Institut Mines-Télécom

Filtrage, transformée de Fourier et échantillonnage sous Matlab

Roland Badeau

TELECOM Paris
IP PARIS

1 Introduction à Matlab et filtrage

1.1 Préliminaires : utilisation de Matlab

Matlab est un langage de programmation et un environnement interactif. On peut exécuter une commande soit en la tapant directement dans le mode interactif soit en exécutant un script qui est une suite de commandes qui se trouvent dans un fichier dont l'extension est "**.m**".

Téléchargez le fichier TP_Intro_Matlab.zip depuis le Moodle du Master ATIAM. Il contient des données et des programmes Matlab. Vous travaillerez dans le répertoire résultant de sa décompression. Pour commencer le TP, il faut lancer le logiciel Matlab.

Ce TP a pour objectif de vous apprendre l'utilisation de Matlab ainsi que d'effectuer des opérations de manipulation simple sur les filtres (systèmes linéaires et invariants). La première partie est tutorielle et peut être exécutée entièrement en mode interactif, mais dès qu'il faudra construire des programmes plus complexes on aura recourt à des scripts afin de gagner en efficacité. Pendant la partie tutorielle, vous êtes encouragés à taper les commandes et à en comprendre le résultat. Faire suivre une commande par un " ;" avant de taper [ENTREE] fait que le résultat du calcul n'est pas visualisé. Si vous voulez voir le résultat il ne faut pas mettre de " ;". De plus le caractère % introduit un commentaire, tout ce qui le suit sur la ligne est ignoré. Exemple

```
3+4; % Vous n'êtes pas obligés de taper les commentaires.
3+4 % ou comment utiliser 10^9 opérations par seconde pour calculer 7
```

1.1.1 Les bases

Dans l'environnement Matlab, toutes les variables sont des matrices dont les entrées sont soit des réels (représentés par un type "double") soit des complexes ($x + iy$, où x et y sont réels). On dit qu'une matrice est de taille $m \times n$ si elle a m lignes et n colonnes. Un **vecteur colonne** est une matrice qui a une seule colonne (de taille $n \times 1$) et un **vecteur ligne** est une matrice qui a une seule ligne (de taille $1 \times n$).

1.1.2 Générer une matrice

Il y a divers moyens de générer une matrice ou un vecteur

- **zeros(m,n)** : Génère une matrice de taille $m \times n$ pleine de zéros.
- **ones(m,n)** : Cette fois-ci la matrice est pleine de 1.
- **a :b :c** : Génère un vecteur ligne commençant à a et incrémenté de b tant que l'on est encore plus petit ou égal à c. Si on tape $a : c$ l'incrément par défaut est 1. Essayez

  ```
  0:3:6
  0:3:7
  1:6
  0:0.1:1
  0:-0.1:1
  1:-0.1:0
  ```
- Entrer directement les valeurs entre crochets :

  ```
  [1 2 3] % vecteur ligne 1 2 3
  [1;2; 3] % vecteur colonne
  [1 2; 3 4] % Une matrice 2 par 2
  [1 2; 3] % UNE ERREUR (voir concaténation plus bas)
  ```

