon). On comprend dès lors l'intérêt d'un filtre anti-repliement à placer avant l'échantillonneur pour limiter le spectre du signal. On conçoit également qu'une simple opération de filtrage passe-bas puisse permettre de reconstruire le signal initial à partir de ces échantillons. La figure précédente montre également une propriété importante de $X_s(f)$: la périodicité.

La transformée de Fourier discrète (X[n]) de x(t) est directement liée à la transformée de Fourier du signal échantillonné ($X_s(f)$) par la relation suivante :

$$X[n] = X_s\left(f = \frac{n}{T_0}\right) = \sum_{m=0}^{N-1} x(t_m) \exp\left(-i2\pi \frac{mn}{N}\right)$$

Cette équation s'obtient en simplifiant l'expression de la transformée de Fourier de $x_s(t)$ calculée en

$f=n/T_0$. T_0 désigne ici le temps de mesure du signal, N représente le nombre de points échantillonnés sur cette durée, et $t_m=mt_e$ où t_e est la période d'échantillonnage qui correspond à T_0/N. La majeure partie des problèmes d'analyse spectrale par calcul de transformée de Fourier discrète proviennent de cette limitation de durée du signal qui est évidemment indispensable. En effet limiter la durée de x(t) revient à multiplier x(t) par une fonction porte de largeur égale à T_0. X(f) sera alors convolué par la transformée de Fourier de cette fonction porte qui est un sinus cardinal. Au lieu d'avoir un spectre de raies pour un signal périodique on aura des fonctions sinus cardinales présentant différents lobes (un lobe principal ou central et des lobes secondaires) qui donnent lieu à l'effet Gibbs. Les problèmes de résolution spectrale en analyse par transformée de Fourier discrète sont liés au recouvrement possible de ces lobes.

A titre d'exemple, prenons le cas d'une fonction cosinus de fréquence et d'amplitude unité :

$$x(t) = \cos(2\pi f_1 t) \text{ avec } f_1 = 1Hz$$

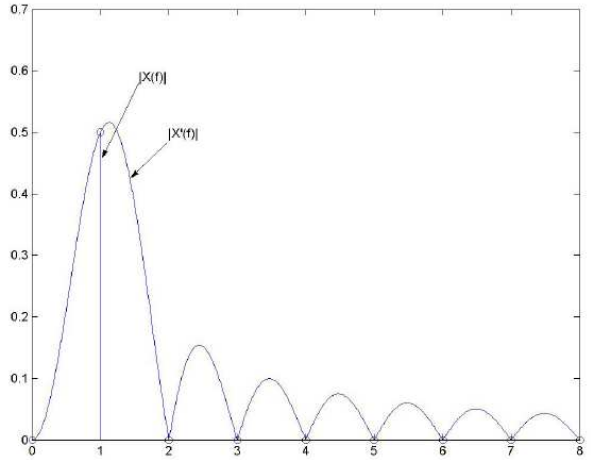
$$X(f) = \frac{1}{2}(\delta(f - f_1) + (\delta(f + f_1)$$

Le spectre en amplitude de ce signal périodique est donc formé d'une impulsion de Dirac dans les fréquences positives située en f_1=1Hz. Pour calculer une TFD sur ce signal, on est amené à limiter le nombre d'échantillons du signal, donc la durée de celui-ci. On note alors x'(t) le signal x(t) limitée à un intervalle de temps allant de 0 à T_0:

$$x'(t) = x(t) r_{T_0}\left(t - \frac{T_0}{2}\right) \qquad \text{avec } T_0=1s$$

$$X'(f) = \frac{T_0}{2} \sin c((f - f_1)T_0) \exp(-i\pi(f - f_1)T_0) + \frac{T_0}{2} \sin c((f + f_1)T_0) \exp(-i\pi(f + f_1)T_0)$$

Le spectre en amplitude du signal limité en durée laisse alors apparaître des pics ou lobes dont le plus important est situé à f_1=1Hz.



