TD de Traitement de Signal

Hugues Garnier
Mayank Shekar Jha

Décembre 2018

Table des matières

Introduction

MATLAB, pour MATrix LABoratory, est un logiciel de calcul numérique, qui permet de traiter des problèmes complexes dans de nombreux domaines scientifiques, d'exploiter et de visualiser des donnèes en 1D/2D/3D, ..., dans un environnement interactif.

Il est en particulier couramment utilisé par les scientifiques et les ingénieurs, pour réaliser les différentes tâches de traitement du signal.

De nombreuses fonctions sont disponibles dans la boîte à outils *Signal Processing* que nous exploiterons lors de ces séances de TD.

La boîte à outils *Signal Processing* comprend des fonctions et des interfaces graphiques (Apps) pour générer, visualiser, transformer, filtrer des signaux. Des algorithmes sont mis à votre disposition pour :

- approcher le spectre de signaux par transformée de Fourier discrète ;
- estimer les spectres de puissance de signaux aléatoires ;
- concevoir et analyser des filtres ;
- estimer des modèles paramétriques de signaux aléatoires ;
- réaliser une prévision de séries temporelles.

Cette boîte à outils permet donc d'analyser et de comparer des signaux dans les domaines temporel, fréquentiel et temps-fréquence, identifier des motifs et des tendances, extraire des caractéristiques, et développer et valider des algorithmes personnalisés afin d'obtenir des informations sur vos données.

Vous allez réaliser des traitements sur des données audio. Apportez vos écouteurs afin de pouvoir entendre l'effet de vos traitements numériques.

TD 1

Jeux de mots et série de Fourier

Objectifs

— Découvrir et utiliser l'interface interactive **Live Editor**
— Se rappeler des objets manipulés sous Matlab
— Segmenter les mots d'une phrase puis les réorganiser pour générer une autre phrase
— Reconstruire un signal créneau à partir de sa décomposition en série de Fourier

Téléchargez le fichier TD1.zip sur le site web du cours.

1.1 Découverte de Live Editor

Depuis 2016, Matlab intègre une interface interactive, baptisée **Live Editor** qui permet d'écrire, d'exécuter et de modifier un programme Matlab (voir Figure 1.1). Les fichiers créés avec cet éditeur interactif sont sauvegardés avec l'extension .mlx.

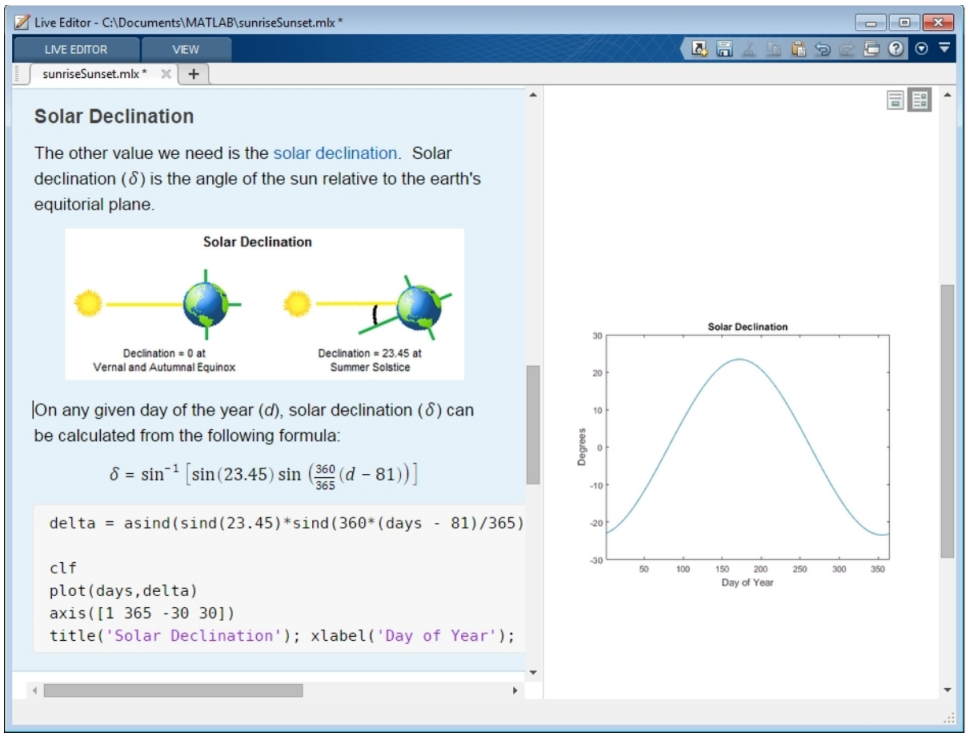


Figure 1.1 – L'interface interactive **Live Editor** de Matlab

Ce nouvel éditeur interactif facilite l'analyse et l'exploration d'un programme en offrant la possibilité d'y intégrer du texte, des hyperliens, des équations mathématiques à l'aide de com-

mandes latex, des schémas ou figures.... A partir du fichier développé, il est ensuite facile de convertir le document en .pdf ou .html, lisible et partageable pour d'autres.
Nous allons découvrir et exploiter cette interface interactive qui permet donc de faire fonctionner Matlab «en ligne» lors des séances de TD.
Visionnez la version courte du webinar de présentation de cette interface interactive disponible à partir du lien suivant :

— `https://fr.mathworks.com/videos/using-the-live-editor-117940.html` (version courte)

— `https://fr.mathworks.com/videos/introducing-the-matlab-live-editor-119100.html` (version longue)

1.2 Rappels : objets manipulés sous MATLAB

1.2.1 Matrice, vecteur, scalaire

Sous Matlab, on travaille essentiellement avec un seul type d'objet : une **matrice** qui peut contenir des valeurs entières, réelles ou complexes.
On peut retenir que dans MATLAB tout est **matrice**.

Lancez MATLAB. Placez-vous sous votre répertoire de travail. Créez un répertoire *TdS* puis placez-vous sous ce répertoire.
Saisissez les différents exemples proposés dans la suite pour bien comprendre la manipulation des vecteurs et matrices sous Matlab.
Attention si vous copiez les commandes depuis le fichier pdf de l'énoncé de TD, il faudra parfois retaper certains caractères (apostrophe en particulier) qui seront soulignés en rouge dans le fichier .mlx.

Sauvegardez l'ensemble des commandes dans un programme dénommé td1_1.mlx qui débutera ainsi :

Manipulation d'objets sous MATLAB

Fichier td1_1.mlx
Votre prénom nom
Date du jour

```
clear
clc
close all
```

1.2.2 Construction d'un vecteur

On peut définir une liste de nombres en donnant à la suite ses éléments séparés par des espaces ou des virgules. Un vecteur est délimité par des crochets.
Exemple

```
a=[2,4,6,8]
```
ou bien
```
a=[2 4 6 8]
a =
2 4 6 8
```
`a` est considéré comme un vecteur (ligne) comportant une ligne et quatre colonnes : 2 4 6 8

Accès aux éléments d'un vecteur

On peut isoler des éléments d'une liste. On accède à l'élément d'indice i de la liste a avec $a(i)$.
Exemple :
```
a(1)
ans =
2
```
On accède au premier élément du vecteur `a`.
Important : Il faut bien noter que les indices d'un vecteur commence à 1 sous Matlab et non 0.
```
a(0)
??? Subscript indices must either be real positive integers or logicals.
```
La demande d'accès à un élément d'indice nul ou négatif conduira donc toujours à une erreur !
```
a(1:3)
ans =
2 4 6
```
On récupère les trois premiers éléments de a.
```
a(1:2:4)
ans =
2 6
```
On récupère les deux premiers éléments aux indices impairs 1 et 3.
Rappels :

- L'expression `x=x0 :pas :xf` crée un vecteur dont les éléments vont de `x0` à `xf` avec un incrément de `pas` ;
- Lorsqu'on ne donne pas le pas, la valeur du pas est par défaut 1.

Exemple :
```
a(1:4)
a =
2 4 6 8
```

1.2.3 Construction d'une matrice

On peut créer des matrices de nombres avec plusieurs lignes en donnant chaque ligne séparée par un point-virgule `";"` .
Exemple :
```
X=[1 2 3 4;5 6 7 8]
X =
1 2 3 4
```

```
5 6 7 8
```

Accès aux éléments d'une matrice

De même, on peut extraire des parties d'une matrice. Le premier nombre désigne la ligne et le deuxième nombre la colonne où se trouve l'élément :
Exemple :

```
 X(1,2)
ans =
2 (élément à la ligne 1 et à la colonne 2)
 X(2,1)
ans =
5 (élément à la ligne 2 et à la colonne 1)

X(:,1)
ans =
1
5 (tous les éléments de la colonne 1)

 X(1,:)
ans =
1 2 3 4 (tous les éléments de la ligne 1)

 X(1,2:4)
ans =
2 3 4
```

Concaténation de matrices

On peut mettre bout à bout plusieurs matrices. Il faut vérifier que la matrice ajoutée a le même nombre de lignes pour une concaténation en colonnes, où le même nombre de colonnes pour une concaténation en ligne.
Exemple :

```
b=[10 12 14 16] ;
c=[a  b]
ans =
2 4 6 8 10 12 14 16  (mise bout à bout sur la même ligne des 2 vecteurs)

Y=[9 10 11 12]
[X ; Y]
ans =
1 2 3 4
5 6 7 8
9 10 11 12 (mise l'une en dessous de l'autre des 2 matrices)
```

Essayez à présent la commande suivante. Que se passe-t-il ? Expliquez.

```
[X  Y]
```

Ce type d'erreur est classique lorsqu'on débute sous Matlab. Pour obtenir la taille d'une matrice ou d'une liste, il est utile d'exploiter les commandes `size` ou `length` présentées ci-dessous.

Dimension d'un vecteur ou d'une matrice

Pour accéder au nombre d'éléments contenus dans une liste et donc à sa longueur, on utilise la commande `length`. Pour accéder au nombre de lignes et de colonnes d'une matrice, on utilise la commande `size`. Exemple :

```
length(a)
ans =
4 (nombre d'éléments de a)
size(X)
ans =
2 4
size(X,1)
ans =
2 (nombre de lignes de X)
size(X,2)
ans =
4 (nombre de colonnes de X)
```

Opération sur les matrices

Toutes les opérations usuelles sur les matrices sont disponibles (addition, multiplication, transposition etc.). Matlab permet aussi d'appliquer les principaux opérateurs éléments par éléments, il faut alors faire précéder l'opérateur voulu par un point « . .
Créez deux vecteurs :
`x=[1 2 3];`
`y=[4 5 6];`
Observez ce que donnent les opérations suivantes :
`x-y`
`x*y`
`x.*y`
`x∧2`
`x.∧2`
`x./y`

Modifiez à présent `x=[0 1 2 3]`.
`x` et `y` n'ont alors plus la même taille. Que se passe-t-il au niveau des opérations ci-dessus ? Conclusion ?

1.3 Jeux de mots

1.3.1 Enregistrement d'une courte phrase à l'aide de votre smartphone

Si vous n'avez pas de smartphone, vous pouvez passer à la section suivante sinon enregistrez-vous à l'aide de votre téléphone en prononçant lentement l'une des deux phrases suivantes :

— « Rien ne sert de courir, il faut partir à point.

— «J'entends, j'oublie. Je vois, je me souviens. Je fais, je comprends.

Une fois enregistré au format .wav ou mp3, envoyez votre fichier sur votre boîte mail et sauvegardez-le sur votre répertoire de travail. Chargez alors votre phrase dans MATLAB à l'aide de la commande suivante (à adapter si votre fichier est au format wav.) :

```
[x,fs] = audioread('maphrase.mp3');
t = (0:length(x)-1)' * 1/fs;
```

Les vecteurs colonne `x` et `t` contenant les échantillons de voix et les instants d'enregistrement. La variable `fs` contient la valeur de la fréquence d'échantillonnage de la phrase.

1.3.2 Visualisation du signal enregistré

Si vous n'avez pas de smartphone, téléchargez le fichier RienNeSertDeCourir.mat sur le site web du cours. Chargez la phrase dans MATLAB en saisissant la commande

```
load RienNeSertDeCourir;\\
fs=22050;
```

Dans ce fichier, se trouvent les vecteurs colonne `x` et `t` contenant les échantillons de voix et les instants d'enregistrement.

Tracez le signal enregistré en fonction du temps.

```
plot(t,x),
shg
```

La commande `shg` pour SHow Graphic permet de mettre au premier plan la figure après une commande de tracé de type plot, stem, ...

1.3.3 Ecoute du signal enregistré

Ecoutez la phrase enregistrée en saisissant la commande :

```
sound(x, fs)
```

Cette commande permet d'écouter la phrase à sa fréquence d'échantillonnage d'enregistrement. Ecoutez la phrase en modifiant la fréquence d'échantillonnage à double ou deux fois plus petite pour me/vous faire parler comme comme « Terminator » ou « Donald Duck ».
Modifier la fréquence d'échantillonnage revient à appliquer un changement d'échelle $y(t) =$

$x(at)$ au signal initial. En fonction de la valeur du facteur d'échelle, cela revient à opérer une compression ou une dilatation du spectre initial d'où la version plus grave ou plus aigüe du signal écouté comme illustré sur la figure 1.2 dans le cas d'une fenêtre rectangulaire.

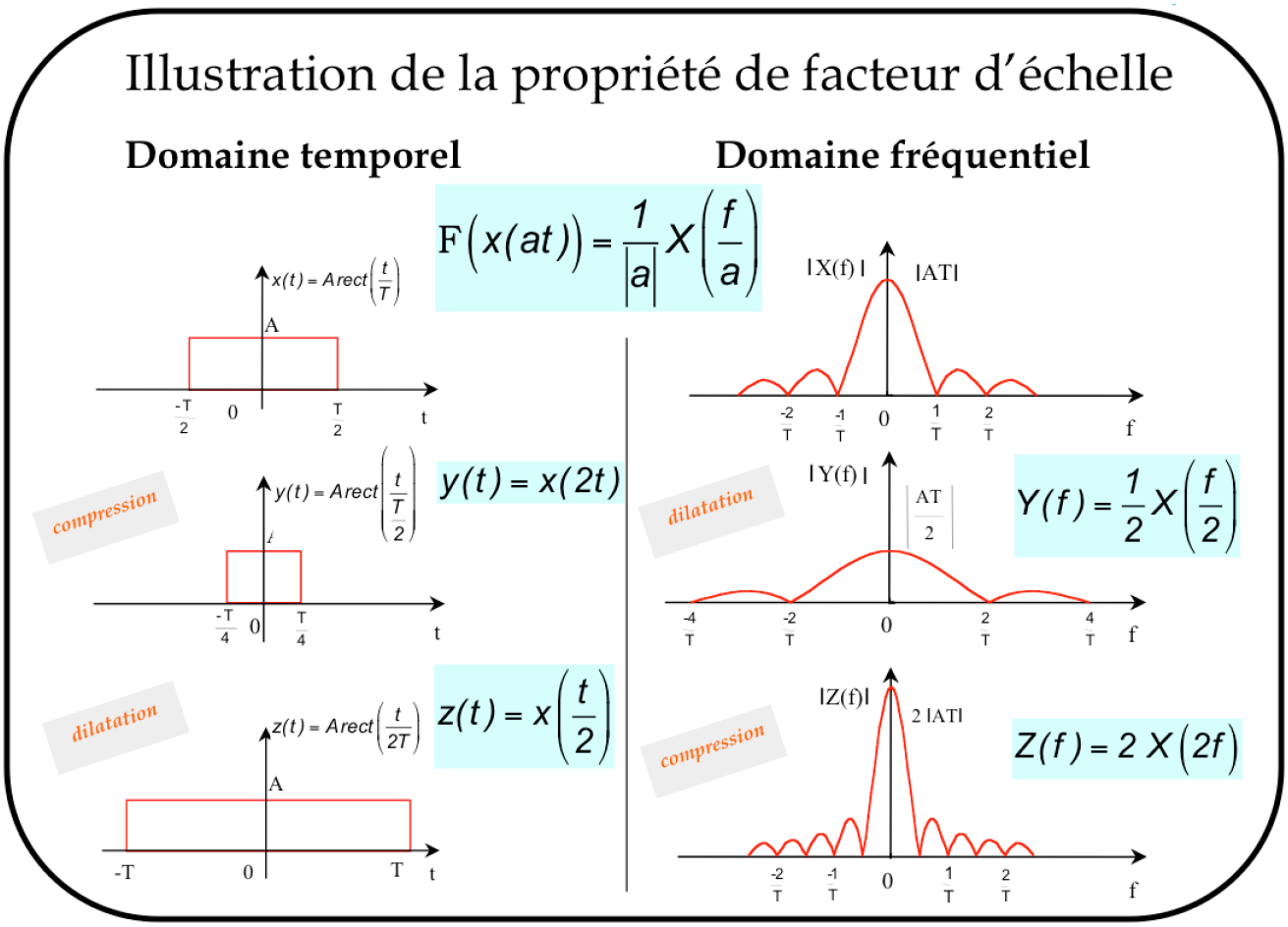


Figure 1.2 – Illustration de l'effet d'un changement d'échelle dans le domaine fréquentiel

Déterminez pour les deux nouvelles fréquences d'échantillonnage la valeur du facteur d'échelle a.

