TP1 : Synthèse de filtres numériques

– Vous devez avoir lu entièrement le sujet de TP avant la séance.
– Les questions dont les numéros sont suivis du signe † sont des questions théoriques, qui doivent être résolues avant la séance de TP.
– La préparation de TP, et en particulier les réponses aux questions théoriques seront vérifiées et notées en début de séance.
– A la fin de séance, il faut rendre un compte-rendu qui contient les réponses aux questions posées dans le sujet, les explications et les justifications des résultats des manipulations, et le listing des programmes.
– Le TP est noté à partir de la préparation, du travail fait pendant la séance, et du compte-rendu.

I INTRODUCTION

L'objectif de cette manipulation est de synthétiser et de mettre en œuvre un filtre numérique dans le but d'extraire un signal noyé dans un bruit.

Avant d'arriver en séance, vous devez impérativement avoir assimilé les notions théoriques concernant :
– la représentation spectrale des signaux discrets et en particulier la Transformée de Fourier Discrète ;
– la synthèse des filtres à réponse impulsionnelle finie (RIF) et à réponse impulsionnelle infinie (RII) ;
– le filtrage des signaux et la représentation spectrale de cette opération.
De même, vous devez avoir étudié les documents concernant l'utilisation de MATLAB et de sa boîte à outils *traitement du signal*, fournis en annexe.

Le signal à traiter est issu d'un capteur dont le faible rendement ne permet pas d'obtenir un bon rapport signal sur bruit. Le schéma d'acquisition du signal est donné Figure 1.

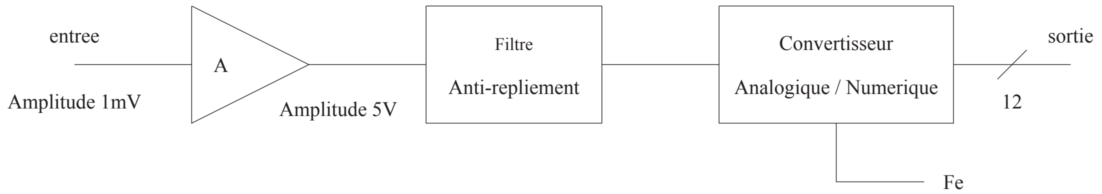


Figure 1 – Manipulation

La fréquence d'échantillonnage est $F_e = 8$ kHz et l'amplitude du signal est quantifiée sur 12 bits. Le résultat de l'acquisition est stocké dans un fichier appelé `fichsig.mat`.

Tout naturellement, votre travail se décompose en trois étapes :
– l'analyse du signal à filtrer ;
– la synthèse du filtre numérique ;
– le filtrage du signal.
Pour ces trois étapes vous utiliserez le logiciel MATLAB et sa boîte à outils *traitement du signal* (*signal processing toolbox*).

II ANALYSE DU SIGNAL

A l'aide de la commande MATLAB `load fichsig.mat` vous allez récupérer le signal sous la forme d'un vecteur nommé `sig`.

A l'aide du logiciel MATLAB :

1. Tracez la représentation temporelle du signal.
2. Tracez la représentation spectrale du signal.

A partir de la représentation spectrale en échelle logarithmique du signal, sachant que le bruit est de type passe-haut, délimitez le domaine fréquentiel correspondant au signal utile.

III SYNTHÈSE DE FILTRES

Afin d'éliminer le bruit on va effectuer le filtrage du signal. On requiert que les caractéristiques fréquentielles du filtre soient les suivantes :
– la bande de transition entre la bande passante et la bande atténuée doit être de l'ordre de 1 kHz ;
– l'atténuation en bande atténuée doit être supérieure à 72 dB ;
– l'attenuation en bande passante doit être inférieure à 1 dB.

III.1 Filtres à réponse impulsionnelle finie (RIF)

Pour synthétiser des filtres à réponse impulsionnelle finie, on utilisera la méthode consistant à échantillonner et tronquer la réponse impulsionnelle du filtre numérique idéal. L'ordre de filtres n'étant pas connu à l'avance, on procédera par essais successifs pour déterminer l'ordre du filtre répondant au gabarit.