Après échantillonnage de ce signal limité à l'intervalle [0,T_0[, le spectre du signal devient périodique de période f_e. Ce spectre est déterminé à partir de la transformée de Fourier du signal échantillonné donné par :

$$X_s(f) = f_e \sum_{k=-\infty}^{+\infty} X'(f - kf_e)$$

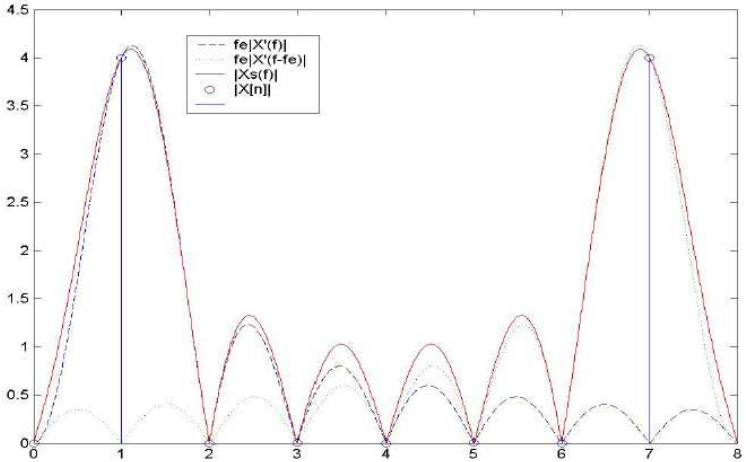
La TFD correspond à une ensemble d'échantillons prélevés sur la transformée de Fourier du signal échantillonné calculés par :

$$X[n] = X_s\left(f = \frac{n}{T_0}\right) = \sum_{m=0}^{N-1} x(t_m)\exp\left(-i2\pi\frac{mn}{N}\right)$$

N correspond aux nombres d'échantillons prélevés sur x(t) sur [0, T0[avec une période d'échantillonnage $t_e = \frac{T_0}{N}$. N correspond aussi au nombre d'échantillons prélevés sur X_s(f) sur [0, f_e[avec un pas appelé résolution spectrale qui vaut : $\Delta f = \frac{1}{T_0}$. Ce dernier résultat découle directement de l'expression de X[n] donnée

précédemment, on vérifie bien qu'en prenant N échantillons distants de $1/T_0$, on couvre bien une gamme de fréquence égale à f_e : $f_e = \frac{N}{T_0}$

Dans notre exemple, nous avons choisi N=8 et f_e=8Hz. Voici ce que nous obtenons dans ce cas au niveau des spectres.

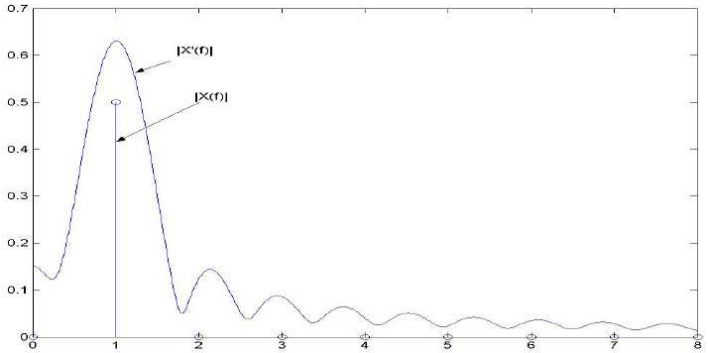


Dans l'exemple que nous venons de traiter, nous avons particulièrement bien choisi la fréquence d'échantillonnage f_e et la durée de mesure T_0.