1.1.3 Opérations sur les matrices

- **A+B** : Renvoie la somme des deux matrices qui doivent être de même taille. **Exception** : Si A ou B est un scalaire (une matrice de taille 1×1), il est ajouté à toutes les entrées de l'autre matrices. L'opération "-" fonctionne de la même manière pour la soustraction
```
ones(2,2)+ones(2,2)
[1 2; 3 4]+1 % on ajoute 1 à toutes les entrées.
```
- **A*B** : Renvoie la multiplication des matrices A et B qui doivent avoir des tailles respectives de $m \times n$ et $n \times l$. Le résultat est de taille $m \times l$. Notez que si $m = l = 1$ alors cette opération est un produit scalaire entre le vecteur ligne A et le vecteur colonne B. Il y a la même exception que ci-dessus.
```
[1 2; 3 4]*[-2 1; 1.5 -0.5]
[1 2; 3 4]*[1 0; 0 1]
[1:20]*[1:20]' % A' est la matrice transposée (et conjuguée) de A
(1:3)'*(1:3)
```
- **A.*B** : Multiplication point par point. A et B doivent avoir la même taille. Comparez
```
[1 2; 3 4]*[-2 1; 1.5 -0.5]
[1 2; 3 4].*[-2 1; 1.5 -0.5]
```
- **Fonctions unaires** : Matlab dispose d'un grand nombre de fonctions usuelles prédéfinies telles que : sin, cos, exp, tan, log ...Les appliquer à une matrice signifie appliquer la fonction à chaque élément de la matrice
```
exp(i*pi) % i et pi sont des constantes de Matlab
exp(2*i*pi*0.123*(0:10)) % Onde de Fourier sur Z de fréquence 0.123, entre 0 et 10
real(exp(2*i*pi*0.123*(0:10))) % Partie réelle de l'onde
plot(real(exp(2*i*pi*0.123*(0:10)))) % "plot" affiche son argument, voir plus bas
```
- **sum(v)** : Si v est un vecteur ligne ou colonne, renvoie la somme de ses éléments. Si c'est une matrice, renvoie une ligne pleine de la somme de chaque colonne.
```
sum(ones(1,10))
sum(ones(10,1))
sum(ones(2,3))
sum((1:10).*(1:10)) % somme des carrés des entiers de 1 à 10
```
- **[A B]** : Si A et B sont deux matrices avec le même nombre de lignes, renvoie la concaténation horizontale des deux.
```
[ ones(2,6) zeros(2,4) ones(2,5)] % matrice à deux lignes identiques 6x1,4x0,5x1
```
- **[A ; B]** : A et B ont le même nombre de colonnes, renvoie la concaténation verticale
- **A(:)** : Quelle que soit la forme de A, renvoie un vecteur colonne qui est la concaténation verticale des colonnes de A. Utile pour s'assurer que deux vecteurs ont la même forme (colonne)
```
A=[1 2; 3 4];
A(:)
u=[1 2 2 1]
v=[1 ;3; 4; 5]
u.*v % erreur
u(:).*v(:) %on est sûr de la forme
```
- **x.^ A** : élève le scalaire x à la puissance chacune des entrées de A (fonctionne aussi si A est un scalaire)
```
(-1/2).^(0:10) % les puissances de -1/2 de 0 à 10
```

1.1.4 Les variables

Les variables sont créées lors de leur première affectation

```
h=0:9; % créée une variable nommée h qui contient le vecteur ligne 0, 1 ...9
h=0:8; % Le contenu de h est ecrasé et remplacé par 0,...,8
```

Les indices en Matlab commencent à 1 : **h(1)** est le premier élément du vecteur **h**. **A(2,1)** est l'élément situé à la ligne 2 colonne 1 de la matrice **A**. On peut aussi accéder à une collection de valeurs en indiquant une collection d'indices :

```
h=0:9;
h(1:3)
sum(h(1:10)) % somme tous les éléments de h
sum(h(1:length(h))) % length(h) renvoie la longueur d'un vecteur
sum(h(1:2:length(h))) % somme h(1)+h(3)+h(5)+...
h(length(h):-1:1) % renvoie le vecteur h à l'envers
h(2:4)=ones(1,3) % remplace certaines entrées de h par des 1
```

1.2 Réaliser une convolution et écouter un écho

Dans cette partie vous allez réaliser une convolution entre deux suites. Vous appliquerez cela à un signal sonore pour simuler un écho et écouterez le résultat. Certes, Matlab possède déjà une fonction qui effectue une convolution, mais, pour vous exercer vous en réaliserez une vous même.

1.2.1 Création d'un fichier fonction

Une fonction Matlab définie par l'utilisateur est un fichier de script spécial qui commence par une ligne du type

```
function out=mafonction(a,b,c)
```

La variable "out" sera celle qui contiendra le résultat de "mafonction". a,b et c sont les paramètres de la fonction. Dans une fonction Matlab, vous n'avez pas accès aux variables globales que vous auriez créées dans l'environnement interactif. Le nom du fichier qui contient cette fonction doit être "mafonction.m" et il doit se trouver, soit dans le répertoire courant (commande `cd`) soit dans un répertoire contenu dans le "path" (tapez `help path` pour plus d'informations). Voici un exemple complet de fonction

```
function out=mafonction(a,b,c)
tmp=a+b; %tmp est une variable locale qui disparaitra à la fin de la fonction
out=tmp*c; % la valeur de out à la fin de l'execution est la réponse de "mafonction"
```

1.2.2 Modélisation en vue de la convolution

L'opération de convolution entre deux suites h et u donne une suite, que l'on va noter v définie par

$$v_n = \sum_{m\in\mathbb{Z}} h_m u_{n-m}$$

Sur un ordinateur, et dans Matlab en particulier, on ne peut avoir que des suites à support fini. Ce que nous signifions par "opération de convolution" doit tenir compte de cette limitation. Quitte à décaler les suites, on peut toujours supposer qu'une suite à support fini a son support du type $\{0,\dots,N-1\}$ pour un certain entier N. Si deux suites ont leur support dans $\{0,\dots,N-1\}$ et $\{0,\dots,M-1\}$ respectivement, alors leur produit de convolution a son support dans $\{0,\dots,N+M-2\}$. Ainsi la convolution entre deux vecteurs de taille N et M sera un vecteur de taille $N+M-1$. Nous ferons la convention que, si un vecteur Matlab **h** [1] représente une suite h alors le premier élément du vecteur **h** (i.e. **h(1)** dans Matlab) est la valeur de h_0 et le support de la suite h est supposé être dans $\{0,\dots,$ **length**(**h**) $-1\}$.