1. † Donnez l'expression littérale de la fonction de transfert du filtre numérique idéal. En déduire sa réponse impulsionnelle.
2. Avec Matlab, tracez cette réponse impulsionnelle en la tronquant sur un intervalle $[-P, P]$. Quel est l'ordre (= le nombre de coefficients) du filtre RIF ainsi obtenu ?

3. Pondérez cette réponse impulsionnelle par les fenêtres rectangulaire et de Blackman.
4. Pour ces deux fenêtres et pour les ordres choisis, tracez les réponses en fréquence correspondantes (cf. la fonction Matlab **freqz**). Par différents essais successifs, déterminez quels sont les ordres des filtres nécessaires pour satisfaire le gabarit pour ces deux fenêtres. Choisissez le filtre qui vous paraît être le plus efficace et le plus intéressant en vous justifiant.

Le résultat idéal du filtrage est le signal non bruité. Afin de rendre possible les comparaisons, celui-ci est accessible dans le vecteur `sig_ideal` qui a été chargé lors de la commande `load fichsig.mat`.

5. Filtrez le signal à l'aide du filtre choisi et la fonction Matlab **filter**. Tracez la représentation temporelle du signal de référence et du signal filtré.
6. Expliquez les différences, en particulier pour l'origine des temps. Quel est le retard observé ? Comment peut-on calculer ce retard à partir de la valeur de P et la période d'échantillonnage T_e ?

III.2 Filtres à réponse impulsionnelle infinie (RII)

Pour synthétiser des filtres à réponse impulsionnelle infinie, on utilisera la méthode de discrétisation d'un filtre analogique par la transformation bilinéaire (voir Section 3.3 du document “Matlab pour le traitement du signal”, fourni en annexe).

1. A partir du gabarit du filtre numérique, déterminez la gabarit du filtre analogique à synthétiser.
2. Calculez l'ordre du filtre analogique de type Butterworth satisfaisant le gabarit.
3. En utilisant les fonctions **buttap**, **zp2tf** et **lp2lp**, déterminez les coefficients de ce filtre Butterworth.
4. En utilisant la fonction **bilinear**, discrétisez ce filtre par transformation bilinéaire.
5. Tracez la réponse en fréquence du filtre numérique.
6. Filtrez le signal et répondez aux deux dernières questions de Section III.1.

IV CHOIX DES FILTRES

Vouz avez construit des filtres de structures différentes répondant à un objectif commun. A partir des résultats obtenus, quel type de filtres choisiriez vous dans chacun des cas suivants :

1. Temps de traitement minimal.
2. Déphasage entre les composantes fréquentielles inéxistant.
3. Stabilité inconditionnelle du flitre.

Donnez vos conclusions sur la synthèse et le choix des filtres.

V ANNEXE : Quelques fonctions Matlab utiles

* *freqz* : Digital filter frequency response.
[H,W] = FREQZ(B,A,N) returns the N-point complex frequency response vector H and the N-point frequency vector W in radians/sample of the filter :

$$H(e^{jw}) = \frac{B(e^{jw})}{A(e^{jw})} = \frac{b(1) + b(2)e^{-jw} + \cdots + b(m+1)e^{-jmw}}{a(1) + a(2)e^{-jw} + \cdots + a(n+1)e^{-jnw}}$$

given numerator and denominator coefficients in vectors B and A. The frequency response is evaluated at N points equally spaced around the upper half of the unit circle. If N isn't specified, it defaults to 512.

[H,F] = FREQZ(B,A,N,Fs) returns frequency vector F (in Hz), where Fs is the sampling frequency (in Hz).

* *fft* : Discrete Fourier transform.
FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension.

* *blackman* : Blackman window.
BLACKMAN(N) returns the N-point symmetric Blackman window in a column vector.