En effet, f_e respecte le théorème de Shannon $f_e > 2f_1$ et T_0 est un multiple de la période du signal ce qui nous permet d'avoir la fréquence du signal f_1 multiple entier de la résolution spectrale Δf. Lorsque ces deux conditions sur f_e et T_0 sont satisfaites, il est possible d'établir la relation suivante entre les coefficients de Fourier du signal périodique et la transformée de Fourier discrète :

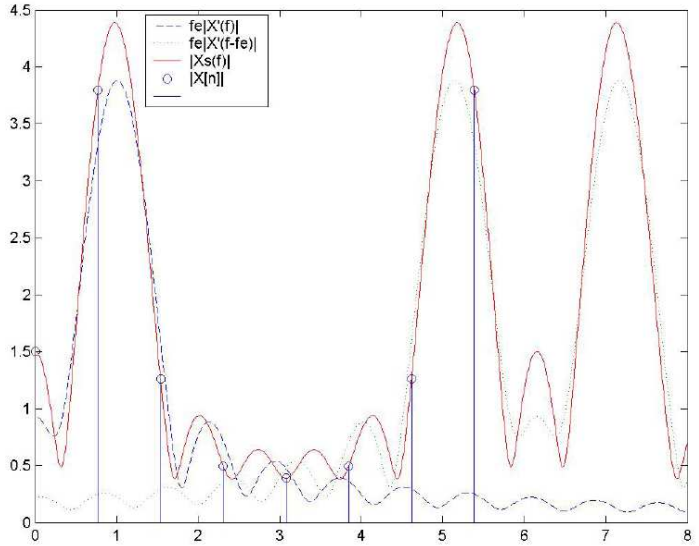
$x_n = \frac{X[n]}{N}$ sur l'intervalle $[0, f_e/2[$. En toute rigueur x_n désigne ici les coefficients de Fourier de la répétition périodique de x'(t), il faut donc nécessairement que T_0 soit égale à la période de x(t) ou alors un multiple entier de celle-ci. Dans le cas contraire, il est impossible d'établir un lien entre les résultats de la TFD et les coefficients de

Fourier ou la transformée de Fourier du signal périodique. C'est ce que nous pouvons observer au travers de l'exemple suivant dans lequel pour le même signal périodique, nous avons choisi une durée de mesure T_0 différente d'un multiple entier de la période.



Pour ces spectres, la durée de mesure a été choisie égale à T_0=1.3s. La fréquence de la fonction cosinus est toujours de 1Hz et 8 échantillons sont prélevés sur la durée T_0.

Dans ce cas, on voit bien que la troncature du signal et le repliement dû à l'échantillonnage modifie fortement les résultats obtenus par TFD et on n'a plus de relation directe entre les coefficients de Fourier et la TFD.



2. Travail demandé

2.1. Analyse d'une fonction cosinus

Créer une fonction tfd qui aura pour but de tracer la représentation temporelle x(t) ainsi que le spectre sous 3 représentations complémentaires :

1) partie réelle du spectre obtenu par calcul de la transformée de Fourier discrète en fonction des fréquences

2) partie imaginaire

3) module

Pour tracer ces 4 graphes, il faut bien sur définir la fonction à étudier et les conditions d'échantillonnage. On choisit d'analyser une fonction cosinus, il faudra fournir son amplitude et sa fréquence ainsi que la fréquence d'échantillonnage f_e et la durée de mesure T_0 comme paramètres d'entrées de votre programme. Si vous voulez par exemple étudier un signal de fréquence 3kHz et d'amplitude 0.2 avec une fréquence

d'échantillonnage 30kHz et une durée de mesure 1ms, il faudra taper tfd(3000,0.2,30000,0.001) pour obtenir les 4 graphes donnant les représentations temporelles et spectrales du signal.

NB : - pour calculer la transformée de Fourier discrète vous pouvez utiliser l'instruction fft.

- utiliser toujours de préférence l'instruction stem plutôt que plot.

- Expliquer comment vous allez calculer le vecteur temps nécessaire à la représentation temporelle du signal.

- Expliquer comment vous allez calculer le vecteur des fréquences nécessaire pour pouvoir représenter le spectre.