1.3.4 Segmentation de la phrase enregistrée

Tracez le signal en fonction des indices du vecteur x

```
plot(x),shg
```

Essayez de repérer les indices de début et de fin du début de la phrase «Rien ne sert de ».
Pour segmenter le premier mot, il faut par exemple créer un vecteur `riennesertde` contenant les 38000 premières valeurs du signal enregistré `x`.

```
riennesertde = x(1:38000);
```

N'oubliez pas de mettre un « ; » à la fin de la ligne ci-dessus pour exécuter la commande sans afficher le résultat dans la fenêtre commande de Matlab.
Pour écouter le mot segmenté :

```
sound(riennesertde, 22050);
```

Segmentez la phrase initiale en créant les variables suivantes : `riennesertde`, `courir`, `ilfaut`, `partirapoint`. Certaines parties de la phrase peuvent être plus difficiles que d'autres à segmenter ...
Sauvegardez l'ensemble de vos commandes dans un programme dénommé td1_2.mlx qui débutera ainsi :

Jeux de mots
Fichier td1_2.mlx
Votre prénom nom
Date du jour

```
clear
clc
close all

[x,fs] = audioread('maphrase.mp3');
t= (0:length(x)-1)' * 1/fs);
%load RienNeSertDeCourir;
riennesertde = x(1:...);
courir = x(...:...);
etc
```

1.3.5 Synthèse d'une nouvelle phrase

Réorganisez les mots pour générer la phrase suivante : « Rien ne sert de partir à point, il faut courir ! »
Notez que le signal initial de parole est un vecteur colonne contenant un certain nombre de valeurs (length(x)).
Pour produire un signal réarrangé, vous devez construire un autre vecteur colonne.
Pour écouter «Rien ne sert de courir», il faut mettre bout à bout les 2 vecteurs verticaux contenant le début de la phrase à l'aide de la commande :

```
sound([riennesertde ; courir], fs);
```

Faites valider votre programme `td1.mlx` et la nouvelle phrase synthétisée par l'enseignant.

1.4 Reconstitution d'un signal créneau à partir de sa décomposition en série de Fourier

Soit un signal créneau $s(t)$ de période T_o d'amplitude A représenté sur la figure 1.3.
La décomposition en série de Fourier de ce signal créneau ne comprend que des harmoniques de rang impair et s'écrit sous la forme :

$$s(t) = \sum_{k=0}^{+\infty} \frac{4A}{(2k+1)\pi} \sin\big((2k+1)2\pi f_o t\big) = \frac{4A}{\pi}\sin(2\pi f_o t) + \frac{4A}{3\pi}\sin(2\pi \times 3 f_o t) + \dots$$

Ecrire un programme Matlab creneau.mlx sous Live Editor qui permet de reconstruire un signal créneau (pour $A = 2$ et $T_o = 2$s) sur 3 périodes à partir de sa décomposition en

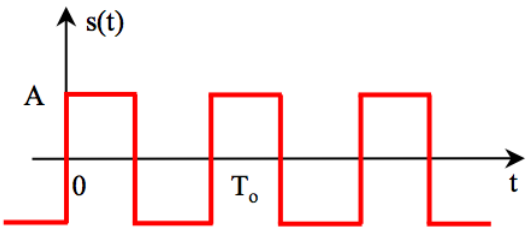


Figure 1.3 – Signal créneau

série de Fourier et de tracer sur une même figure le signal créneau original et le signal reconstruit à partir de n harmoniques. Tracer également les harmoniques successifs sur 3 périodes.

Rappels et conseils :

— Un signal sinusoïdal non retardé est défini par :

$$s(t) = A \sin(2\pi f_o t)$$

Pour le créer sous Matlab, il faudra choisir avec attention la période d'échantillonnage T_e (pas de discrétisation du vecteur temps `t`).

— Tracez dans un premier temps le signal créneau et le signal reconstruit à partir de l'harmonique fondamental puis des 3 premiers harmoniques, avant de considérer le cas général à l'aide d'une boucle `for ... end`. Vous pouvez vous inspirer des commandes ci-dessous :

```
A=2;
To=2 ;
fo=1/To;
fe=100*fo; % Théorème de Shannon fe > 2*fmax
Te=1/fe;
t=0 :Te :3*To ;
cren=A*square(2*pi*fo*t); % Signal creneau
h1=4*A/pi*sin(2*pi*fo*t); % harmonique de rang 1
plot(t,cren,t,h1)
h3=...
plot(t,cren,t,h1+h3)
```

— Jusqu'à quel harmonique, la fréquence d'échantillonnage vérifie-t-elle le théorème de Shannon. Justifier.

Faire valider votre programme creneau.mlx par l'enseignant.

TD 2

Analyse spectrale de signaux audio-numériques

Objectifs

- Découvrir l'interface graphique `signalAnalyzer`
- Déterminer la fréquence d'un diapason
- Vérifier la présence d'un filtre anti-repliement sur votre smartphone
- Générer et analyser le spectre d'une gamme musicale
- Faire l'analyse spectrale d'un signal sinusoïdal sous Matlab à l'aide de la commande `FFT`

Téléchargez le fichier TD2.zip sur le site web du cours.
Vous sauvegarderez l'ensemble de vos commandes dans des programmes

```
Fichier td2_X.mlx
Votre prénom nom
Date du jour

clear
clc
close all

..., etc
```

2.1 Détermination de la fréquence fondamentale d'un diapason

Un diapason est un petit instrument en acier composé d'une tige aux branches en forme de U (voir Figure 2.1) dont on peut se servir pour accorder certains instruments de musique.
Placez-vous dans le répertoire de travail dans lequel vous avez sauvegardé les fichiers contenant le fichier TD2.zip.
Chargez le son du diapason dans MATLAB en saisissant la commande :
`load diapason;`
Dans ce fichier se trouvent le vecteur colonne `y` contenant les échantillons du son produit par le diapason et `fs` la fréquence d'échantillonnage (sampling frequency).
Ecoutez le son émis par le diapason en saisissant la commande :

Figure 2.1 – Diapason

`sound(y, fs);`
Matlab dispose à présent d'un outil graphique d'analyse de signaux `signalAnalyzer` qui permet de visualiser les caractéristiques d'un signal dans les domaines temporel et spectral.
Vous pouvez à tout moment consulter la documentation technique de cet outil graphique en tapant dans la fenêtre Command de MATLAB : >>doc signalAnalyzer

L'interface graphique est accessible en cliquant sur son icône dans le menu Apps ou via la commande ci-dessous à saisir dans la fenêtre command de Matlab :
`signalAnalyzer;`
Une fenêtre graphique s'ouvre.

Sélectionnez la variable `y` dans la fenêtre Workspace browser. Glissez-là vers la fenêtre principale de droite.
L'évolution temporelle est alors tracée en fonction des indices du tableau.
Pour avoir les bonnes échelles sur le tracé temporel (puis fréquentiel), procédez comme suit :
Sélectionnez la variable `y` puis à l'aide d'un clic droit ouvrez la fenêtre TimeValues et spécifiez la fréquence d'échantillonnage du signal (`Sample rate = fs`) via le menu Sample Rate and Start Time.
Vérifiez que le signal est bien tracé sur une durée de 5 s.

2.1.1 Détermination de la fréquence par analyse temporelle

De cette observation dans le domaine temporel, déterminez une valeur approximative de la fréquence fondamentale du signal émis par ce diapason.
Le son produit par ce diapason peut-il être considéré comme un son *pur* produit par un signal sinusoïdal ?
Comment expliquer les variations d'amplitude du signal sur les 5 s enregistrées ?

2.1.2 Détermination de la fréquence par analyse spectrale

L'objectif à présent est d'affiner l'estimation de la fréquence du diapason à l'aide d'une analyse spectrale.
Tracez le spectre d'amplitude du son émis par le diapason.
Sur quelle plage de fréquences le spectre est-il affiché ?
Rappelez le lien entre la valeur maximale de la fréquence affichée et la fréquence d'échantillonnage.
Quelle est l'échelle utilisée sur l'axe des ordonnées ?
A l'aide du menu Data cursors, déterminez d'après le spectre la fréquence fondamentale du son émis par ce diapason.

Faites valider votre analyse par l'enseignant.

2.2 Filtre anti-repliement sur votre smartphone ?

Le fichier RienNeSertDeCourir.mp3 a été enregistré avec un smartphone. Chargez la phrase dans MATLAB en saisissant la commande

```
[x,fs] = audioread('RienNeSertDeCourir.mp3');
```

Dans ce fichier, se trouvent le vecteur colonne `x` contenant les échantillons de voix et `fs` la fréquence d'échantillonnage.

Ecoutez la phrase enregistrée en saisissant la commande :

```
>>sound(x, fs)
```

Utilisez l'outil graphique d'analyse de signaux `signalAnalyzer` pour visualiser le spectre du signal enregistré. Attention à bien préciser la fréquence d'échantillonnage pour avoir un affichage de l'axe des abscisses en secondes.
La numérisation du signal a-t-elle été correctement effectuée ? En particulier, un filtre anti-repliement a-t-il été appliqué au signal analogique avant de le numériser ? Si oui, précisez la fréquence de coupure du filtre et vérifiez sa cohérence par rapport à la fréquence d'échantillonnage.

Répétez l'opération avec le signal que vous avez enregistré avec votre smartphone lors de la première séance de TD.

Ce dernier possède-t-il bien un filtre anti-repliement pour éviter le recouvrement spectral ? Si oui, précisez la fréquence de coupure du filtre et vérifiez sa cohérence par rapport à la fréquence d'échantillonnage.

Répétez l'opération avec l'enregistrement du signal émis par le diapason. Ce dernier possède-t-il bien un filtre pour éviter le repliement spectral ? Si oui, précisez la fréquence de coupure du filtre et vérifiez sa cohérence par rapport à la fréquence d'échantillonnage.

Faites valider vos résultats par l'enseignant.

2.3 Synthèse et analyse spectrale d'une gamme de musique

2.3.1 Synthèse d'une gamme de musique

Les notes de musique produites par un piano peuvent être synthétisées approximativement numériquement. En effet, chaque note peut être considérée comme étant un son *pur* produit par un signal sinusoïdal. La fréquence de la note *La* est par exemple de 440 Hz.
Créez un programme `td2_1`.mlx qui permet de jouer une gamme de musique. La fréquence de chaque note est précisée dans le tableau ci-dessous.

Do1	Ré	Mi	Fa	Sol	La	Si	Do2
262 Hz	294 Hz	330 Hz	349 Hz	392 Hz	440 Hz	494 Hz	523 Hz

Chaque note aura une durée de 1s. La durée de la gamme sera donc de 8s.
La fréquence d'échantillonnage fe sera fixée à 8192 Hz.
La période d'échantillonnage pour la définition du temps sera donc égal à $Te = 1/fe$.
Rappel : la mise bout à bout de vecteurs en ligne X, Y, Z s'effectue à l'aide de la commande : [X Y Z] ;

Sauvegardez la gamme synthétisée dans une variable G.
Ecoutez la gamme générée via la commande :
`sound(G,fe)`

2.3.2 Spectre de la gamme de musique

Utilisez l'outil graphique d'analyse de signaux `signalAnalyzer` pour visualiser le spectre de votre gamme. Observez les 8 fréquences contenues dans la gamme et vérifiez leur valeur numérique à l'aide des curseurs.

Tracez le spectrogramme qui permet de visualiser le contenu fréquentiel du signal au cours du temps (comme le fait une partition de musique) mais la précision sur l'axe des fréquences n'est pas suffisante pour relever précisément les 8 fréquences.

2.4 Approximation du spectre d'un signal sinusoïdal à temps continu par FFT

Le spectre d'un signal à temps continu peut être approché par transformée de Fourier discrète (TFD) ou sa version rapide (Fast Fourier Transform (FFT) - voir annexe A en fin d'énoncé).
Matlab possède la fonction `fft` qui calcule la transformée de Fourier rapide d'un signal à temps discret.
Créez un fichier `td2_1`.mlx et saisissez les commandes ci-dessous qui permettent d'afficher le spectre d'amplitude approché entre [0 ; fe/2] d'un signal sinusoïdal à temps discret de fréquence 1 Hz observé sur une durée de 0.9s et échantillonné à la période T_e=0.1s :

```
A=1;
f0=1;
T_e=0.1;
duree=0.9;
t=0 :T_e :duree;
y=A*sin(2*pi*f0*t);
fe=1/T_e;
N=length(y);
Yfft=fft(y);
m=(0:(N-1)/2);
f=(m)/N*fe;
```

```
% tracé du spectre d'amplitude entre [0;fe/2]
figure
subplot(2,1,1)
if mod(N,2) == 0  \% test si N est pair pour le tracé du spectre
   stem(f,abs(Yfft(1:N/2))/N)
else
   stem(f,abs(Yfft(1:(N+1)/2))/N)
end
% la normalisation par N permet d'avoir les bonnes amplitudes au niveau
% du spectre
ylabel('Abs(S(f))')
% Le tracé du spectre peut s'effectuer de manière continu
% à l'aide de la fonction plot au lieu de stem.
title('Spectre d''amplitude')
```

Visualisez le spectre d'amplitude de ce signal sinusoïdal lorsque celui-ci est défini sur 1 période (cas ci-dessus avec `duree=0.9`), puis sur un nombre entier de périodes du signal (`duree=1.9, 9.9, 99.9`) et enfin sur un nombre non-entier de périodes (`duree=1.7, 9.7, 99.7`). Conclusions.

Les spectres obtenus sont-ils en accord avec la théorie vue en cours et dont les principaux résultats sont rappelés en annexe A ?
Expliquez dans votre programme les résultats observés.

On peut également afficher la densité spectrale de puissance avec une échelle en dB :

$$S(dB) = 20 \times \log_{10} \mid S(f)) \mid$$

Pour plus d'infos sur le lien entre une échelle normale et celle en dB, voir l'aide de la fonction `db` (`help db`).
Voir également le fichier de démo :
`DecibelsFromVoltageAndPowerExample.mlx`

```
 subplot(2,1,2)
if mod(N,2) == 0 % test si N est pair pour le tracé du spectre
plot(f,20*log10(abs(Yfft(1:N/2))/N))
else
plot(f,20*log10(abs(Yfft(1:(N+1)/2))/N))
end
ylabel('Densité spectrale de puissance (en dB)')
xlabel('fréquence (Hz)')
```

Faire valider par l'enseignant.