1. à partir de maintenant, on écrira en gras des variables Matlab et en italique leur représentation mathématique

1.2.3 Valeur en un point

Avec ces conventions, écrire une fonction qui prend en entrée deux vecteurs **h** et **u** et un entier n et renvoie la valeur de leur convolution en n (i.e. $(h * u)_n$). S'il n'y avait pas de problèmes de bord cela s'écrirait

```
function out=valconv(h,u,n)
a=length(h);
b=length(u);
u=u(:); % s'assurer que u est un vecteur colonne
h=h(:); % s'assurer que h ait la m^
eme forme que u
idx=(0:a-1); %indices pour h (en tant que suite)
out= sum(h(idx+1).*u((n-idx)+1));
```

Après avoir bien compris l'expression dans "sum(...)" proposer une manière de remédier au problème des bords. On remarquera que pour que **h(m+1)** (m sera un élément du vecteur **idx**) intervienne dans la somme sans poser de problème d'indice, il faut et il suffit d'avoir

$$1 \leq m+1 \leq a \text{ pour que } \mathbf{h(m+1)} \text{ existe}$$
$$1 \leq n-m+1 \leq b \text{ pour que } \mathbf{u(n\text{-}m+1)} \text{ existe}$$

En déduire la formule correcte pour le vecteur **idx** en utilisant `min(x,y)` et `max(x,y)` qui renvoient le minimum et le maximum entre x et y.

Testez votre fonction avec

```
valconv(ones(1,3),ones(1,2),0) % doit renvoyer 1
valconv(ones(1,3),ones(1,2),2) % doit renvoyer 2
```

1.2.4 Convolution complète

Modifiez votre fonction pour qu'elle renvoie le vecteur entier de la convolution (on supprime le paramètre n). Pour cela on utilisera une boucle

```
...
for n=0:a+b-2 %La boucle va être parcourue avec n parcourant le vecteur 0,1,...,a+b-2
idx=... % Formule trouvée précédemment
out(n+1)= sum(h(idx).*u((n-idx)+1));
end %fin de la boucle
```

Testez votre fonction avec de petites suites dont vous calculerez la convolution à la main pour vérifier.

1.2.5 Ecoute d'un écho

Charger un son dans Matlab à l'aide de la fonction

```
[y,Fs] = audioread('handel.wav');
```

Cela va charger une variable y qui contient un son. La commande suivante permet d'écouter un son :

```
soundsc(y,Fs) % Fs est la fréquence d'échantillonnage
```

Que donne la commande suivante ?

```
soundsc(y,Fs/2)
```

On peut modéliser un écho par le fait que le son s'ajoute à lui même avec un certain retard, si u représente le signal sonore original, le signal reçu sera du type

$$v_n = u_n + 0.8u_{n-t_1}$$

Ceci signifie que le son rebondit sur un obstacle, perd 20% de son amplitude et arrive retardé de t_1. On constate que cette modélisation revient à réaliser la convolution du son u avec une réponse impulsionnelle h définie par $h_0 = 1$, $h_{t_1} = 0.8$ (tous les autres termes de h sont nuls).

Créer le vecteur **h** correspondant à un retard approximatif de 0,01 seconde (on rappelle que le son utilisé ici est échantillonné à F_s échantillons par seconde). Convoler, avec votre programme, ce vecteur **h** avec le signal sonore **y** (stockez le résultat dans une variable).

À combien estimez-vous le temps de calcul pris par votre programme ?

Utilisez la commande

```
z=conv(h,y);
```

Cette fonction est une fonction Matlab qui fait ce que fait votre programme de convolution. Que constatez-vous sur le temps de calcul ?

Ecoutez le son **z**. (soundsc(z,Fs))

Augmentez le temps de l'écho à 0,1 seconde, quel impression cela donne-t-il ? En termes de distance parcourue par le son, que représentent 0,1 et 0,01 secondes, cela vous aide-t-il à interpréter l'effet de l'écho ?