- En reprenant l'exemple donné précédemment expliquez comment vous pouvez exploiter le spectre pour retrouver l'amplitude et la fréquence du signal.

- Déterminer les valeurs minimales de f_e et T_0 qui permettent d'obtenir un spectre 'exploitable' pour l'étude d'un signal défini par $x(t) = 5\cos(100\pi t)$, j'entends par spectre exploitable un spectre à partir duquel il est possible de déterminer l'amplitude et la fréquence du signal sans aucune ambiguïté. Il est bien sûr souhaitable de donner les graphes obtenus pour les valeurs de f_e et T_0 choisies.

- Expliquer pourquoi lorsque vous tapez tfd(1000,1,1050,0.04) vous observez une représentation temporelle du signal qui laisse à penser que le signal a une période de 0.02s. Commenter et interpréter également le spectre obtenu.

2.2. Analyse d'une fonction cosinus bruitée

Reprenez la fonction tfd précédente en rajoutant cette fois à la fonction cosinus du bruit obtenu grâce à l'instruction randn qui permet de générer des nombres aléatoires qui suivent une loi gaussienne. Pour réaliser cette addition, il faudra par

exemple sommer à la fonction cosinus un vecteur ab*randn(1,length(t)) où ab désigne l'amplitude du bruit et length(t) la longueur du vecteur temps noté t.

La variable ab devra être fourni comme paramètre d'entrée de la fonction tfd. Au niveau des graphes à tracer, contentez vous à présent de représenter le signal en fonction du temps et le spectre en amplitude (reporter uniquement le module et pas les parties réelles et imaginaires).

Si vous voulez par exemple étudier un signal sinusoïdal de fréquence 3kHz et d'amplitude 0.1 bruité par la fonction randn d'amplitude 0.1 et échantillonné à une fréquence d'échantillonnage 100kHz sur une durée de mesure 1ms, il faudra taper tfd(3000,0.1,0.1,100000,0.001) pour obtenir les 2 graphes donnant les représentations temporelles et spectrales du signal.

- Observer et commenter les graphes obtenus pour le signal pris comme exemple auparavant en choisissant pour ab différentes valeurs : 0.01, 0.05 et 0.1. Essayer dans chaque cas d'exploiter les deux représentations (temporelles et fréquentielles) pour en extraire l'amplitude et la fréquence de la sinusoïde, quelles conclusions peut-on en tirer quant à l'intérêt de l'étude fréquentielle d'un signal.

2.3. Analyse d'une somme de sinusoïdes

Reprenez la fonction tfd précédente en remplaçant cette fois le bruit par une deuxième fonction cosinus. Pour réaliser cette addition, il faudra par exemple sommer à la fonction cosinus de départ un vecteur a2*cos(2*pi*f2*t) où a2 et f2 désignent l'amplitude et la fréquence de la deuxième sinusoïde. Les variables a2 et f2 devront être fournis comme paramètres d'entrée de la fonction tfd.

Si vous voulez par exemple étudier une somme d'un signal sinusoïdal de fréquence 3kHz et d'amplitude 0.1 et d'une fonction cosinus de fréquence 3.3kHz et d'amplitude 0.05 échantillonnée à une fréquence d'échantillonnage 100kHz sur une

durée de mesure 1ms, il faudra taper tfd(3000,0.1,3300,0.05,100000,0.001) pour obtenir les 2 graphes donnant les représentations temporelles et spectrales du signal.

- Observer et commenter les graphes obtenus dans l'exemple précédent. Essayer d'extraire de ces graphes les amplitudes et fréquences des deux sinusoïdes.

- Quel est le paramètre d'échantillonnage à modifier pour pouvoir extraire correctement les amplitudes et fréquences des deux sinusoïdes. Représenter et exploiter les graphes alors obtenus.

- En guise de conclusion générale à ce TP donner les points forts de l'analyse fréquentielle de signaux périodiques par transformée de Fourier discrète.