Annexe A

Rappels : Approximation du spectre d'un signal à temps continu par Transformée de Fourier Discrète (TFD)

On rappelle ici comment approcher le spectre d'un signal à temps continu en utilisant la transformée de Fourier discrète (TFD). La TFD est normalement la transformée à utiliser pour déterminer la représentation spectrale d'un signal à temps discret périodique. Le caractère fini du calcul nécessaire à l'évaluation de la TFD fait que cet outil se trouve particulièrement bien adapté à l'implantation sur un calculateur numérique. C'est la raison pour laquelle la TFD est très souvent utilisée afin de déterminer une version approchée du spectre d'un signal à temps continu quelconque (ce dernier étant théoriquement obtenu à l'aide de la transformée de Fourier à temps continu). Cependant, le calcul du spectre d'un signal à temps continu par TFD engendre quelques approximations qu'il est important de connaître.
Soit $s(t)$ un signal à temps continu et $s(kT_e)$ le signal à temps discret provenant de l'échantillonnage de $s(t)$ avec un pas T_e. Les différentes étapes nécessaires pour pouvoir déterminer une version approchée du spectre de $s(t)$ à partir de $s(kT_e)$ par TFD sont les suivantes :

1. Il faut limiter la durée d'observation du signal à temps continu à un intervalle $T_N = NT_e$ où T_e représente la période d'échantillonnage et N le nombre d'échantillons du signal qui vont servir à calculer les N composantes fréquentielles du spectre du signal à temps continu. Le choix de la durée d'observation du signal est lié à un premier choix à effectuer qui porte sur le nombre d'échantillons N du signal temporel.
2. Il faut ensuite échantillonner le signal à temps continu résultant afin d'obtenir un signal à temps discret. Ceci implique de faire un deuxième choix délicat de la valeur de la période d'échantillonnage T_e qui doit vérifier le théorème de Shannon.
3. Les N composantes fréquentielles du spectre peuvent ensuite être calculées à l'aide de la transformée de Fourier discrète :

$$S(m) = S(m\Delta f) = \sum_{0}^{N-1} s(k)e^{-j2\pi\frac{km}{N}}$$

avec :

— k : indice temporel, $k = 0, 1, ..., N-1$

— m : indice fréquentiel, $m = 0, 1, ..., N-1$

— N : nombre d'échantillons sur l'intervalle d'observation du signal et de composantes d'une période du spectre

— $s(k) = s(kT_e)$: k-ième échantillon temporel du signal

— $S(m)$: m-ième composante spectrale du signal

— T_e : période d'échantillonnage temporelle (intervalle entre 2 échantillons de $s(kT_e)$)

— f_e : fréquence d'échantillonnage $f_e = \frac{1}{T_e}$)

— $\Delta f = \frac{f_e}{N}$: pas fréquentiel aussi appelé précision fréquentielle (intervalle entre 2 raies du spectre).

Le calcul du spectre d'un signal à temps continu par TFD va donc permettre de déterminer N composantes de $S(f)$ régulièrement espacées de Δf dans l'intervalle $[0; fe[$.

Remarques

— Si le signal échantillonné considéré est à valeurs réelles, le spectre d'amplitude est pair et le spectre de phase est impair. On se contentera de représenter $S(m\Delta f)$ sur l'intervalle de fréquence $[0; f_e/2[$, qui est appelé partie ou zone utile du spectre. Cette zone utile est par exemple celle qui est systématiquement affichée sur l'écran d'un oscilloscope numérique présentant un module FFT permettant de déterminer une version approchée du spectre d'un signal à temps continu.

— La précision en fréquence concerne le problème de la mesure de la fréquence d'une seule sinusoïde. Le problème provient du calcul approché par TFD du spectre d'un signal. A partir d'un ensemble de valeurs obtenues par TFD, on ne peut avoir qu'une mesure approchée de la fréquence et son exactitude (sa précision) dépend du nombre de points choisis par effectuer le calcul de TFD. Si N désigne le nombre de points de calcul de la TFD, la précision en fréquence est égale à $1/N$. Pour des signaux échantillonnés à la fréquence f_e (Hz), cela donne une précision de f_e/N.

— La résolution fréquentielle est la capacité à distinguer des fréquences distinctes contenues dans un même signal. Elle est souvent définie comme l'écart minimum en fréquence qu'il faut mettre entre deux sinusoïdes d'amplitude différente pour observer sur le spectre de leur somme un creux de plus de 3 dB entre les deux maxima. La résolution exprimée en Hz est de l'ordre de grandeur de f_e/N qui est l'inverse de la durée d'observation du signal à savoir NT_e.

Choix de T_e et Δf

Le choix de T_e est généralement fixé par la condition d'échantillonnage (Shannon) pour éviter le recouvrement spectral, l'obtention d'un bon pas fréquentiel Δf se fera au prix de l'allongement de la durée NT_e.

Choix de N - fenêtrage

Nous avons vu que l'utilisation de la TFD pour déterminer une version approchée du spectre d'un signal à temps continu implique, lors de la première étape, la limitation de la durée d'observation du signal à temps continu à un intervalle $T_N = NT_e$ où N représente le nombre d'échantillons du signal. Cela revient, d'un point de vue mathématique, à effectuer un fenêtrage sur une durée $T_N = NT_e$ du signal à temps continu et donc à multiplier le signal à temps continu par une fenêtre rectangulaire de largeur NT_e. Ce produit simple dans le domaine temporel entraîne un produit de convolution dans le domaine fréquentiel, autrement dit cela revient à convoluer le spectre du signal à temps continu par le spectre de la fenêtre rectangulaire de largeur NT_e.
Cette opération de convolution a pour effet d'introduire des ondulations dans le spectre obtenu par TFD que l'on appelle fuites spectrales. A ce niveau il est légitime de se demander comment se traduisent ces fuites spectrales dans le domaine temporel ?
Or nous avons vu qu'à un spectre de raies périodique correspond un signal à temps discret et périodique. On en déduit donc que le signal réellement analysé (à partir des N échantillons temporels) est constitué de la juxtaposition de morceaux de signal de durée NT_e. Ainsi on a enregistré exactement une période du signal, la mise bout à bout reconstitue le signal de $-\infty$ à $+\infty$. Cette reconstitution peut engendrer des discontinuités

à chaque raccordement. Ce sont ces discontinuités en début et en fin d'enregistrement qui sont sources de fuites spectrales. Pour éviter cela, il faut diminuer l'influence des extrémités de l'enregistrement. C'est le rôle des fenêtres.
Les différentes étapes nécessaires pour calculer le spectre d'un signal à temps continu par TFD sont résumées sur la figure 9.2 qui considère le cas classique de l'approximation du spectre d'un signal sinusoïdal à temps continu par TFD. On constate que le spectre final obtenu par TFD diffère du résultat attendu qui devrait être une seule raie à la fréquence du signal. Des raies parasites dues au fenêtrage apparaissent.
On vérifie ici le résultat bien connu concernant le calcul du spectre d'un signal à l'aide de la fenêtre naturelle rectangulaire qui est égal au spectre (échantillonné) de la fenêtre considérée centrée autour de la fréquence du signal.
La solution idéale afin de ne pas avoir de fuites spectrales serait d'avoir une fenêtre dont le spectre serait égal à une impulsion de Dirac, qui représente l'élément neutre du produit de convolution continu mais qui est malheureusement impossible à obtenir en pratique.
Plusieurs types de fenêtres d'aire unité et de profil varié dont le module de leur spectre s'approche de l'impulsion ont été proposés afin de réduire l'effet des fuites spectrales.
Parmi les différentes fenêtres, on peut citer les plus courantes : rectangulaire, triangulaire, Hamming, Hanning, Blackman, etc.
Dans tous les cas, l'utilisation d'une fenêtre va améliorer la lisibilité du spectre obtenu par TFD. C'est la raison pour laquelle cette possibilité est offerte à présent sur tous les appareils de mesure tels que les oscilloscopes numériques présentant un module de calcul de TFD via l'implantation de la version rapide (FFT) de l'algorithme.

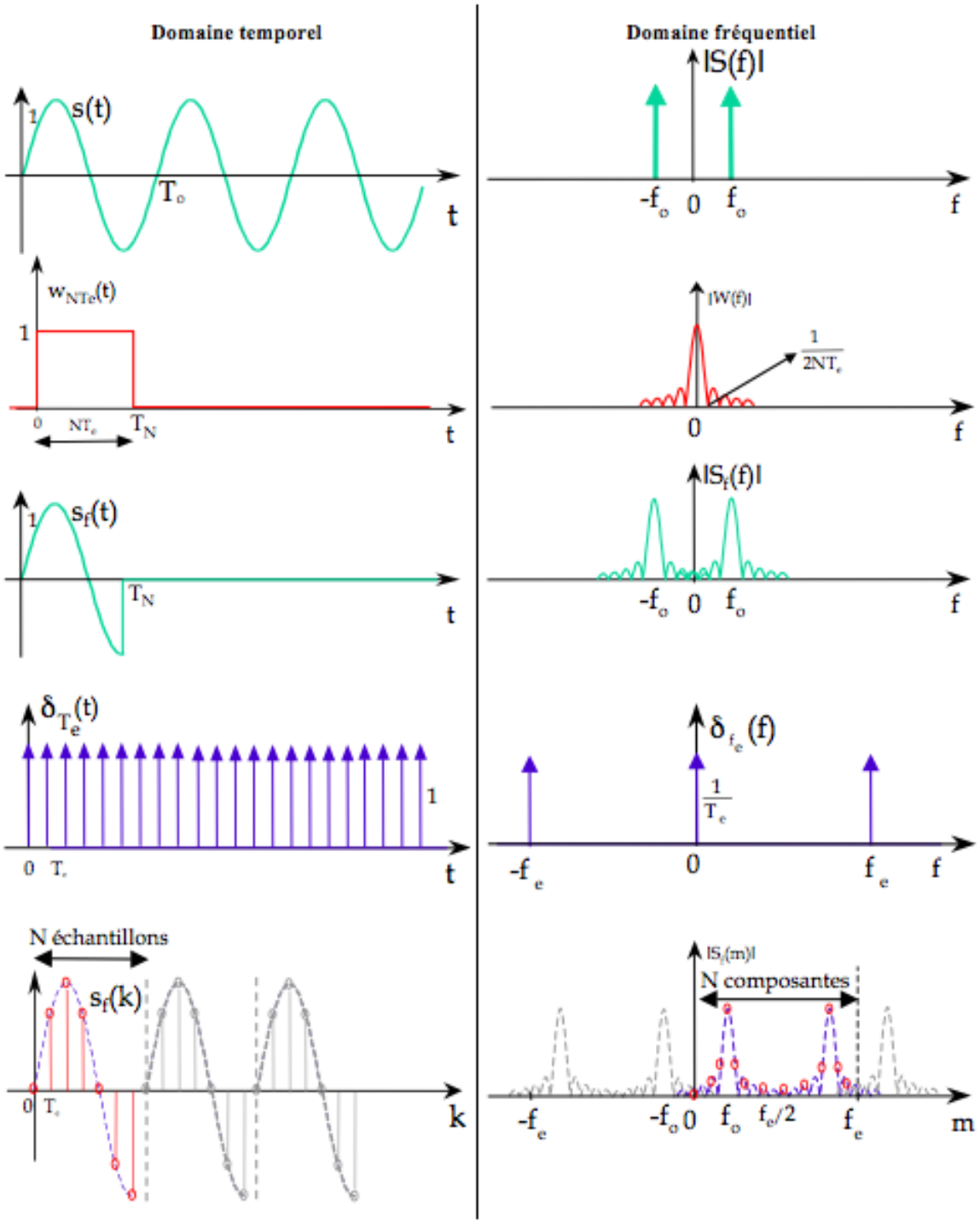


Figure 2.2 – Illustration graphique dans les domaines temporel et fréquentiel des différentes étapes pour pouvoir calculer par TFD une version approchée du spectre d'un signal à temps continu

TD 3

Analyse spectrale du chant d'une baleine bleue & d'une cigale - Détermination d'un numéro masqué & premier filtrage

Objectifs

- Analyser le chant d'une baleine bleue dans le domaine spectral
- Analyser le chant d'une cigale dans le domaine spectral
- Déterminer un numéro de téléphone par analyse spectrale
- Supprimer par filtrage un harmonique parasite dans un morceau de musique

Téléchargez le fichier TD3.zip sur le site web du cours.
Vous sauvegarderez l'ensemble de vos commandes dans des programmes `td3_1.mlx`, et `td3_2.mlx` qui débuteront ainsi :

```
% Fichier td3_X.mlx
% Votre prénom nom
% Date du jour

clear
clc
close all

...
```

3.1 Analyse spectrale du chant d'une baleine bleue

Chargez dans MATLAB l'enregistrement du chant d'une baleine bleue en saisissant la commande :
`ChantBaleine=fullfile(matlabroot,'examples','matlab','bluewhale.au');`
`[x,fs] = audioread(ChantBaleine);`
Relevez la fréquence d'échantillonnage.
Utilisez la fonction `sound` pour écouter le chant de la baleine :
`sound(x,fs);`
Lancez l'analyseur de signaux `SignalAnalyzer`.

Visualisez les 20 secondes enregistrées et affichez le spectre du chant de baleine.

On rappelle que la bande de fréquences audibles par l'oreille humaine s'étale de 20 Hz à 20 kHz. Lorsque la fréquence est faible, le son est grave (de 20 à 200 Hz). On parle de son médium pour une fréquence comprise entre 200 et 1000 Hz et de son aigu lorsque la fréquence est comprise entre 1000 et 15000 Hz.
Relevez la plage de fréquences et en déduire le type de son produit par le chant d'une baleine.

Il est possible de restreindre l'analyse d'un signal sur une durée d'observation donnée en cliquant sur l'icône `Panner` dans le menu `Display`.
Réalisez l'analyse du signal sur la fenêtre de 0 à 5s permettant de se focaliser sur le premier morceau du chant. Observez les harmoniques principaux de ce premier son émis par la baleine. Faites glisser la fenêtre du panner pour observer les caractéristiques des 3 autres sons émis par la baleine.
L'outil `Panner` sera exploité avantageusement dans l'exercice 3.3.

3.2 Analyse spectrale du chant d'une cigale

Chargez dans MATLAB l'enregistrement du chant d'une cigale en saisissant la commande :
```
[y,fe] = audioread('crickets.wav');
```
Relevez la fréquence d'échantillonnage.
Utilisez la fonction `sound` pour écouter le chant de la cigale :
```
sound(x,fe);
```
Lancez l'analyseur de signaux `SignalAnalyzer`.
Visualisez les 2 secondes enregistrées et affichez le spectre du chant de la cigale.

Relevez la plage de fréquences et en déduire le type de son produit par le chant d'une cigale.
Réalisez l'analyse du signal sur la fenêtre de 0 à 0.5s à l'aide du Panner. Faites glisser la fenêtre du panner pour observer les caractéristiques des 3 autres sons émis par la cigale et relevez le caractère périodique du son.

3.3 Détermination d'un numéro de téléphone par analyse spectrale

Un code DTMF (dual-tone multi-frequency) ou FV (Fréquences Vocales) est une combinaison de fréquences utilisée pour la téléphonie filaire traditionnelle utilisée sur le Réseau téléphonique commuté (RTC) (sauf voix sur IP). Ces codes sont émis lors de l'appui sur une touche du clavier téléphonique, et sont utilisés pour la composition des numéros de téléphones et de fax.

Techniquement, chaque touche d'un téléphone correspond à un couple de deux fréquences audibles qui sont émises simultanément. On s'intéresse ici au code DTMF utilisé aux Etats-Unis qui utilise sept fréquences bien distinctes qui permettent de coder les douze touches du clavier téléphonique représenté sur la figure 3.1. Ces sept fréquences sont visibles sur le bord gauche et sur le bas de la figure 3.1.
Elles sont également précisées dans le tableau ci-dessous.

697 Hz	770 Hz	852 Hz	941 Hz	1209 Hz	1336 Hz	1477 Hz

Le son généré par l'appui d'une touche résulte de la somme de deux sinusoïdes aux fréquences associées. Ainsi l'appui de la touche 1 produira le signal suivant :

$$y_1 = \frac{1}{2}\big(\sin(2\pi \times 697 \times t) + \sin(2\pi \times 1209 \times t)\big);$$

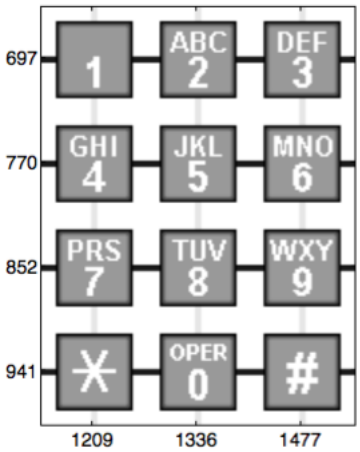


Figure 3.1 – Diapason

La figure 3.2 représente le signal et son spectre d'amplitude lorsqu'on appuie sur la touche "1" du clavier. Les deux fréquences peuvent être clairement identifiées à partir du spectre.