1.3 Filtre rejecteur

Dans la suite on utilise la commande Matlab filter.

Après avoir consulté l'aide par help filter, dire comment utiliser filter pour effectuer une convo- lution d'un signal contre un RIF (réponse impulsionnelle finie).

Dans la suite nous allons voir comment utiliser filter pour effacer une fréquence parasite dans un son. Pour cela nous créons une version polluée de la variable **y** :

```
n=0:length(y)-1; %le temps discret que dure le signal y
n=n'; %n est colonne
f0=1261;
yb=y+0.1*cos(2*pi*f0/Fs*n); %on ajoute une onde parasite
```

Écouter le son **yb** à la fréquence **Fs**. Quelle est la fréquence en Hz et fréquence réduite de l'onde parasite ?

Donner la TZ de la RI du filtrage qu'effectue la fonction rejette1.m ? Écouter le résultat de ce filtrage sur le signal **yb**. Cela vous paraît-il satisfaisant ?

Donner la TZ de la RI du filtrage qu'effectue la fonction rejette2.m ? Écouter le résultat sur le signal **yb**.

Pour une valeur de **rho** proche de 1, placer les zéros et les pôles des TZ précédentes dans le plan complexe. Utiliser ces positions pour tracer (approximativement) le module des deux TFtD correspondantes (sur l'intervalle [-1/2,1/2[). Conclure quand à la qualité comparée des deux filtrages.

Pourquoi rho (paramètre de rejette2) doit-il toujours être strictement plus petit que 1 ?

En transmettant à la fonction rejette2 une fréquence **f0=1267** et un paramètre **rho** très très proche de 1, conclure au compromis nécessaire dans le choix de **rho**.

2 Transformation de Fourier et échantillonnage

2.1 Préliminaires

Dans cette partie nous illustrons les propriétés de la transformation de Fourier. Nous manipulons la fonction Matlab **fft** pour visualiser des transformées de Fourier et bien en comprendre le fonctionnement. Nous verrons en particulier comment l'algorithme Fast Fourier Transform peut être utilisé pour calculer rapidement des choses qui, a priori, n'ont pas de rapport avec la transformation de Fourier.

Nous abordons ensuite les problèmes que pose l'échantillonnage des signaux, en écoutant l'effet qu'a un sous-échantillonnage sur un son.

2.2 Transformation de Fourier

Sur ordinateur, on ne manipule que des suites finies de nombres. En cours, nous avons vu diverses transformations de Fourier. A priori, seule la TFD (contrairement à la TFTD, TFTC et coefficients de Fourier) peut être calculée sur ordinateur.

On fera la convention que, si h dénote une suite à support fini, alors **h** dénote le vecteur Matlab qui la représente et on a $\mathbf{h}(\mathbf{1}) = h_0$ car Matlab indexe les éléments d'un tableau à partir de 1.

Si **h** est vecteur Matlab et M un entier. Alors :

```
fft(h); % renvoie un tableau de même taille que h et qui en est la TFD
fft(h,M);% Si M est plus grand que la taille de h, renvoie la TFD de h complété par
%des zéros pour atteindre la taille M.
% Si M est plus petit que la taille du vecteur h alors: renvoie la TFD
% du vecteur h(1:M) (i.e. le début de h)
```

On rappelle que la TFD d'une séquence finie $h_0, ..., h_{N-1}0...$ où l'on a complété la séquence h_n par des zéros pour obtenir un signal de taille M est exactement l'échantillonnée de la TFTD de la séquence $h_n,\ n \in Z$ aux fréquences k/M pour k allant de 0 à $M-1$.

Il ne faut pas confondre FFT et TFD. La FFT est un algorithme rapide de calcul de la TFD. Matlab fait cette confusion, c'est dommage. L'algorithme de la FFT est particulièrement rapide pour des vecteurs de taille une puissance de 2. Il s'exécute en un temps proportionnel à $N\log(N)$ où N est la taille du signal.

— Créer deux vecteurs de longueur 100 qui sont deux ondes de Fourier de fréquence 0.1 et 0.123. Et observer leur TFD par la commande **plot(abs(fft(x)));**[2] (**x** est le vecteur). Quelles différence remarquez-vous entre les deux cas ? Pourquoi ? (souvenez-vous de la différence entre une onde de Fourier sur $\mathbb{Z}$ et une onde de Fourier sur $\{0, \ldots, N-1\}$)

— Pouvez-vous expliquer la position exacte du pic dans TFD du signal de fréquence 0.1 (=10/100) ? (en cliquant sur l'icône jaune+croix noire vous pouvez ensuite inspecter la courbe tracée). Expliquer aussi la position du second pic qui apparaît dans le cas où l'on trace la TFD de la partie réelle de l'onde de fréquence 0.1 (**plot(abs(fft(real(x))))**). Pour un signal réel, il est inutile de tracer toute la TFD, seule la moitié de celle-ci suffit le reste se déduisant par symétrie hermitienne.