* *filter* : One-dimensional digital filter.
Y = FILTER(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y. The filter is a "Direct Form II Transposed" implementation of the standard difference equation :

$$a(1)*y(n) = b(1)*x(n)+b(2)*x(n-1)+...+b(n_b+1)*x(n-n_b)-a(2)*y(n-1)-...-a(n_a+1)*y(n-n_a)$$

TP2 : Réalisation d'un démodulateur stéréo

Les questions dont les numéros sont suivis du signe ' † ' sont des questions théoriques, qui doivent être résolues avant la séance de TP.

But de la manipulation

Pour réaliser un enregistrement stéréophonique, il faut utiliser deux microphones placés respectivement à droite et à gauche de la scène. Lors de la restitution, deux haut-parleurs sont chargés de reproduire le signal de chacun des deux canaux, droite et gauche.
Pour la radiophonie, il faut transmettre simultanément, et par le même canal de transmission, les deux signaux enregistrés par les deux microphones (notés $g(t)$ et $d(t)$ dans la suite). La contrainte principale pour une telle transmission est que le signal doit tout de même pouvoir être reçu par des récepteurs monophoniques (équipés d'un seul haut-parleur). Pour cela, on construit un signal composé :

- du signal correspondant à la somme des signaux enregistrés par deux microphones ($x_1(\mathrm{t})=g(t)+d(t)$),
- du signal correspondant à la différence de ces signaux ($x_2(\mathrm{t})=g(t)-d(t)$), modulé en fréquence, c'est-à-dire multiplié par une porteuse (cosinus de fréquence f_0),
- de la porteuse (avec une fréquence divisée par 2) ayant servi à moduler ce signal afin de permettre une démodulation cohérente.

Le signal transmis est donc noté :

$$y(t) = x_1(t) + x_2(t)\cos(2\pi f_0 t) + \cos\left(2\pi \frac{f_0}{2} t\right)$$

L'objectif de ce TP est de restaurer, dans les récepteurs, les signaux $x_1(t)$ et $x_2(t)$ à partir du signal transmis $y(t)$. Une fois cette opération effectuée, un récepteur monophonique pourra alimenter son haut-parleur avec le signal $x_1(t)=g(t)+d(t)$, tandis qu'un récepteur stéréophonique pourra reproduire les signaux $g(t)$ et $d(t)$ à partir des signaux $x_1(t)$ et $x_2(t)$.

I. Schéma du démodulateur

1.† Utiliser la propriété de translation en fréquence de la transformée de Fourier (rappelée en annexe) pour calculer la transformée de Fourier théorique du signal transmis $y(t)$ en fonction des transformées de Fourier des signaux $x_1(t)$ et $x_2(t)$.
L'allure spectrale du signal transmis $y(t)$ est montrée en Fig. 1 (pour les fréquences positives).

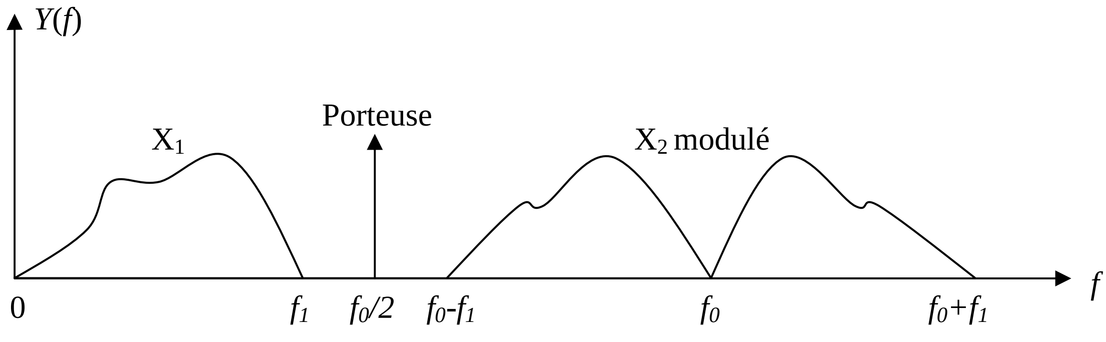


Figure 1 : Allure du spectre du signal transmis

Le schéma bloc du démodulateur, qui permet de récupérer les signaux x_1 et x_2 à partir du signal transmis *y*, est montré en Figure 2.