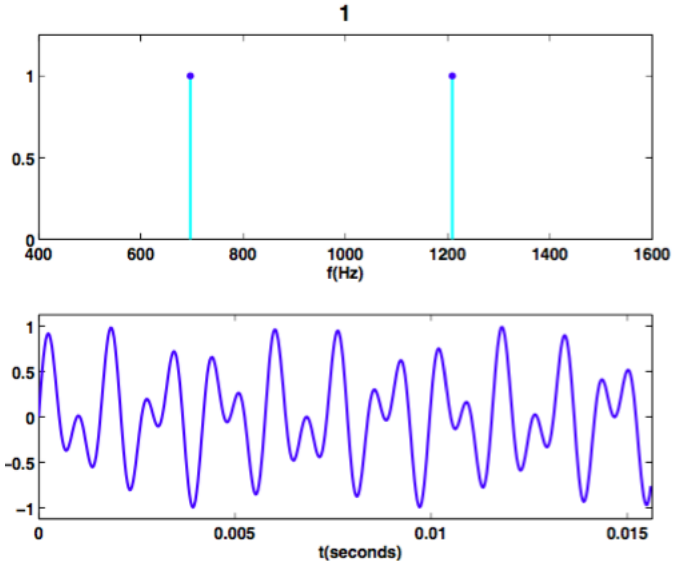


Figure 3.2 – Evolution temporelle et spectre du signal résultant de l'appui de la touche 1 du clavier

Un fichier `numero_tel.mat` contenant un numéro de téléphone américain est disponible sur le site web du cours.
Dans ce fichier se trouvent les vecteurs colonne `y` et `t` contenant les échantillons du numéro composé et les instants d'enregistrement ainsi que la fréquence d'échantillonnage `fe`.
Créez un nouveau fichier `td3_1`.m.
Chargez le numéro de téléphone dans MATLAB en saisissant la commande :

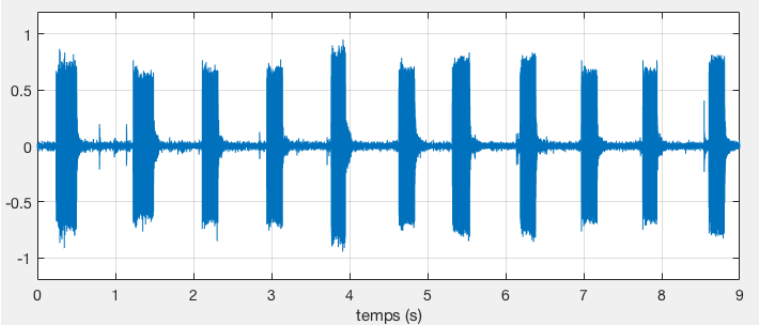


Figure 3.3 – Evolution temporelle du numéro de téléphone composé

```
load numero_tel;
```

Tracez l'évolution temporelle du numéro de téléphone sur les 9 s d'enregistrement disponibles à l'aide de la commande suivante :
`plot(t,y),shg`
Vérifiez que vous obtenez un tracé similaire à celui représenté sur la figure 3.3.
En zoomant, observez la forme de chaque son. Que constatez-vous ?
De combien de chiffres est composé le numéro de téléphone ?
Est-il possible de déterminer le numéro de téléphone d'après l'évolution temporelle du signal composé ?
Ecoutez le numéro composé en saisissant la commande :
`sound(y,fe);`
On peut tracer la représentation temps-fréquence du numéro composé en saisissant la commande :
`spectrogram(y,kaiser(128,18),120,128,fe,'yaxis');`

La figure permet de visualiser le contenu fréquentiel du signal au cours du temps mais la précision sur l'axe des fréquences n'est pas suffisante pour identifier les deux fréquences.

Utilisez l'outil graphique d'analyse de signaux `SignalAnalyzer` pour déterminer le numéro de téléphone (le panner sera en particulier très utile).

A l'aide d'Internet, déterminez le nom de la société à qui appartient le numéro. Trouvez son adresse aux Etats-Unis et utilisez Google Maps pour visualiser la porte d'entrée principale de l'entreprise américaine. Un indice est fourni sur la figure 3.4.

En pratique, la détection de tonalités DTMF peut être effectuée en utilisant différents algorithmes. Pour en savoir plus, parcourez l'article publié en 1996 qui présente l'analyse comparative des performances de trois algorithmes :

`http://ptolemy.eecs.berkeley.edu/papers/96/dtmf_ict/ict96.pdf`

Relevez le nom des trois algorithmes comparés.

Trouvez en surfant sur Internet le code DTMF ou combinaison de fréquences utilisée pour la téléphonie filaire traditionnelle utilisé en France. Générer le numéro de l'accueil de Polytech Nancy. Tracez son évolution temporelle et son spectrogramme à l'aide de l'outil graphique

Figure 3.4 – Indice photographique pour trouver à qui appartient le numéro à identifier

d'analyse de signaux de MATLAB.

Faites valider vos résultats par l'enseignant.

3.4 Traitement d'un extrait musical parasité

Lors de l'enregistrement d'un signal musical, un signal parasite haute fréquence est venu perturber l'enregistrement. L'objectif est donc d'identifier par analyse spectrale du signal enregistré la perturbation puis de la supprimer au travers d'un filtrage afin de retrouver le signal d'origine.

Créez un nouveau fichier `td3_2`.mlx.

Chargez le fichier `extrait_musique.wav`, à l'aide de la fonction Matlab `audioread` :
`[y,fe] = audioread('extrait_musique.wav');`

Quelle est la fréquence d'échantillonnage utilisée pour enregistrer le signal ?

Utilisez la fonction `sound` pour écouter l'extrait musical bruité (attention au volume car cela peut s'avérer désagréable à entendre) :
`sound(y,fe);`

Observez le spectre d'amplitude du signal. Vous pouvez utiliser l'outil `SignalAnalyzer` pour visualiser le spectre. Déterminez la fréquence de l'harmonique parasite.

On propose de supprimer la perturbation par application d'un filtrage passe-bas de Butterworth en étudiant l'influence du choix de l'ordre. Matlab comporte une fonction `butter` (help `butter`) qui fournit les paramètres de la fonction de transfert du filtre à partir de la spécification de l'ordre, de la fréquence de coupure du filtre normalisée et du type de filtre (low, high,...). On pourra s'inspirer de la solution suivante en spécifiant la fréquence coupure souhaitée :

```
fc=????; % fréquence de coupure du filtre
fcn=fc/(fe/2);  % fréquence de coupure du filtre normalisée par rapport à fe/2
n=1; % ordre du filtre
[b,a] = butter(n,fcn,'low');  % Calcul du numérateur et du dénominateur du filtre
```

```
freqz(b,a,512,fe) % tracé de la réponse fréquentielle du filtre
yf=filter(b,a,y);  % produit de convolution du filtre et du signal bruité
sound(yf,fe); % pour écouter l'extrait filtré
```

Renouvelez la procédure en augmentant l'ordre du filtre jusqu'à déterminer l'ordre qui permet de supprimer totalement l'harmonique perturbateur et ainsi écouter le signal musical d'origine.

MATLAB possède la fonction `buttord` qui fournit l'ordre du filtre de Butterworth qui permet de respecter l'atténuation en dB souhaitée. Utilisez cette fonction pour déterminer l'ordre minimal qui permet de ne plus entendre la perturbation dans l'extrait musical.

Faites valider votre filtrage par l'enseignant.

TD 4

The z-transform and its applications in digital filter analysis

Objectives

— Use of the Signal Processing toolbox commands to analyse digital FIR and IIR filters

A knowledge of the z-transform is very important in digital signal processing work, as it is an invaluable tool for representing, analyzing and designing digital filters. The unilateral z-transform of a discrete-time sequence is given by :

$$X(z) = \sum_{k=0}^{\infty} x(k) z^{-k}$$

The z-transforms of common discrete-time input sequences are available, in closed form, and are given in the form of tables. Such tables are particularly useful in finding the inverse z-transform.

Digital filter analysis is quite easy by using the Signal Processing toolbox in Matlab. The objective of this lab is to use this toolbox to analyse digital filters.

4.1 Reminder - Signals in Matlab

This first section will describe how to use Matlab for some basic signal representation and manipulation :

— creating discrete-time signals

— sampling continuous-time signals

— visualizing discrete-time signals

4.1.1 Discrete Signals

Time base : `t = [0.0 0.1 0.2 0.3]`
Signal data : `x = [1.0 3.2 2.0 8.5]`

The central data construct in Matlab is the numeric ***array***, an ordered collection of real or complex numeric data with one or more dimensions. The basic data objects of signal processing (one-dimensional signals or sequences, multichannel signals, and two-dimensional signals) are all naturally suited to array representation.
Matlab represents ordinary one-dimensional sampled data signals, or sequences, as vectors (or arrays). Vectors are 1-by-n or n-by-1 arrays, where n is the number of samples in the sequence.

One way to introduce a sequence into Matlab is to enter it as a list of elements at the command prompt.
The statement
```
x = [1 2 3 4 5]
```
creates a simple five-element real sequence in a row vector. It can be converted to a column vector by taking the transpose :
```
x = [1 2 3 4 5]'
```
Column vectors extend naturally to the multichannel case, where each channel is represented by a column of an array.
Another method for creating vector data is to use the colon operator ':'. Consider a 1-second signal sampled at a sampling frequency $fs = 1000$ Hz. An appropriate time vector would be
```
fs=1000;
Ts=1/fs;
t = 0:Ts:1;
```
where the colon operator ':' creates a 1001-element row vector representing time from zero to one second in steps of one millisecond.
You can also use the `linspace` function to create vector data :
```
N=1000;
t = linspace(0,1,N);
```
creates a vector of 1000 linearly spaced points between 0 and 1.
Enter then :
```
t1 = [0 .1 .2 .3];
t2 = 0:0.1:0.3;
t3 = linspace(0, 0.3, 4);
T = [t1' t2' t3']
```
What does these lines show ?
Enter now :
```
t4 = [0 .1 .2];
T = [t1' t2' t3' t4'];
```
You should get an error message. Explain why ?

4.1.2 Sampling signals

Analog signal sources include electromagnetic, audio, sonar, biomedical and others. Analog signals must be sampled in order to be processed digitally.

Sampling
Let us consider a discrete-time signal denoted as $x(k) = x_a(kT_s)$ sampled from the analog signal $x_a(t)$ with a sampling period of T_s and a sampling frequency of $F_s = 1/T_s$.
Try :
```
fs = 100;
Ts=1/fs;
N = 1000;
stoptime = 9.99;
f0=2;
t = 0:Ts:stoptime;
```

```
x = sin(2*pi*f0*t);
plot(t,x);
xlabel('Time (s)');
```

Signal visualization

— View signal amplitude versus time index

— Functions : `plot, stem, stairs, strips`

— Listen to data : `sound`

Note : the `sound` and `soundsc` commands will not work if your computer hardware is not set up. If that is the case, view the signals instead of listening to them.
Try and compare :

```
t = [0.1 0.2 0.3 0.4];
x = [1.0 8.0 4.5 9.7];
plot(t,x)
figure
stem(t,x)
figure
stairs(t,x)

fs = 1000;
t = 0:1/fs:2;
f = 250 + 240*sin(2*pi*t);
x = sin(2*pi*f.*t);
strips(x,0.25,fs)
sound(x,fs)
figure
plot(t,x)
figure
plot(t(1:200),x(1:200))
```

What does the strips command do ? (See 'help strips'.)

4.2 Digital filters

The Signal Processing Toolbox provides several models for representing digital filters :

— impulse response and convolution sum ;

— difference equations ;

— transfer functions ;

— zero-pole-gain and pole-zero diagram.

4.2.1 Digital filter representation : impulse response and convolution sum

An arbitrary linear time-invariant digital filter can be completely described by its impulse response. The impulse response $h(k)$ is the response of the system to a unit impulse at $k = 0$. In Matlab, vector indices start at 1, so we will define the impulse response as the response of

the system to a unit impulse at $n = 1$. By superposition, the response of the system $s(k)$ to an arbitrary input $e(k)$ can be described as a linear combination of scaled and shifted impulse responses, giving rise to the convolution sum.

$$s(k) = h(k) \star e(k) = \sum_{i=-\infty}^{+\infty} h(i)e(k-i)$$

Try a few simple convolutions :

```
h = [1 2 3]; % impulse response of the filter
stem(h)
e=1; % input is an impulse
s=conv(h,e);
figure
stem(s) % output is the impulse response
e=[1 1 1];
s=conv(h,e);
figure
stem(s)
e=[1 2 3];
s=conv(h,e);
figure
stem(s)
```

4.2.2 Digital filter representation : difference equations

A second way to describe a digital filter is to use the difference equation of the filter. Difference equations describe the input/output behavior of an LTI system directly in terms of signal values

$$\sum_{i=0}^{N} a_i s(k-i) = \sum_{i=0}^{M} b_i e(k-i), \quad a_0 = 1$$

or

$$s(k) = \sum_{i=0}^{M} b_i e(k-i) - \sum_{i=1}^{N} a_i s(k-i)$$

Note the summation limits in the two equations.

A difference equation can be described in Matlab as two vectors :

```
b = [b0 b1 ... bM];
a = [a0 a1 ... aN];
```

When `a` is nonzero (so that `s(k)` depends on its previous values), the system is said to be *recursive*. It will have an *infinite impulse response* (IIR).

When $a_i = 0$ for $i > 1$ and a_0=1, the system is said to be nonrecursive. It will have a finite impulse response (FIR).

Example : Represent the following difference equation as two vectors :

$$s(k) + 0.2s(k-2) = e(k) + 2e(k-1)$$

```
b = [1 2 0];
a = [1 0 0.2];
```
Why is the rightmost zero in the expression for b necessary ?

4.2.3 Digital filter representation : transfer functions

The z-transform transforms a difference equation into a transfer function. The transfer function can be written either in discrete filter form :

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_N z^{-N}}$$

or, by multiplying the numerator and denominator by the highest power of z, in proper form :

$$H(z) = \frac{b_0 z^M + b_1 z^{M-1} + b_2 z^{M-2} + ... + b_M}{a_0 z^N + a_1 z^{N-1} + a_2 z^{N-2} + ... + a_N}$$

Either form can be represented in Matlab by a vector of coefficients for the numerator and denominator :
```
b = [b0 b1 ... bM];
a = [a0 a1 ... aN];
```

Let us consider the previous difference equation

$$s(k) + 0.2s(k-2) = e(k) + 2e(k-1)$$

If we apply the z-transform to both sides of this equation, it comes

$$S(z) + 0.2z^{-2}S(z) = E(z) + 2z^{-1}E(z)$$

The ***discrete filter form*** of the transfer function is

$$H(z) = \frac{S(z)}{E(z)} = \frac{1 + 2z^{-1}}{1 + 0.2z^{-2}}$$

By multiplying the numerator and denominator by z^2, the ***proper form*** of the transfer function is

$$H(z) = \frac{S(z)}{E(z)} = \frac{z^2 + 2z}{z^2 + 0.2}$$

Either form can be represented in Matlab by
```
b = [1 2];
a = [1 0 0.2];
[beq aeq] = eqtflength(b,a)
```
What does the last line do ?

4.2.4 Digital filter representation : zero-pole-gain

If the numerator and denominator of the proper form of a transfer function are factored, the zeros q_i (*i.e.*, the roots of the numerator) and poles p_i (*i.e.*, the roots of the denominator)

become apparent :

$$H(z) = \frac{S(z)}{E(z)} = k\frac{(z-q_0)(z-q_1)...(z-q_{N-1})}{(z-p_0)(z-p_1)...(z-p_{N-1})}$$

The leading coefficient in the numerator, k, is called the gain [1].

Example : Let us consider the following transfer function

$$H(z) = \frac{z^2+2z}{z^2+0.2}$$

The zeros are obtained for :

$$\begin{aligned} z^2+2z &= 0 \\ z(z+2) &= 0 \\ z=0 \quad \text{or} \quad z &= -2 \end{aligned}$$

The poles are obtained for :

$$\begin{aligned} z^2+0.2 &= 0 \\ z &= \pm\sqrt{0.2}j \end{aligned}$$

The gain is equal to :

$$k = 1$$

The location of the zeros and poles of the transfer function determines the response of a digital filter. The Signal Processing Toolbox provides a number of functions to assist in the zero-pole-gain analysis of a system.

`[z,p,k] = tf2zpk(b,a)` finds the zeros `z`, poles `p`, and gain `k` from the coefficients `b` and `a` of a transfer function in discrete filter form.

`[z,p,k] = tf2zp(b,a)` finds the zeros `z`, poles `p`, and gain `k` from the coefficients `b` and `a` of a transfer function in proper form.
There is no difference between `tf2zpk` and `tf2zp` if the lengths of the filter coefficient vectors are first equalized using `eqtflength`.

`zplane(z,p);`
`zplane(b,a);`
both display the zeros and poles of a system. A `"o"` marker is used for the zeros and an `"x"` marker is used for the poles.

The unit circle is also plotted for reference during stability and phase analysis.