— La commande **plot(x,y)** trace la courbe représentant les **y** en fonction de **x**. Pour avoir un axe des abscisses bien indexé lors de la visualisation d'une TFD il convient de faire

```
t=0:99; %vecteur',N-1 pour N=100
nu=0.1; %fréquence
onde=exp(2*i*pi*nu*t); %création de l'onde de fréquence nu
plot((0:99)/100,abs(fft(onde))); %tracé correctement indexé de la TFD
```

— Vérifier sur des exemples simples que le produit de convolution est bien transformé en multiplication en Fourier (exemple, la séquence [1 2 3] convoluée avec [1 1 0] dont on calcule la convolution circulaire à la main). Vérifier également que la multiplication par une onde de Fourier conduit à une translation de la TFD.

— Enfin, la fonction **ifft(y)** réalise la TFD inverse de y. Que donnent les commandes suivantes ?

```
ifft(fft([1 1 1]))
ifft(fft([1 1 1],4))
ifft(fft([1 2 3]).*fft([1 1 0]))-[4 3 5]
```

Comme vu en cours la formule de TFD inverse est très proche de celle de la TFD. L'algorithme qui la calcule est aussi rapide que la FFT.

— **Simuler une convolution sur $\mathbb{Z}$ grâce à une convolution circulaire :** Si $h_0, \ldots, h_{N-1}$ et $g_0, \ldots, g_{N-1}$ sont deux séquences de taille N et si **h** et **g** sont les vecteur (ligne) Matlab qui les représentent. Que donne la séquence Matlab suivante ? (dans la variable out)

```
N=length(h);% supposé égale à length(g)
hz=[h zeros(1,N)];
gz=[g zeros(1,N)];
out=ifft(fft(hz).*fft(gz)); %TFD inverse de produit de TFD
out=out(1:2*N-1);
```

Cette méthode permet en particulier de calculer le produit de deux polynôme de degré $N-1$ en un temps proportionnel à $N\log(N)$alors qu'il faudrait de l'ordre de N^2 opérations en appliquant simplement la formule qui donne les coefficients du polynôme produit en fonction des coefficients des polynômes d'origine.

2.3 Fréquence en Hz, fréquence réduite

Si $h_n = f(nT_e)$ est le résultat de l'échantillonnage d'une fonction f à la fréquence $F_e = 1/T_e$ alors une onde de Fourier sur $\mathbb{R}$ de fréquence ν va donner

$$h_n = e^{2i\pi\nu nT_e} = e^{2i\pi\frac{\nu}{Fe}n}$$

soit une onde de Fourier à la fréquence $\frac{\nu}{Fe}$. Ainsi, la bonne normalisation pour l'axe des abscisses, si on connaît la fréquence d'échantillonnage F_e , est

```
Fe=1000; Te=1/Fe; t=(0:299)*Te; %fréquence d'échant. 1000Hz. t qui représente le temps
% pour 300 échantillons
nu=100; %fréquence en Hertz
```

2. on peut aussi utiliser **stem** à la place de **plot** pour de petits signaux

```
h=cos(2*pi*nu*t);% équivalant à cos(2*pi*(nu/Fe)*(0:299))
fh=fft(h);
fh=fh(1:150); % comme h est réel inutile de garder plus de la moitié de la TF
plot((0:149)/300*Fe, abs(fh)); % on indexe en vraies fréquences.
% la position 150 dans la TFD donne Fe/2=500Hz, ici.
```

Générer une onde de fréquence 0,06 et longue de 10000 échantillons ($n \mapsto e^{2i\pi\nu n}$, ν est la fréquence). Écoutez sa partie réelle(real(u)) par

```
soundsc(real(u),8000) % Le son est joué à 8000 échantillons par seconde
soundsc(real(u),4000)% le son est joué à 4000 échantillons par seconde
```

Quelles fréquences en Hz (1Hz=1/seconde) percevez-vous dans les deux cas ?

On dit que la fréquence en Hz est la fréquence réelle (ce qui nécessite de connaître la fréquence d'échantillonnage) alors que la fréquence 0,06 est la fréquence réduite (qui est toujours, pour un signal défini sur $\mathbb{Z}$, entre -1/2 et 1/2).