TP 5 : UTILISATION DE LA TFD POUR L'ESTIMATION D'UNE FONCTION DE CORRÉLATION

1. Introduction

Un signal aléatoire X(t,ω) est défini à chaque instant $t_1,t_2,t_3,\ldots$ par sa loi de probabilité temporelle:

$p(x_1,x_2,x_3,\ldots,t_1,t_2,t_3,\ldots)$. Il existe un grand nombre de lois de probabilité dont la loi gaussienne et la loi uniforme:

loi gaussienne : $$p(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-m_x)^2}{2\sigma^2}\right)$$

loi uniforme : $$p(x)=\begin{cases}\dfrac{1}{x_{\sup}-x_{\inf}} & sur \quad [x_{\inf},x_{\sup}]\\ 0 & ailleurs\end{cases}$$

Il est possible de générer les signaux aléatoires à partir de variables aléatoires qui suivent ces lois de probabilité. On peut ainsi générer des bruits en prenant comme modèle:

$$Y(t,\omega)=Y(t,X)=aX+b$$

avec X une variable aléatoire gaussienne ou uniforme, et a et b des constantes qui permettent de paramétrer la puissance moyenne totale, la variance ou encore la valeur moyenne du signal Y. Un signal aléatoire est dit stationnaire au sens strict si ses propriétés statistiques sont indépendantes de l'origine des temps. Il est stationnaire au second ordre si son moment d'ordre 1 (valeur moyenne statistique) ne dépend pas de l'instant choisi et si le moment mixte d'ordre 2 (l'autocorrélation statistique) ne dépend que de τ: l'écart entre les instants t_1 et t_2 choisis pour relever les ensembles statistiques.

Ces moments sont calculés à partir d'ensembles statistiques formés par des observations du signal en différents instants (t_1, t_2, …). Pour obtenir des résultats significatifs, il est donc nécessaire de constituer des ensembles importants, et pour cela un grand nombre de relevés doit être entrepris. Il est souvent plus facile de travailler sur un seul relevé du signal en fonction du temps et de calculer alors des moments temporels. C'est ce qui est fait habituellement pour des signaux déterministes, pour les signaux aléatoires, ces moments temporels ne donnent des résultats significatifs que si on fait l'hypothèse d'ergodicité, c'est à dire que les moments ou moyennes temporelles correspondent aux moments statistiques. Nous allons présenter les deux principales manières d'estimer numériquement la fonction d'autocorrélation.

2. Estimation numérique de l'autocorrélation

La fonction d'autocorrélation Rx(τ) est calculée en prenant la valeur moyenne de X(t,ω) multiplié par X(t-τ,ω). Pour simplifier les écritures, les signaux numérisés sont le plus souvent exprimés en fonction d'indices prenant des valeurs entières. Par exemple, au lieu d'écrire x(t_m) avec t_m=mt_e (t_e: période d'échantillonnage), on préfère écrire x(m). Ainsi, pour estimer numériquement la fonction d'autocorrélation, on pose simplement le calcul d'une valeur moyenne sur l'ensemble de N échantillons qui constituent x(m):

$$r_1(p) = \frac{1}{N}\sum_{m=p}^{N-1} x(m)x^*(m-p) \qquad \text{pour} \qquad 0 \le p \le N-1$$

Lorsque p tend vers N-1, peu de termes interviennent dans le calcul de la moyenne alors que le terme de normalisation reste constant à 1/N. Cela a pour conséquence d'introduire un biais dans l'estimation: l'autocorrélation est pondérée par une fenêtre triangulaire.

Pour éliminer le biais, un second estimateur peut être défini de la façon suivante:

$$r_2(p) = \frac{1}{N-p}\sum_{m=p}^{N-1} x(m)x^*(m-p) \qquad \text{pour} \quad 0 \le p \le N-1$$

L'avantage de cet estimateur est son absence de biais, mais sa variance devient importante lorsque p tend vers N-1. Les fonctions de corrélation qui vont être calculées par la suite permettront de saisir les notions de biais et de variance d'un estimateur.