Try :

```
b=[1 2 0];
a=[1 0 0.2];
[z,p,k] = tf2zpk(b,a)
zplane(z,p)
```

1. Note that this gain is different from the static gain which is obtained when $z = 1$.

4.3 Digital filter analysis

4.3.1 Impulse response

The impulse response $h(k)$ and its z-transform, *i.e.* the transfer function $H(z)$, completely characterize the response of a LTI system.

For an input $e(k)$ in the time-domain ($E(z)$ in the frequency-domain), the output of the system $s(k)$ in the time-domain ($S(z)$ in the frequency-domain) is given by :

$$s(k) = h(k) \star e(k) \quad \text{(convolution) Time-domain}$$

$$S(z) = H(z)E(z) \quad \text{(product) Frequency-domain}$$

The Signal Processing Toolbox function
`[h,t]=impz(b,a,n,fs)`
computes the response of the digital filter with transfer function coefficients `b` and `a` for a unit impulse $e(k) = \delta(k)$. It computes `n` samples and produces a vector `t` of length `n` so that the samples are spaced `1/fs` units apart.
It returns the response in the column vector `h` and sample times in the column vector `t`. When called with no outputs, it directly plots the response.

Try :
```
b=[1 2 0];
a=[1 0 0.2];
impz(b,a,10,1e3)
```

4.3.2 Frequency response

The frequency response of a stable digital filter is related to the z-transform by

$$H\left(e^{j\omega T_s}\right) = H(z)|_{z=e^{j\omega T_s}}$$

which can be rewritten as

$$H\left(e^{j\omega T_s}\right) = |H\left(e^{j\omega T_s}\right)| \quad e^{j\mathrm{Arg}\left(H\left(e^{j\omega T_s}\right)\right)}$$

from which the magnitude and the phase of the frequency response can be plotted.
In Matlab, the functions `abs` and `angle` can be used to compute the magnitude and the angle of each element in a complex vector.

Frequency response via Matlab
Tedious calculation and plotting of $H\left(e^{j\omega T_s}\right)$ by hand is usually unnecessary if a computer program such as Matlab is available. The Signal Processing toolbox function `freqz` is provided for that purpose. The frequency response can be evaluated for a certain range of ω and then its magnitude and phase can be plotted
`freqz(b,a)`
when called with no output arguments, plots the magnitude and unwrapped phase of the filter in the current figure window.

The following form of the function
`[h,w]=freqz(b,a)`
computes the frequency response of the LTI system with transfer function coefficients `b` and `a`. It returns the frequency response vector `h` and the corresponding angular frequency vector `w`. The vector `w` has values ranging from `0` to the Nyquist rate, which is normalized to π radians per sample by default. The frequency response is calculated using a default of 512 samples.

Example : Let us again consider the transfer function

$$H(z) = \frac{z^2 + 2z}{z^2 + 0.2} = \frac{1 + 2z^{-1}}{1 + 0.2z^{-2}}$$

The frequency response is

$$H\left(e^{j\omega T_s}\right) = \frac{1 + 2e^{-j\omega T_s}}{1 + 0.2e^{-j2\omega T_s}}$$

Matlab could be used to evaluate the frequency response for certain values of ω. The functions `abs` and `angle` could then be used to compute the magnitude and the phase of the frequency response.
Try and compare :

```
Ts=1;
N=512;
w=0:1/N:1-1/N;
H=(1+2*exp(-j*w*Ts))./(1+0.2*exp(-2*j*w*Ts));
figure
subplot(211), plot(w,20*log10(abs(H)))
subplot(212), plot(w,(180/pi)*angle(H))
xlabel('Normalized frequency')
```

The function `freqz` can be used to directly compute the frequency response of the filter from the transfer function coefficients defined in two vectors `b` and `a` `b=[1 2 0];`

```
a=[1 0 0.2];
Hfreqz=freqz(b,a,w);
figure
subplot(211), plot(w,20*log10(abs(Hfreqz)),w,20*log10(abs(H)))
subplot(212), plot(w,(180/pi)*angle(Hfreqz),w,(180/pi)*angle(H))
xlabel('Normalized frequency')
```

Note that `H=freqz(b,a,w)` returns the frequency response vector `h` calculated at the frequencies (in radians per sample) supplied by the vector `w`.

Tips
It is advised to use the `freqz` routine as follows

```
[H,f]=freqz(b,a,512,fs)
subplot(211), plot(f,abs(H))
subplot(212), plot(f,(180/pi)*angle(H))
```

In this syntax, the frequency response is calculated using 512 samples and a sampling frequency specified by the scalar `fs` in Hz. The frequency vector `f` is calculated in Hz and has values ranging from `0` to `fs/2` Hz.

4.4 Filtering a signal

To filter a signal using a system described by transfer function coefficients `b` and `a`, use the Matlab `filter` function.

`s=filter(b,a,e)`
filters the data in vector `e` with the filter described by numerator coefficient vector `b` and denominator coefficient vector `a`.
If `a(1)` is not equal to `1`, `filter` normalizes the filter coefficients by `a(1)`.
If `a(1)` equals `0`, `filter` returns an error. The filter function is implemented in a direct form II transposed filter architecture.
This is the difference equation

$$s(k) = b(1)e(k) + b(2)e(k-1) + \ldots + b(M+1)e(k-M) - a(2)s(k-1) - \ldots - a(N+1)s(k-N)$$

where N is the filter order (the highest degree in the proper form of the transfer function).

Enter the following lines in a program file called `sine.m` to generate a noisy sine wave :

```
fs = 1e4;
f0=262; % Middle C at 262 Hz
tstop=5/f0; t = 0:1/fs:tstop;
s = sin(2*pi*f0*t); % Middle C : noise-free sine at 262 Hz
n = 0.1*randn(size(s));% Noise
sn = s + n; % Noisy sine
figure
plot(t,sn),
b=[.25 .25 .25 .25];
a=[1 0 0 0];
snf1=filter(b,a,sn);
figure
plot(t,sn,t,snf1),
h=impz(b,a);
snf2=conv(h,sn);
figure
plot(t,sn,t,snf2(1:length(t)))
```

In this example a simple low-pass (averaging) filter is applied to the noisy sine signal.

How do the two outputs (`snf1` and `snf2`) compare ? How do the methods (`filter` and `conv`) differ ?

4.5 Exercices

Exercice 1. Consider the so-called 2-point averaging filter given by the following difference equation

$$s(k) = \frac{1}{2}e(k) + \frac{1}{2}e(k-1)$$

Derive the filter transfer function, $H(z)$.
Write a td4_1.mlx file which answers the following questions :

(a) Define the transfer function.
(b) Sketch the impulse response.
(c) Specify the filter type : FIR or IIR.
(d) Plot the poles and the zeros in the z-plane.
(e) Is the filter stable ?
(f) Sketch the amplitude and phase frequency responses in the interval $0 \leq f \leq \frac{f_s}{2}$, where f_s is the sampling frequency in Hz.
(g) Deduce from the frequency responses the type of filtering (low-pass, high-pass, ...) and the cut-off frequency of the digital filter.

Exercice 2. Consider a digital filter given by the following difference equation

$$s(k) = -\frac{1}{2}e(k) + \frac{1}{2}e(k-1)$$

Write a td4_2.mlx file which answers the same questions as in Exercice 1.

Exercice 3. Consider a digital filter given by the following difference equation

$$s(k) = 0.5s(k-1) + e(k)$$

Write a td4_3.mlx file which answers the same questions as in Exercice 1.

Exercice 4. Consider a digital filter given by the following difference equation

$$s(k) = -0.5s(k-1) + e(k)$$

Write a td4_4.mlx file which answers the same questions as in Exercice 1.

Exercice 5. Write a Matlab function file called `my_filter.m` (.m here not .mlx!!) which implements a 10-point averaging FIR filter defined as :

$$H(z) = \sum_{k=0}^{9} \frac{1}{10} z^{-k}$$

To implement the filtering (the difference equation), you will need to use a `for ...end` loop. Below is a simple example to illustrate its use :

```
j=0;
for i=1:3
j=j+i;
end
```

To test your file, we will consider a 1-second duration signal sampled at 100 Hz, composed of two sinusoidal components at 3 Hz and 40 Hz. The goal is to filter out the 40Hz sine.
Test the following commands in a td4_5.mlx file.

```
fs = 100;
t = 0:1/fs:1;
x = sin(2*pi*3*t)+1/4*sin(2*pi*40*t);
figure
plot(t,x)
```

Now create a 10-point averaging FIR filter. Filter using your file `my_filter` the Matlab `filter` program for comparison :

```
b = ones(1,10)/10;
a=1;
y = my_filter(b,a,x);
yy = filter(b,a,x);
plot(t,x,t,y,t,yy)
legend('Original','My filtering','Normal filtering')
```

Both programs should eliminate the 40 Hz sinusoid.

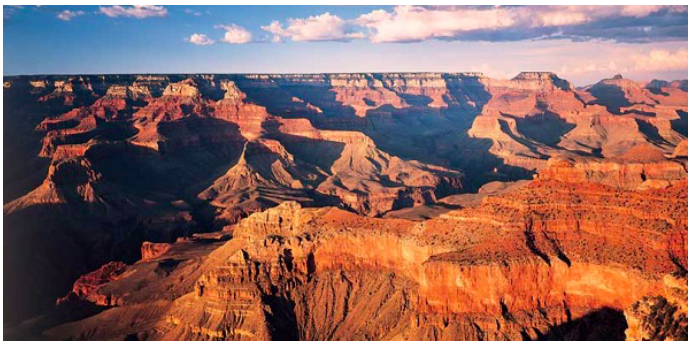
The plot should also show that both filtered signals are delayed by about five samples which is introduced by the phase distortions. To remove the delay, try to use the `filtfilt` function and study its effects

```
yyy = filtfilt(b,a,x);
plot(t,x,t,y,t,yy,t,yyy)
legend('Original','My filtering','Normal filtering','Zero-phase filtering')
```

By using the documentation of the function `doc filtfilt`, try to figure out how the filtfilt function works.

TD 5

Filtrage numérique à l'église et dans le Grand Canyon - Suppression d'un harmonique parasite dans un morceau de trompette et de la note Sol de la gamme



Objectifs

- Découvrir l'outil d'analyse de filtre `filterDesigner`
- Comprendre les différences majeures entre les filtres numériques RIF et RII
- Opérer un produit de convolution à l'église et dans le Grand Canyon
- Supprimer un harmonique parasite par filtre réjecteur
- Supprimer la note Sol de la gamme musicale par filtre réjecteur

Vous sauvegarderez l'ensemble de vos commandes dans des programmes `td5_1.mlx`, `td5_2.mlx` et `td5_3.mlx` qui débuteront ainsi :

```
% Fichier td5_X.mlx
% Votre prénom nom
% Date du jour

clear
clc
close all

..., etc
```

5.1 Analyse de filtres numériques RIF/RII

Nous allons ici découvrir et exploiter l'outil d'analyse de filtres numériques disponibles dans Matlab `filterDesigner`.

Lancez cet outil à partir de la fenêtre Command de Matlab : `filterDesigner` On va réaliser dans un premier temps l'analyse d'un filtre RIF du premier ordre décrit par :

$$H_1(z) = \frac{1}{2} + \frac{1}{2}z^{-1}$$

Cliquez sur la deuxième icône en partant du bas gauche de la fenêtre *Import Filter from Workspace*. Dans le menu déroulant *Filter Structure*, sélectionnez *Direct-Form FIR* et spécifiez `[0.5 0.5]` pour le numérateur et une fréquence d'échantillonnage Fs=10kHz. Cliquez ensuite sur *Import* pour obtenir sur la fenêtre du haut la réponse fréquentielle d'amplitude du filtre. Comparez le tracé obtenu avec celui présenté dans les transparents de cours et vérifiez le filtrage de type passe-bas réalisé.

Utilisez les icônes au dessus de la réponse fréquentielle pour tracer la réponse fréquentielle en phase (vérifiez qu'elle est bien linéaire), les retards de phase et de groupe (vérifiez qu'ils sont constants), la réponse impulsionnelle (vérifiez qu'elle comporte un nombre fini d'échantillons non nuls (2 ici !)), le diagramme des pôles et des zéros (vérifiez que le pôle est bien à l'origine).

Quel type de filtrage obtient-on lorsque le numérateur du filtre devient `[0.5 -0.5]` ? Il faudra cliquer sur *Import* pour obtenir la réponse pour ce nouveau filtre.

Réalisez à présent l'analyse du filtre RII du premier ordre décrit par :

$$H_2(z) = \frac{0.2}{1 - 0.8z^{-1}}.$$

Dans le menu déroulant de *Filter Structure*, sélectionnez *Direct-Form I* et spécifiez `[0.2]` pour le numérateur, `[1 -0.8]` pour le dénominateur et une fréquence d'échantillonnage Fs=1kHz. Cliquez ensuite sur *Import*.

Comparez le tracé obtenu avec celui présenté dans les transparents de cours et vérifiez le filtrage de type passe-bas réalisé.

Tracez la réponse fréquentielle en phase (vérifiez qu'elle est non linéaire), les retards de phase et de groupe (vérifiez qu'ils ne sont plus constants), la réponse impulsionnelle (vérifiez qu'elle comporte un nombre infini d'échantillons non nuls), le diagramme des pôles et des zéros (vérifiez que le pôle est bien à l'intérieur du cercle unité).

Quel type de filtrage obtient-on lorsque le dénominateur du filtre devient `[1 0.8]` ? Il faudra cliquer sur *Import* pour obtenir la réponse pour ce nouveau filtre.

Faites valider votre traitement et analyse par l'enseignant.

5.2 Du filtrage numérique à l'église et dans le Grand Canyon

Créez un fichier `td5_1`.mlx.
Chargez le fichier `inigo.wav` à l'aide de la fonction Matlab `audioread` :
```
[y,fe] = audioread('inigo.wav');
```
Ecoutez l'extrait enregistré à l'aide de la fonction `sound`.
```
sound(y,fe);
```

La simulation de l'écoute d'un signal sonore dans un environnement donné peut être obtenue en convoluant la réponse impulsionnelle (l'empreinte sonore) d'un environnement (qui peut être vu comme un filtre) avec le signal sonore d'origine.

Chargez les fichiers `church.wav` et `grandcanyon.wav`. A l'aide de la fonction `stem`, tracez les réponses impulsionnelles (les empreintes sonores) de l'église et du Grand Canyon.

A l'aide de la fonction Matlab `conv` (`help conv`), réalisez la convolution entre l'empreinte sonore de l'église puis du Grand Canyon et le signal sonore d'origine. Ecoutez le signal obtenu dans chaque cas.

A présent, on considère un filtre ayant comme réponse impulsionnelle :

$$h(k) = \delta(k) + 0.5\delta(k - i), \quad \text{avec} \quad i = 1000.$$

A partir de $h(k)$, déterminez la fonction de transfert $H(z)$ du filtre. De quel type de filtre (RIF/RII) s'agit-il ?

A partir de $H(z) = \frac{Y(z)}{E(z)}$, déduisez l'équation aux différences liant les échantillons de la sortie $y(k)$ de ce filtre à ceux de l'entrée $e(k)$.

A l'aide de l'outil graphique `filterDesigner`, tracez la réponse impulsionnelle $h(k)$ et le diagramme des pôles et des zéros. Le filtre est-il stable ?
On pourra s'inspirer de la solution suivante pour définir le filtre sous Matlab puis renseigner les champs Numerator et Denominator du filtre RII (Direct-Form I) avec les variables `B` et `A` :
```
i=1000;
B=[1 zeros(1,i-1) 0.5];
A=[1 zeros(1,i)];
```

Utilisez la fonction Matlab `filter` (`help filter`) pour réaliser le filtrage (cela revient à opérer le produit de convolution) du signal `inigo.wav`.
```
yf=filter(B,A,y);
```

Ecoutez le signal sonore obtenu :
```
sound(yf,fe);
```

Que constatez-vous ?

Augmentez progressivement la valeur de i et déterminez la valeur minimale du retard (en ms) permettant de générer un écho qui est perçu par l'oreille humaine.

Faites valider votre traitement et analyse par l'enseignant.

5.3 Suppression d'un harmonique parasite dans un morceau de trompette

Créez un nouveau fichier `td5_2`.mlx.

On a enregistré un trompettiste jouant une note devant un microphone.
Chargez le fichier `trompette1.mat` :

Utilisez la fonction `soundsc` pour écouter le morceau de trompette :
`soundsc(y,fe);`

Observez le spectre d'amplitude du morceau de trompette. Vous pouvez utiliser l'outil `signalAnalyzer` pour visualiser le spectre.
S'agit-il d'un son pur ? Quelle est la fréquence de l'harmonique fondamental ? Combien observez-vous d'harmoniques ?

Ce morceau de trompette contient un harmonique parasite que l'on souhaite supprimer. Identifiez la fréquence de cet harmonique parasite.

En vous appuyant sur les transparents de cours, rappelez le diagramme des pôles et zéros du filtre réjecteur qui permet de supprimer un harmonique parasite.
Rappelez également la fonction de transfert $H(z)$ du filtre réjecteur.
Matlab ne comporte pas de fonction dédiée à ce type de filtre réjecteur. On pourra directement saisir les coefficients de la fonction de transfert $H(z)$ en s'inspirant de la solution suivante :

```
f0=??; % fréquence parasite à supprimer
Omega0=2*pi*f0/fe;
a= ??; %coefficient influençant la sélectivité du filtre réjecteur
B=[??]; % coefficients du numérateur du filtre réjecteur en puissance décroissante de z
A=[??]; % coefficients du dénominateur du filtre  réjecteur en puissance décroissante de
figure
freqz(B,A,512,fe); % Calcul de la réponse fréquentielle du filtre pour 512 points
yf=filter(B,A,y);  % produit de convolution du filtre et du signal bruité
soundsc(yf,fe); % pour écouter du morceau filtré
```

Tracez la réponse impulsionnelle, fréquentielle et le diagramme des pôles/zéros de votre filtre réjecteur avec l'outil `filterDesigner` ou bien en utilisant les différentes fonctions Matlab comme `impz, zplane, freqz, dbode`.

L'efficacité du filtrage pourra être observée à l'aide l'outil graphique `signalAnalyzer` en superposant les spectres du signal original et filtré.

Optimisez les paramètres du filtre réjecteur afin de supprimer le plus efficacement possible l'harmonique perturbateur et ainsi récupérer le signal musical d'origine avec le moins de

distorsions possibles.

Faites valider votre filtre réjecteur par l'enseignant.

5.4 Suppression d'une note de la gamme musicale

Créez un nouveau fichier `td5_3`.mlx.

Concevez un filtre numérique qui permet de supprimer totalement la note *sol* de la gamme musicale que vous avez synthétisée lors du TD n°2.
Faites valider votre solution par l'enseignant.

TD 6

Analyse et conception de filtres numériques - Elimination d'échos sur des communications téléphoniques

Exercice 6.1. Analyse d'un filtre numérique
La fonction de transfert d'un filtre numérique a deux pôles en $z = 0$, et deux zéros en $z = -1$ et $z = 1$.

(a) Déterminer la fonction de transfert $H(z)$.

(b) En déduire l'équation aux différences du filtre.

(c) Déterminer la réponse impulsionnelle du filtre

(d) Tracer le diagramme des pôles et des zéros.

(e) Le filtre est-il stable. Justifier.

(f) Préciser le type de filtres numériques : RIF ou RII.

Exercice 6.2. Conception d'un filtre RII par la méthode bilinéaire
Concevoir par la méthode bilinéaire le filtre numérique équivalent à un filtre analogique passe-bas de type RC.
On supposera la fréquence de coupure du filtre RC égale à $f_c = \frac{1}{2\pi RC} = 30$Hz et une fréquence d'échantillonnage $f_e = 60\pi$ Hz.

(a) Rappeler la fonction de transfert de Laplace du filtre RC.

(b) Déterminer par la méthode bilinéaire la fonction de transfert en z du filtre numérique RII équivalent.

(c) Préciser l'ordre du filtre numérique.

(d) Exprimer la fonction de transfert en z en puissance négative de z.

(e) En déduire l'équation aux différences du filtre.

(f) Donner les commandes Matlab qui permettent de calculer les coefficients du filtre RII à partir des coefficients du filtre analogique (fonction bilinear) et de tracer sur la même figure les diagrammes de Bode des filtres analogiques et numériques.

Exercice 6.3.
L'élimination d'écho sur des communications téléphoniques constitue une difficulté. Nous supposons une situation simple où un seul écho est créé. Soit $e(n)$ le signal transmis. A cause de l'écho, le signal reçu s'écrit :

$$s(k) = e(k) + \alpha e(k - D)$$

où α représente le facteur d'atténuation avec $0 < \alpha < 1$ et D est le nombre d'échantillons de retard du signal répliqué.

(a) Déterminer la réponse impulsionnelle $h(k)$ du système d'écho.

(b) Déterminer la fonction de transfert $H(z)$ du système d'écho.

(c) Déterminer les pôles et les zéros de $H(z)$ pour $\alpha = 0.1$ et $D = 12$. On supposera $(0.826)^{12} = 0.1$

(d) En déduire le diagramme des pôles et des zéros.

(e) Déterminer la réponse fréquentielle $H(f)$ du système d'écho.

(f) En déduire $|H(f)|^2$. Tracer $|H(f)|^2$ pour f compris entre 0 et $f_e/2$.

(g) On suppose à présent avoir une bonne estimation de $\alpha = 0.1$ et $D = 12$. On souhaite concevoir un système d'annulation d'écho, c'est à dire déterminer le filtre numérique de réponse impulsionnelle $g(k)$ qui appliqué au signal reçu permettra de supprimer l'écho :

$$\hat{e}(k) = g(k) * s(k)$$

(i) Déterminer la fonction de transfert $G(z)$ du filtre d'annulation d'écho.

(j) Déterminer les pôles et les zéros de $G(z)$. En déduire le diagramme des pôles et des zéros.

(k) Déterminer l'équation aux différences du filtre d'annulation d'écho. Y-a-t-il une difficulté pratique pour implanter ce système d'annulation d'écho ?

TD 7

Conception de filtres numériques - Applications pour séparer deux codes Morse et décrypter le son de Canal+

Objectifs

- Découvrir et utiliser l'interface graphique **sptool**
- Concevoir et appliquer des filtres numériques RIF et RII sur un extrait musical
- Séparer deux codes Morse enregistrés simultanément par filtrage RIF
- Décrypter le son Canal+ par modulation d'amplitude et filtrage RIF/RII

7.1 Filtrage RIF d'un morceau de musique

Le filtrage d'un morceau de musique correspond par exemple à l'opération que vous effectuez lorsque vous ajustez le bouton des aigus ou des graves sur une chaîne audio ou sur votre autoradio, ce qui a pour effet de modifier la proportion graves/aigus du son.
Nous allons appliquer un filtrage de type RIF sur un morceau de musique à l'aide de la fonction `fir1` qui a le prototype suivant :
`b = fir1(M,fcn,'type_filtrage',type_fenetre(M+1));`
où

- `M` est l'ordre du filtre RIF
- `fcn` est la fréquence de coupure normalisée comprise entre 0 et 1 (fcn=fc/(fe/2))
- `type_filtrage` = low, high, ...
- `type_fenetre` = boxcar, hamming, ...

Ouvrez le fichier ci-dessous à l'aide de la commande suivante :
`open exemple_filtrage.mlx`
Cliquez sur `run and Advance` et affichez les figures dans le texte en sélectionnant l'icône approprié dans le coin haut à droite de l'éditeur. Veillez lorsque le programme est en pause a bien avoir la fenêtre *Command* active en cliquant dessus pour faire progresser le programme et afficher les différentes figures.

Modifiez la fréquence de coupure `fc` entre 500Hz et 15 kHz et relancez le programme. Ecoutez le résultat (le son doit devenir plus ou moins aigu, le filtre ayant éliminé plus ou moins de graves).

Complétez le programme pour appliquer un filtrage RIF passe-bande de votre choix en consultant l'aide en ligne de la fonction `fir1` (doc fir1).
Faire valider votre programme par l'enseignant.

7.2 Découverte de SPTOOL et séparation de deux codes Morse enregistrés simultanément

Un des grands domaines du traitement du signal est la séparation de sources. L'idée étant, à partir d'un mélange de différents signaux provenant de plusieurs sources (par exemple plusieurs personnes qui parlent en même temps), de parvenir à reconstruire le message associé à chacune des sources (par exemple retrouver ce qu'a dit chaque personne). C'est un problème courant en acoustique donc, mais aussi en télécommunications, ...
Cette séparation de sources peut être réalisée via à un filtrage linéaire adapté.
Le but dans cette partie est d'appliquer un filtrage numérique pour séparer deux codes Morse enregistrés simultanément mais qui occupent des bandes de fréquences disjointes.

Chargez le fichier `code_morse.mat`. Ce signal est sauvegardé dans la variable y et a été enregistré à la fréquence d'échantillonnage $f_e = 11025$ Hz.

Nous allons découvrir et exploiter un nouvel outil graphique. Il s'agit de l'application SPTOOL de traitement de signaux numériques sous Matlab. Pour cela, lancez depuis la fenêtre Command :
`sptool`
Importez le signal (Menu File+Import+Workspace). Dans le menu déroulant en haut à gauche sélectionnez `Import as: Signal` puis renseignez les champs "Data" et "Sampling Frequency" (Sélectionnez (en cliquant sur les flèches) les variables correspondantes dans "Workspace Content").

Observez l'évolution temporelle du signal en cliquant sur "View" dans le panneau "Signals", et écoutez le signal en cliquant sur le haut-parleur (sélectionnez avec les réglettes verticales le support temporel à écouter). Combien de sources sonores distinctes entendez-vous ?
Calculez le spectre du signal (panneau "Spectra" + Create) à l'aide de la méthode "FFT" en spécifiant NFFT à 11025. Vous devez observer nettement la présence de deux raies (ou pics) fréquentielles. Expliquez pourquoi les zones fréquentielles autour des deux pics sont susceptibles de contenir l'information des deux sources sonores précédemment identifiées. Vous pouvez consulter si besoin la page Wikipédia sur le code Morse.

Vérifiez cette assertion en extrayant du signal les sons centrés autour du premier pic. Pour cela il vous faudra concevoir un filtre RIF de type passe-bas (panneau "Filters" + New), en configurant les différentes options de la manière suivante :
Response type : Lowpass
Design method : FIR + Window
Filter Order : Minimum order
Options : Window+Kaiser
Sampling Frequency : 11025

Enfin renseignez les fréquences des bandes passante et atténuée que vous choisirez à partir de l'analyse spectrale du signal.
Cliquez sur *Design Filter* pour voir la réponse fréquentielle du filtre RIF ainsi conçu par la méthode de la fenêtre. Relevez l'ordre du filtre et vérifiez que la réponse obtenue vérifie bien les spécifications attendues.

Le filtrage du signal se fait par la commande "Apply" dans le panneau "Filters". Visualisez et écoutez le signal filtré. Visualisez le spectre du signal filtré. Conclusion.

Répétez les opérations précédentes mais avec un filtre passe-bande pour extraire le code centré autour du second pic. Faites valider votre traitement et votre analyse par l'enseignant.
Faites valider votre solution par l'enseignant.

7.3 Décryptage du son canal+

Créez un fichier `td7.mlx`.

Chargez le fichier `son-canal+_crypte.wav` à l'aide de la fonction Matlab `audioread` :
```
[y,fe] = audioread('son-canal+_crypte.wav');
```
Ecoutez l'extrait enregistré à l'aide de la fonction `sound`.
```
sound(y,fe);
```
Il s'agit d'un extrait de son Canal+ analogique crypté. Le but de cet exercice est de décrypter ce morceau, ce qui revient à compenser l'effet d'une modulation. La procédure de modulation/démodulation est communément utilisée en télécommunications pour transmettre un message basse-fréquence (signal de parole) sur des ondes hautes-fréquences (ondes hertzienne). Elle est aussi utilisée en cryptage, comme illustré ici pour le cryptage du son Canal+.

La procédure de cryptage du son est la suivante :

1. filtrage passe-bas (PB) du signal avec $f_c = 12{,}8$ kHz
2. modulation d'amplitude du signal filtré par multiplication par une sinusoïde de fréquence fondamentale f_c
3. filtrage passe-bas du signal modulé avec $f_c = 12{,}8$ kHz

Ces 3 opérations peuvent être résumées ainsi :

$$x(t) \Longrightarrow \boxed{\text{Filtrage PB}} \Longrightarrow y(t) \Longrightarrow 2y(t)\sin(2\pi f_c t) \Longrightarrow \boxed{\text{Filtrage PB}} \Longrightarrow z(t)$$

A l'aide d'une interprétation des opérations dans le domaine fréquentiel, rappelez que les opérations (2) et (3) décrites ci-dessus reviennent à faire une «inversion» de spectre du signal original comme illustrée sur le dernier transparent de cours portant sur la transformée de Fourier à temps continu.

Quelle est le rôle du premier filtrage passe-bas décrit par l'opération (1) ci-dessus ?

La procédure de décryptage du son Canal+ est réalisée de manière similaire à celle du cryptage et nécessite les opérations suivantes :

1. filtrage passe-bas du signal crypté avec $f_c = 12{,}8$ kHz

2. modulation d'amplitude du signal filtré par multiplication par une sinusoïde de fréquence fondamentale f_c
3. filtrage passe-bas du signal modulé avec $f_c = 12{,}8$ kHz

Ces 3 opérations peuvent être résumées ainsi :

$$z(t) \Longrightarrow \boxed{\text{Filtrage PB}} \Longrightarrow z_f(t) \Longrightarrow 2z_f(t)\sin(2\pi f_c t) \Longrightarrow \boxed{\text{Filtrage PB}} \Longrightarrow \hat{x}(t)$$

Elles sont illustrées sur la figure 7.1.

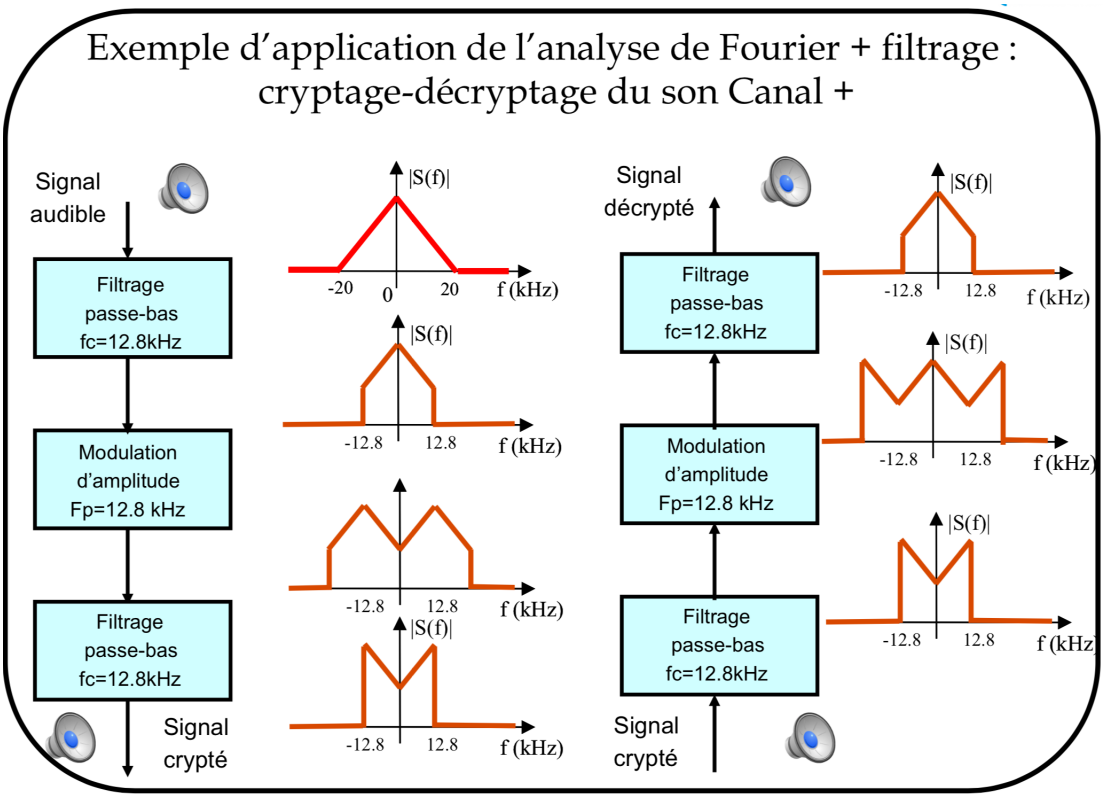


Figure 7.1 – Résumé des opérations pour crypter et décrypter le son canal+

Il vous est demandé de concevoir le filtre passe-bas (nécessaire pour réaliser le décryptage) ayant les spécifications suivantes :

1. fréquence de coupure $f_c = 12{,}8$ kHz
2. gain unitaire en bande passante, avec des oscillations autorisées de 0,01 dB
3. atténuation de 40 dB en bande atténuée
4. transition de 500 Hz au plus

Concevez ce filtre à l'aide de l'outil graphique SPTOOL. Comparez l'efficacité des filtres de type "FIR Window Kaiser" et "Butterworth IIR" sous la contrainte d'utiliser un nombre minimum de paramètres. L'efficacité de chaque filtrage pourra être évaluée en termes de :

1. coût de calcul (nombre de paramètres du filtre)
2. linéarité de la phase
3. platitude de la bande passante
4. atténuation en bande stoppée
5. largeur de la zone de transition

La conception des filtres peut aussi être réalisée à l'aide des fonctions `fir1` et `butter`.
Implantez les 3 opérations nécessaires pour réaliser le décryptage du morceau de canal+.
La modulation d'amplitude peut facilement être appliquée à l'aide de l'opération de multiplication élément par élément `.*` sous Matlab.
Faites valider le décryptage du son par l'enseignant.