3. Utilisation de la TFD pour calculer ces estimées

Pour calculer une fonction de corrélation numérique, il est plus intéressant de passer dans le domaine fréquentiel au moyen de la TFD pour des signaux présentant plus de 80 échantillons car les opérations sont moins nombreuses donc plus rapides à exécuter. L'utilisation de la TFD est basée sur le théorème de Plancherel qui dit que la transformée de Fourier d'un produit de convolution de deux signaux donne le produit simple de la transformée de Fourier des deux signaux : $TF(x(t) * y(t)) = X(f)Y(f)$

Il est possible d'utiliser ce théorème pour estimer la fonction d'autocorrélation car cette dernière renferme une convolution : $R_x(\tau) = x(\tau) * x^*(-\tau)$

Si on applique cette dernière relation au cas d'un signal numérique sur lequel on essaie de calculer l'estimateur biaisé de l'autocorrélation, on obtient :

$$r_1(p) = \frac{1}{N}\sum_{m=p}^{N-1} x(m)x^*(m-p) = \frac{1}{N}(x(p) * x^*(-p))$$

Où * désigne la convolution linéaire définie par: $x(p) * y(p) = \sum_{m=0}^{N-1} x(m)y(p-m)$

Avec y(p -m) = 0 pour (p -m) est en dehors de l'intervalle [0, N -1]

La difficulté pour les signaux numériques est qu'il existe deux définitions de la convolution : la convolution cyclique et la convolution linéaire. La convolution cyclique est définie par la relation :

$x(p) * y(p) = \sum_{m=0}^{N-1} x(m) y(p-m)$ en imposant une périodicité N pour les signaux et x et y de manière à pouvoir calculer les valeurs de y(m-p) lorsque m-p<0 (y(m-p)=y(m-p+N)).

La convolution cyclique présente la propriété suivante :

$TFD(x(p) * y(p)) = TFD(x(p)).TFD(y(p))$, ce qui permet de réaliser cette opération de la manière suivante :

$x(p) * y(p) = TFD^{-1}(TFD(x(p)).TFD(y(p)))$ où TFD^{-1} désigne la TFD inverse

Cette dernière égalité n'étant pas applicable à la convolution linéaire, une manière de contourner le problème pour pouvoir utiliser la TFD afin de calculer une convolution linéaire est de rajouter N zéros aux signaux x et y. La convolution inéaire de deux signaux x et y de N échantillons chacun peut donc être calculée de la manière suivante :

$x(p) * y(p) = TFD^{-1}(X'Y')$ avec X' et Y' correspondant à : X '= TFD(x + N0), Y'= TFD(y + N0). Les deux estimées de l'autocorrélation peuvent donc être calculées de la manière suivante :

$$r_1(p) = \frac{1}{N} TFD^{-1}(X'X'^{*})(p))$$

$$r_2(p) = \frac{1}{N-p} TFD^{-1}(X'X'^{*})(p))$$

Ces deux estimées sont obtenus directement sous Matlab grâce à la fonction xcorr :

[r1,p]=xcorr(x,x,'biased') pour l'estimateur biaisé

[r2,p]=xcorr(x,x,'unbiased') pour l'estimateur non biaisé

r_1 et r_2 sont les deux vecteurs renfermant les deux estimées ; p représente un vecteur qui contiendra les valeurs entières allant de –(N-1) à (N-1) et va servir pour calculer

l'axe des abscisses nécessaire à représenter les autocorrélations. Les valeurs de r(p) pour des valeurs de p négatives sont obtenues grâce à la propriété de périodicité des autocorrélations estimées avec la TFD.