TD 8

Casting de siffleur - Préservation de l'anonymat de témoin

Ce TD ferait l'objet d'un compte-rendu qui sera noté.

Il est demandé de suivre les consignes présentées ci-dessous pour rédiger votre compte-rendu de projet.

Consignes pour la rédaction de votre compte-rendu

Un compte-rendu de projet est un document scientifique. Il doit donc respecter une certaine organisation. Votre compte-rendu devra être structuré de la manière suivante :

- une introduction générale précisant les objectifs principaux du projet
- Pour chaque partie ou exercice :
  - ▷ une présentation brève des attendus
  - ▷ une présentation succincte des méthodes de traitement ou d'analyse
  - ▷ une description des résultats sous la forme demandée
  - ▷ une analyse critique des résultats obtenus
  - ▷ une conclusion brève
- une conclusion générale qui résume ce qui a été réalisé et les éventuelles difficultés rencontrées.
- une annexe rassemblant les programmes Matlab que vous avez développés.

Vous pouvez exploiter le nouvel éditeur de Matlab Live Editor.

Vous pouvez exploiter l'outil `SPTOOL` *(ou* `filterDesigner`*) pour réaliser les différentes opérations et analyses. Vous pouvez préférer directement utiliser les commandes Matlab qui seront alors plus faciles à intégrer dans un programme .mlx du nouvel éditeur de Matlab Live Editor.*
Votre compte-rendu à rédiger obligatoirement en binôme (sauf exception accordée par l'enseignant) est `à envoyer par courriel` à l'enseignant sous la forme d'un `fichier unique au format pdf` joint au message.
Les fichiers Matlab seront également joints au courriel sous la forme d'un fichier zippé.

8.1 Casting de siffleur

Le sifflement est une pratique musicale populaire, comme en témoigne la chanson «*Siffler en travaillant*» dans le film *Blanche-Neige et les Sept Nains*. Des artistes de music-hall et de vaudeville sont siffleurs professionnels mais n'importe qui peut siffler une mélodie sans apprentissage particulier.

Figure 8.1 – Un siffleur professionnel !

Cet exercice va permettre d'évaluer vos talents de siffleur et en particulier déterminer si vous êtes capable de siffler de manière la plus pure possible. Qui c'est, vous pourriez à l'issue de cette expérience décider de participer au casting de siffleur et ainsi siffler aux côtés d'un orchestre symphonique lors d'un concert de plein air comme celui qui a eu lieu à l'été 2017 près de Royan.
`http://www.violonsurlesable.com/casting-siffleurs-st-palais-un-violon-sur-la-ville_2017.html`

Visionnez la courte vidéo (2 mn) et tester vos talents de siffleur en essayant de siffler la musique du film *Le Pont de la rivière Kwaï*. Cela vous permettra de vous échauffer pour la suite.

8.1.1 Mesure de la pureté de votre sifflement

Le sifflement par resserrement des lèvres est certainement la forme la plus commune de sifflement mélodique. La hauteur ou fréquence et l'intonation musicale dépendent de l'interprète, sa durée est limitée par le souffle du siffleur. On rappelle que la bande de fréquences audibles par l'oreille humaine s'étale de 20 Hz à 20 kHz. Lorsque la fréquence est faible, le son est grave (de 20 à 200 Hz). On parle de son médium pour une fréquence comprise entre 200 et 1000 Hz et de son aigu lorsque la fréquence est comprise entre 1000 et 15000 Hz.

A l'aide de votre téléphone, enregistrez-vous pendant une dizaine de secondes en train de siffler à une fréquence constante de votre choix et de manière la plus pure possible.

Chargez le fichier sous Matlab, écoutez l'enregistrement et sélectionnez deux secondes qui vous semblent les meilleures pour l'analyse de la pureté de votre sifflement.

Affichez le spectre d'amplitude du signal et déterminez la fréquence fondamentale de votre sifflement. Observez la présence d'éventuels harmoniques ou distorsions.
Il existe différentes manières de mesurer la qualité d'un oscillateur qui produit, comme dans le cas de votre sifflement, un signal sinusoïdal. Une manière consiste à calculer le coefficient de

distorsion harmonique défini comme :

$$\eta = 1 - \frac{E_{fond}}{E_{tot}}$$

où E_{fond} représente l'énergie de l'harmonique fondamental et E_{tot} représente l'énergie totale du signal enregistré.
Une deuxième façon pour mesurer la qualité d'un oscillateur est d'estimer un modèle paramétrique AR d'ordre 2 et de calculer la distance des pôles au cercle unité comme mesure de distorsion harmonique. Cette seconde manière ne sera pas exploitée ici.

L'énergie totale d'un signal échantillonné $x(k)$ calculée à partir de N échantillons enregistrés est définie par :

$$E_{tot} = \sum_{k=1}^{N} |x(k)|^2$$

Sous Matlab, on pourra calculer l'énergie totale du signal à l'aide de la commande, suivante : `Etot=sum(abs(x).^2)`

1. Calculez l'énergie totale de votre sifflement.
2. Appliquez un filtrage passe-bande pour ne conserver que l'harmonique fondamental de votre sifflement.
3. Calculez l'énergie de l'harmonique fondamental (signal filtré).
4. En déduire la valeur du coefficient de distorsion harmonique η.

8.1.2 Séparation de deux sifflements

L'objectif à présent est de réaliser la séparation de deux sifflements enregistrés simultanément.

A l'aide de votre téléphone, avec votre binôme [1], enregistrez-vous en train de siffler simultanément à des fréquences constantes mais différentes pendant une dizaine de secondes. Il est conseillé que l'un des sifflements soit, si possible, aigu et l'autre médium ou grave.

Chargez le fichier sous Matlab, écoutez l'enregistrement et sélectionnez deux secondes qui vous semblent les meilleures pour l'analyse des deux sifflements.

Affichez le spectre d'amplitude du signal et déterminez les deux fréquences fondamentales des sifflements.

Déterminer les deux filtres numériques à appliquer pour séparer les deux sifflements. Préciser les caractéristiques des deux filtres : type de filtrage, fréquences de coupure, etc.
Vous pouvez concevoir les filtres à l'aide de l'outil graphique `filterDesigner` puis générer un programme Matlab qui génère le filtre (Menu *File* puis *Generate Matlab Code* puis *Filter Design Function*) qui fournira le filtre directement lors de l'appel sous Matlab.

1. Si vous n'avez pas de binôme, demandez à un de vos amis !

Appliquez à l'aide de la fonction `filter` de Matlab les filtrages sur le signal original et vérifiez que les deux sifflements sont bien séparés en les écoutant.

Confirmez votre impression en affichant le spectre d'amplitude de ces derniers.

8.2 Du traitement numérique pour préserver l'anonymat d'un témoin

Il existe différents traitements qui permettent de préserver l'anonymat d'une témoin interrogé en "masquant" sa voix. Le but de cet exercice est de mettre en oeuvre un traitement qui donne un effet métallique tout en conservant compréhensible la voix du témoin.
Le principe du traitement est simple : il consiste à décaler le spectre de la voix originale de quelques centaines de Hertz. Cela revient à faire une modulation sans porteuse. On effectue un produit simple entre la voix originale et un signal sinusoïdal à fréquence fixe.

1. Enregistrez une phrase courte pendant 10s.
2. Appliquez une modulation sans porteuse du signal en choisissant la fréquence porteuse à 1300 Hz (on pourra affiner cette valeur au besoin).
3. Le signal modulé en amplitude est alors inaudible. Expliquez pourquoi en déterminant le spectre du signal modulé à l'aide d'un raisonnement graphique.
4. En déduire l'opération à appliquer sur le signal modulé pour ne conserver que la partie produisant l'effet métallique de la voix.
5. Appliquez l'opération sur le signal modulé et vérifiez l'effet métallique de la voix ainsi transformée.

TD 9

Processus aléatoires

Quelques rappels
En théorie du signal, on étudie le plus souvent des signaux dépendant du temps et dont l'évolution semble imprévisible. Pour les modéliser, on utilise la notion de *processus aléatoire* qui associe à chaque évènement une réalisation qui est une fonction du temps.
Un signal aléatoire noté $y(k)$ est donc considéré comme la réalisation d'un processus aléatoire noté en gras $\mathbf{y}(k)$. Un processus aléatoire comprend donc une infinité de réalisations :

— à un instant donné k, les valeurs du processus correspondent à une variable aléatoire

— pour un évènement donné, les valeurs du processus forment une réalisation ou encore un signal aléatoire (c'est ce qui est observé en pratique).

Le concept de processus aléatoire est une idéalisation mathématique abstraite mais très pratique pour faire l'analyse d'un signal aléatoire.
Dans Matlab, la fonction `randn` permet de générer facilement des réalisations de processus aléatoire. Les lignes ci-dessous permettent, par exemple, de tracer quatre réalisations d'un processus aléatoire gaussien centré de variance 1 observé sur 100 points :

```
y=randn(100,4);
for i=1:4
subplot(2,2,i)
plot(y(:,i))
end
```

Les quatre réalisations du processus aléatoire gaussien sont représentées sur la figure 9.1. Bien que les réalisations semblent totalement imprévisibles, on peut déduire pour les 4 réalisations que la moyenne est nulle et que l'écart-type est d'environ 1.
On restreindra notre attention au cas des signaux stationnaires au sens large (ou second ordre) qui vérifient les deux propriétés suivantes :

— la moyenne $E\big(\mathbf{y}(k)\big)$ est indépendante de k (E représente l'espérance mathématique ou valeur espérée) ;

— la fonction d'auto-corrélation $E\big(\mathbf{y}(k)\mathbf{y}(k-\tau)\big)$ est indépendante de l'instant k où est elle est évaluée.

Soient deux processus aléatoires stationnaires $\mathbf{x}(k)$ et $\mathbf{y}(k)$, on rappelle les définitions suivantes :

— **Moyenne** de $\mathbf{y}(k)$
$$m_{\mathbf{y}} = E\big(\mathbf{y}(k)\big)$$

— **Variance** de $\mathbf{y}(k)$
$$\sigma_{\mathbf{y}}^2 = E\big(\mathbf{y}(k) - E(\mathbf{y}(k)^2)\big)$$

— **Fonction d'inter-corrélation**
$$R_{\mathbf{yx}}(\tau) = E\big(\mathbf{y}(k)\mathbf{x}(k-\tau)\big)$$

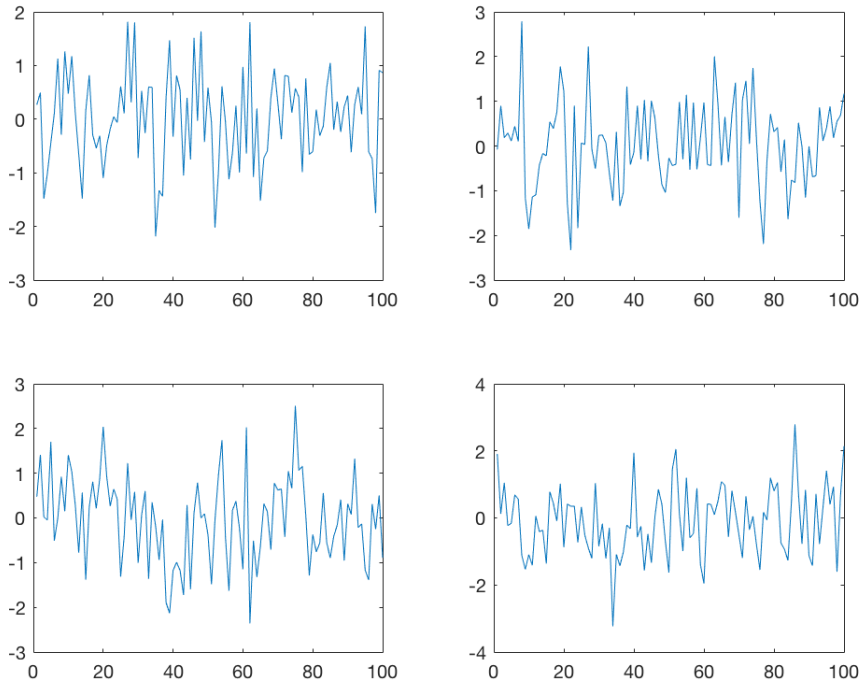


Figure 9.1 – Exemples de réalisations d'un processus aléatoire gaussien

— **Fonction d'auto-corrélation**

$$R_{\mathbf{yy}}(\tau) = E\big(\mathbf{y}(k)\mathbf{y}(k-\tau)\big)$$

— **Puissance** de $\mathbf{y}(k)$:

$$P_{\mathbf{y}} = E\big((\mathbf{y}(k))^2\big) = R_{\mathbf{yy}}(0) + m_y^2$$

Si le signal est centré ($m_{\mathbf{y}} = 0$), $P_{\mathbf{y}} = R_{\mathbf{yy}}(0)$

— **Densité spectrale de puissance**. C'est la transformée de Fourier à temps discret de la fonction d'auto-corrélation :

$$\Phi_{\mathbf{y}}(f) = \sum_{\tau=-\infty}^{\infty} R_{\mathbf{yy}}(\tau) e^{-j2\pi f \tau T_e}$$

— **Corrélation**. Si la valeur de $\mathbf{y}(k)$ à l'instant k n'est pas corrélée à la valeur de $\mathbf{x}(k-\tau)$ (pour une valeur de τ fixée) alors

$$R_{\mathbf{yx}}(\tau) = 0$$

— **Dépendance**. Si $\mathbf{y}(k)$ et $\mathbf{x}(k)$ sont indépendants alors

$$R_{\mathbf{yx}}(\tau) = 0 \quad \forall \quad \tau$$

— **Bruit blanc**. Un bruit blanc $\mathbf{e}(k)$ est un processus aléatoire centré très spécial. Sa densité spectrale de puissance est constante sur tout l'axe des fréquences. Sa fonction d'auto-corrélation est une impulsion discrète en 0 d'amplitude $\sigma_{\mathbf{e}}^2$. La valeur du processus à l'instant k est donc non-corrélée avec les valeurs du processus aux autres instants. Les propriétés d'un bruit blanc $\mathbf{e}(k)$ de variance $\sigma_{\mathbf{e}}^2$ sont résumées ci-dessous :

$$E\big(\mathbf{e}(k)\big) = 0 \qquad R_{\mathbf{ee}}(\tau) = \begin{cases} \sigma_{\mathbf{e}}^2 & \text{pour } \tau = 0 \\ 0 & \text{sinon} \end{cases} \qquad \Phi_{\mathbf{e}}(f) = \sigma_{\mathbf{e}}^2 \quad \forall \quad f$$

— **Filtrage d'un processus aléatoire**. Soit un processus aléatoire stationnaire $\mathbf{x}(k)$ de densité spectrale de puissance $\Phi_{\mathbf{x}}(f)$ mis à l'entrée d'un filtre linéaire de réponse impulsionnelle $h(k)$ et de réponse fréquentielle $H(f)$, on montre que le processus aléatoire en sortie du filtre est également stationnaire et possède les propriétés suivantes :

▷ Sa moyenne est :

$$m_{\mathbf{y}} = H(0)m_{\mathbf{x}}$$

▷ Sa densité spectrale de puissance est :

$$\Phi_{\mathbf{y}}(f) = |H(f)|^2\Phi_{\mathbf{x}}(f).$$

— **Processus MA d'ordre** n_c. Son équation de récurrence est :

$$\mathbf{y}(k) = \mathbf{e}(k) + c_1\mathbf{e}(k-1) + \ldots + c_{n_c}\mathbf{e}(k-n_c)$$

où $e(k)$ est un bruit blanc de variance $\sigma^2_{\mathbf{e}}$.
Sous Matlab, il est très facile de générer une réalisation d'un processus MA en utilisant la fonction `filter`. Les lignes ci-dessous permettent par exemple de générer une réalisation d'un processus MA d'ordre 2 observé sur 100 points :

```
e=randn(100,1);
y=filter([1 1.5 -1.2],1,e)
plot(y)
```

— **Processus AR d'ordre** n_d. Son équation de récurrence est :

$$\mathbf{y}(k) + d_1\mathbf{y}(k-1) + d_2\mathbf{y}(k-2) + \ldots + d_{n_d}\mathbf{y}(k-n_d) = \mathbf{e}(k)$$

où $\mathbf{e}(k)$ est un bruit blanc de variance $\sigma^2_{\mathbf{e}}$.
Sous Matlab, il est très facile de générer une réalisation d'un processus AR en utilisant la fonction `filter`. Les lignes qui suivent permettent par exemple de générer une réalisation d'un processus AR d'ordre 2 observé sur 100 points :

```
e=randn(100,1);
y=filter(1,[1 -0.9],e)
plot(y)
```

— **Processus ARMA**(n_c, n_d). Son équation de récurrence est :

$$\mathbf{y}(k) + d_1\mathbf{y}(k-1) + d_2\mathbf{y}(k-2) + \ldots + b_{n_d}\mathbf{y}(k-n_d) = \mathbf{e}(k) + c_1\mathbf{e}(k-1) + \ldots + c_{n_c}\mathbf{e}(k-n_c)$$

où $\mathbf{e}(k)$ est un bruit blanc de variance $\sigma^2_{\mathbf{e}}$.