4. Travail demandé

4.1. Autocorrélation d'une sinusoïde

Ecrivez une fonction que vous nommerez autocor.m et dans laquelle vous générez par exemple un signal sinusoïdal d'amplitude 1 de fréquence 0.1Hz échantillonné à une fréquence de 1Hz en 50 points en partant de 0s. Les paramètres d'entrée seront : l'amplitude et la fréquence de la sinusoïde, la fréquence d'échantillonnage et le nombre d'échantillons. On considérera que le vecteur temps part de 0. Pour traiter l'exemple précédent, il faudra taper : autocor(1,0.1,1,50).

- La fonction devra effectuer les calculs et tracés des estimations biasées et non biaisées de ce signal par les différentes formules données auparavant : 4.1, 4.2, 4.3 et 4.4. Vérifier que vous obtenez les mêmes résultats par ces trois méthodes pour les deux estimations. Ne passer pas plus d'une heure sur cette partie car l'essentiel du TP n'est pas là. Par la suite, on se contentera de calculer et tracer uniquement les deux estimées obtenues avec l'instruction xcorr.

- Tracer et comparer les estimations biaisées et non biaisée de l'autocorrélation du signal pris comme exemple auparavant.

- Exploiter ces tracés pour retrouver les caractéristiques du signal, à savoir la puissance moyenne totale et la fréquence.

- Sachant que la puissance moyenne totale est donnée par la fonction d'autocorrélation en zéro, donnez l'expression de l'estimée numérique de la puissance moyenne totale à partir des estimateurs biaisé et non biaisé de l'autocorrélation. Calculer l'estimée la puissance moyenne totale du signal et confronter votre résultat à la théorie.

4.2. Autocorrélation d'un bruit blanc

Sous Matlab comme dans la plupart des langages de programmation, il existe un générateur de nombres aléatoires qui permet de modéliser un bruit échantillonné. Matlab propose deux types de générateur: rand et randn.

- En comparant les histogrammes obtenus pour ces deux bruits en tapant hist(rand(1,1000)) et hist(randn(1,1000)) donnez les différences essentielles des deux générateurs.

- Calculer l'estimée de la puissance moyenne totale du signal obtenu à partir de l'instruction randn(1,1000).

- Calculer également l'estimée de la valeur moyenne du même signal.

- Comment à partir d'un ensemble de variables aléatoires qui suivent une loi de distribution gaussienne de moyenne nulle et de variance 1 est-il possible de générer un signal aléatoire de moyenne a et de variance b ?

- Remplacer dans la fonction autocor le signal sinusoïdal par un bruit blanc gaussien. Les paramètres à fournir seront la valeur moyenne, la variance du bruit ainsi que le nombre d'échantillons. Par exemple, pour générer 100 échantillons d'un bruit blanc gaussien de valeur moyenne 2 et de variance 3, il faudra taper : autocor(2,3,100). La fréquence d'échantillonnage sera prise égale à 1Hz dans tous les cas.

- Tracer et comparer les estimations biaisées et non biaisée de son autocorrélation pour l'exemple choisi auparavant.

- Expliquer comment utiliser ces tracés pour extraire la valeur moyenne, la puissance moyenne totale et la valeur efficace du signal.

- Recommencer avec 1000 échantillons, commenter l'évolution des résultats.

4.3. Autocorrélation d'un sinus bruité

Rajoutez à présent dans la fonction autocor.m des instructions pour générer et tracer un sinus bruité par un bruit blanc gaussien centré. On prendra comme exemple un sinus d'amplitude 1 et de fréquence 0.1Hz constitué de 500 échantillons relevés à une fréquence de 1Hz.

- Sachant que l'on désire un rapport signal sur bruit de 0dB, en déduire la variance du bruit blanc.

- Commenter le tracé du signal bruité, peut-on déterminer l'amplitude et la fréquence de la fonction sinus ?

- Tracer et commenter les estimations biaisées et non biaisées de son autocorrélation

- Exploiter ces tracés afin de retrouver la puissance moyenne totale du bruit, du sinus et la fréquence du sinus.