— **Opérateur retard** q^{-1}. Il est défini tel que :

$$\mathbf{y}(k-1) = q^{-1}\mathbf{y}(k)$$

On supposera dans les exercices ci-dessous que les signaux sont échantillonnés à une fréquence f_e=1.

Exercice 9.1. Caractérisation d'un processus aléatoire non centré
On considère un processus aléatoire stationnaire $\mathbf{y}(k)$ généré par la relation suivante :

$$\mathbf{y}(k) = 10 + \mathbf{e}(k) \tag{9.1}$$

où $\mathbf{e}(k)$ est un bruit blanc de variance $\sigma_{\mathbf{e}}^2 = 4$. Sur la figure 9.2 sont représentées deux réalisations de ce processus aléatoire pour $0 \leq k \leq 4$

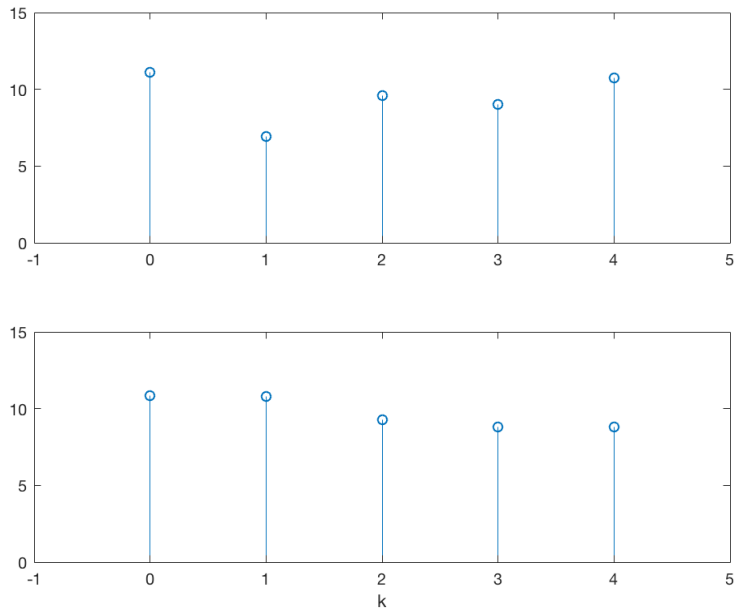


Figure 9.2 – Deux réalisations du processus aléatoire non centré décrit par (9.1)

(a) Expliquer pourquoi les deux réalisations sont différentes sur la figure 9.2.
(b) Déterminer la moyenne de $\mathbf{y}(2)$? En déduire la moyenne de $\mathbf{y}(k)$ pour les autres valeurs de k ?
(c) Déterminer la la variance de $\mathbf{y}(2)$? En déduire la variance de $\mathbf{y}(k)$ pour les autres valeurs de k ?
(d) La figure 9.2 correspond-elle à ce que vous avez déterminé dans les questions b et c ? Justifier.
(e) Déterminer la fonction d'auto-corrélation $R_{\mathbf{yy}}(\tau)$ de $\mathbf{y}(k)$ pour toutes les valeurs de τ.
(f) Calculer la puissance $P_{\mathbf{y}}$ de $\mathbf{y}(k)$.

Exercice 9.2. Caractérisation d'un processus aléatoire
On considère un processus aléatoire stationnaire $\mathbf{y}(k)$ défini par la relation suivante :

$$\mathbf{y}(k) = \mathbf{e}(k-1) - a\mathbf{e}(k-2) \tag{9.2}$$

où $\mathbf{e}(k)$ est un bruit blanc de variance $\sigma_{\mathbf{e}}^2$ et a est un réel.

(a) Déterminer la moyenne de $\mathbf{y}(k)$.
(b) Déterminer la fonction d'auto-corrélation $R_{\mathbf{yy}}(\tau)$ pour toutes les valeurs de τ.
(c) Expliquer pourquoi $\mathbf{y}(k)$ et $\mathbf{y}(k-2)$ sont non corrélés, c'est à dire pourquoi $R_{\mathbf{yy}}(2) = 0$?
(d) Déterminer la fonction d'inter-corrélation $R_{\mathbf{ye}}(\tau)$ pour toutes les valeurs de τ.

(e) Calculer la puissance $P_{\mathbf{e}}$ et $P_{\mathbf{y}}$ de $\mathbf{e}(k)$ et $\mathbf{y}(k)$.

(f) Déterminer la densité spectrale de puissance $\Phi_{\mathbf{e}}(f)$ du signal aléatoire $\mathbf{e}(k)$. Représenter son allure.

(g) Déterminer la densité spectrale de puissance $\Phi_{\mathbf{y}}(f)$ du signal aléatoire $\mathbf{y}(k)$ obtenu en sortie du filtre. Représenter son allure.

Exercice 9.3. Filtrage d'un bruit blanc et équation de Yule-Walker
On considère le processus aléatoire décrit par :

$$\mathbf{y}(k) + a\mathbf{y}(k-1) = \mathbf{e}(k)$$

où $\mathbf{e}(k)$ est un bruit blanc de variance $\sigma_{\mathbf{e}}^2$ et $|a| < 1$.

(a) Déterminer la forme polynomiale de l'équation définissant le processus aléatoire et en déduire le type de modèle (AR, MA, ARMA) pour $\mathbf{y}(k)$?

(b) Déterminer la moyenne de $\mathbf{y}(k)$.

(c) Déterminer la fonction de transfert $H(z)$.

(d) Ce filtre est-il stable ? Justifier.

(e) Déterminer la densité spectrale de puissance $\Phi_{\mathbf{y}}(f)$ de $\mathbf{y}(k)$.

(f) Déterminer la fonction d'auto-corrélation $R_{\mathbf{yy}}(\tau)$ pour $\tau = 0$ et ± 1 puis pour toutes les autres valeurs de τ.

(g) Montrer que les valeurs de la fonction d'auto-corrélation $R_{\mathbf{yy}}(\tau)$pour $\tau = 0$ et ± 1 vérifient le système d'équations

$$\begin{cases} R_{\mathbf{yy}}(0) + aR_{\mathbf{yy}}(1) = \sigma_e^2 \\ R_{\mathbf{yy}}(-1) + aR_{\mathbf{yy}}(0) = 0 \end{cases}$$

(h) Ce système d'équation peut s'écrire sous forme matricielle :

$$RA = c$$

En déduire l'expression des matrices R, A et c.

Remarque. Ce résultat se généralise sans difficulté au cas d'un modèle AR d'ordre n_c. L'équation matricielle à laquelle on arrive est appelée équation de *Yule-Walker*. La matrice R représente alors la matrice de covariance d'ordre $n_c + 1$ du processus $\mathbf{y}(k)$. Comme $\mathbf{y}(k)$ est réel, cette matrice est symétrique.

Exercice 9.4. Filtrage d'un bruit corrélé
On considère le filtre numérique défini par la relation de récurrence suivante :

$$y(k) - \frac{1}{2}y(k-1) = x(k)$$

avec $y(k) = 0$, pour $k < 0$.

(a) Déterminer la fonction de transfert $H(z)$.

(b) Ce filtre est-il stable ? Justifier.

(c) Déterminer analytiquement la réponse fréquentielle $H(f)$. Montrer que son module peut s'écrire :

$$|H(f)| = \sqrt{\frac{1}{\frac{5}{4} - \cos\left(2\pi f\right)}}$$

(d) Tracer l'allure de $|H(f)|^2$.

(e) Calculer sa réponse impulsionnelle $h(k)$ de deux façons différentes :

- ▷ à partir de la définition de la réponse impulsionnelle ;
- ▷ en partant de $H(z)$.
  On rappelle que pour $|z| < 1$:

$$\frac{1}{1-z} = \sum_{k=0}^{\infty} z^k$$

(f) On suppose que $x(k)$ est un signal aléatoire stationnaire au second ordre, centré et de fonction d'auto-corrélation :

$$R_{xx}(\tau) = \begin{cases} 1 & \text{si } \tau = 0 \\ \frac{1}{2} & \text{si } \tau = \pm 1 \\ 0 & \text{sinon} \end{cases}$$

Représenter $R_{xx}(\tau)$.

(g) Déterminer la densité spectrale de puissance $\Phi_x(f)$ du signal aléatoire $x(k)$. Représenter son allure.

(h) Déterminer la densité spectrale de puissance $\Phi_y(f)$ du signal aléatoire $y(k)$ obtenu en sortie du filtre. Représenter son allure.

TD 10

Estimation paramétrique de modèles AR

10.1 Etude en simulation

10.1.1 Génération d'une réalisation du processus aléatoire

Soit le processus aléatoire AR décrit par :

$$\mathcal{S}: \quad D(q^{-1})\mathbf{y}(k) = \mathbf{e}(k) \tag{10.1}$$

où q^{-1} représente l'opérateur retard, $\mathbf{e}$ un bruit blanc de variance 1, et le polynôme $D(q^{-1})$ est défini par :

$$D(q^{-1}) = 1.0 - 1.5q^{-1} + 0.7q^{-2} \tag{10.2}$$

1. Déterminer la fonction de transfert du modèle du processus aléatoire.
2. Donner son équation aux différences.

Nous allons, dans un premier temps, générer une réalisation du processus aléatoire à partir de laquelle il faudra estimer les deux paramètres du système $\mathcal{S}$. Pour cela, une fois sous Matlab, entrer, dans un script Matlab, la séquence de commandes suivante :

```
N=500;
D=[1  -1.5  0.7];
randn('state',sum(100*clock));
e=randn(N,1);
y=filter(1,D,e);
data=iddata(y,[]);
idplot(data)
title('Réalisation du processus AR d''ordre 2')
```

Vous avez à présent, à votre disposition, un jeu de données contenu dans l'objet de données `data` pour effectuer l'estimation paramétrique.

10.1.2 Estimation paramétrique

Choix du type de modèle et de l'ordre du modèle

On supposera que le type de modèle (AR ici) ainsi que l'ordre $n_d = 2$ du modèle sont parfaitement connus.

Estimation par moindres carrés simples

On s'intéresse à l'estimation des paramètres par la méthode des moindres carrés simples. Le modèle recherché est de type AR et prend donc la forme :

$$\mathcal{M}_{\text{AR}} : D(q^{-1})y(k) = e(k) \tag{10.3}$$

1. Rappeler l'équation aux différences associée au modèle recherché d'ordre 2.
2. En déduire l'écriture du modèle sous forme de régression linéaire à l'instant k.
3. En supposant N mesures du signal aléatoire, formuler le problème sous forme matricielle. Donner la dimension de chaque matrice.
4. Rappeler la solution de l'estimateur des moindres carrés (voir transparents de cours).
5. Implanter cette solution sous Matlab pour le jeu de données à votre disposition. Que pouvez-vous dire sur la qualité des estimées des 2 paramètres ? On rappelle que l'on réalise ici une estimation ponctuelle et que l'on ne peut parler de biais de l'estimateur mais plutôt d'erreur d'estimation. Vous pouvez exécuter plusieurs fois votre programme pour dégager la tendance au niveau du biais d'estimation.
6. La boîte à outils Matlab *System Identification* contient différentes fonctions permettant de réaliser l'estimation des paramètres d'un modèle. La fonction `ar` permet en particulier d'estimer les paramètres d'un modèle AR par moindres carrés (*least squares* en anglais). La commande est la suivante :

   ```
   nd=2;
   Mar=ar(data,nd,'ls');
   ```

   `nd` correspond au nombre de paramètres à estimer du polynôme $D(q^{-1})$, l'option `'ls'` spécifie la méthode des moindres carrés. Les paramètres estimés peuvent être visualisés par la commande `present(Mar)`. L'écart-type de chaque paramètre est affiché entre parenthèses.
7. Pour le jeu de données à votre disposition, comparer les résultats de votre implantation avec ceux obtenus via la fonction de la boîte à outils SID.

Analyse de la qualité de votre estimateur de modèle AR par simulation de Monte Carlo

Si le temps le permet, écrire un programme qui permet d'évaluer la qualité de l'estimateur par simulation de Monte Carlo (voir transparents de cours), sinon poursuivre à la section qui suit.

10.2 Modélisation AR d'un sifflement

Le but ici est de modéliser votre sifflement à l'aide d'un modèle AR d'ordre 2 puis de reproduire artificiellement ce sifflement.

1. Enregistrez-vous en train de siffler de la manière la plus pure possible et conserver dans `y` les 2 meilleures secondes. Vous pouvez également reprendre le sifflement enregistré lors d'un TP précédent.
2. Décimer le signal afin d'avoir une fréquence d'échantillonnage d'environ 8000Hz (voir la commande Matlab `decimate`) :
   `ydec=decimate(y,R);`

3. Estimer les paramètres d'un modèle AR d'ordre 2 par la méthode des moindres carrés en vous inspirant des commandes suivantes :
```
data=iddata(ydec,[]);
Mar=ar(data,2,'ls');
C=1;
D=Mar.A;
```

4. Former la fonction de transfert du modèle à l'aide de la commande suivante :
`H=tf(C,D,1/fe)`
puis calculer les pôles de H
`pole(H)`
Justifier pourquoi ce modèle AR(2) est capable de modéliser le sifflement.
5. Calculer la prévision à un pas du sifflement à l'aide des commandes suivantes :
```
yhat=predict(Mar,data,1);
N=1000;
stem([ydec(N:N+49) yhat(N:N+49)])
```

6. Ecouter la prévision du sifflement à l'aide de la commande suivante :
`sound(yhat.y,fe/R)`
`fe/R` est la nouvelle valeur de fréquence d'échantillonnage après l'étape de décimation éventuelle.

10.3 Modélisation AR de sons élémentaires

Le but à présent est de modéliser une lettre prononcée (voyelle ou consonne) à l'aide d'un modèle AR puis de reproduire artificiellement cette lettre.

1. Enregistrez-vous en train de prononcer une voyelle (par exemple a ou o) et conserver dans `y` les 2 meilleures secondes.
2. Décimer le signal afin d'avoir une fréquence d'échantillonnage d'environ 8000Hz (voir la commande Matlab `decimate`) :
`ydec=decimate(y,R);`
3. Estimer les paramètres d'un modèle AR d'ordre 8 par la méthode des moindres carrés.
4. Calculer la prévision à un pas de la voyelle à l'aide des commandes suivantes :
```
yhat=predict(Mar,data,1);
N=1000;
stem([ydec(N:N+49) yhat(N:N+49)])
```

5. Ecouter la prévision de la voyelle à l'aide de la commande suivante :
`sound(yhat.y,fe/R)`
`fe/R` est la nouvelle valeur de fréquence d'échantillonnage après l'étape de décimation éventuelle.
6. Tester la procédure de modélisation avec les consonnes m ou n.