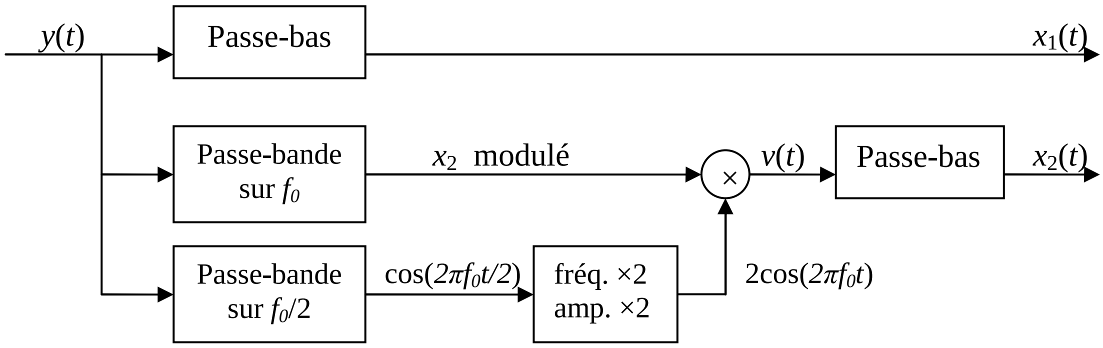


Figure 2 : Schéma du démodulateur

2.† Montrer théoriquement que ce système arrive à restaurer les signaux x_1 et x_2 en sortie.

Dans ce TP, la démodulation sera effectuée avec un système numérique, recevant en entrée le signal à temps discret $y[n]=y(nT_e)$ et fournissant en sortie les signaux $x_1[n]= x_1(nT_e)$ et $x_2[n]= x_2(nT_e)$ où T_e est la période d'échantillonnage. Par conséquent, les filtres utilisés sont des filtres numériques.

II. Analyse du signal

Vous disposez dans le fichier multi.mat, le signal $y[n]$ échantillonné à la fréquence Fe=132300 Hz.

3. Avec Matlab, tracez la représentation fréquentielle de ce signal. Voyez-vous les trois composantes ? Quelles sont les fréquences f_0 et f_1 définies en Figure 1 ?

4. A partir de cette représentation fréquentielle, déterminez les gabarits des 4 filtres utilisés dans le démodulateur. Pour chacun des filtres, on autorise une bande de transition de largeur 2 kHz, une atténuation maximale en bande passante de 1 dB et une atténuation minimale en bande coupée de 40 dB.

III. Récupération du signal $x_1(t)$

Pour récupérer le signal $x_1(t)$, il suffit de filtrer le signal $y(t)$ par un filtre passe-bas. La fonction Matlab fir1, décrit en annexe, permet de synthétiser des filtres RIF passe-bas, passe-haut, passe-bande et coupe-bande, par le fenêtrage de la réponse impulsionnelle d'un filtre idéal, étudié au premier TP.

5. Synthétiser ce filtre passe-bas avec la fonction fir1. Les paramètres du filtre n'étant pas connus à l'avance, on procédera par essais successifs pour déterminer les paramètres du filtre d'ordre le plus faible -approximativement- satisfaisant le gabarit (comme dans le premier TP).

6. Préciser l'ordre et en déduire le retard induit par le filtre (voir le TP1).

7. En utilisant la fonction Matlab filter, filtrer le signal $y(t)$ avec le filtre synthétisé. Tracer la représentation fréquentielle du signal filtré.

8. Ecouter le signal filtré avec la fonction wavplay de Matlab. Qu'est-ce que vous entendez?

IV. Récupération de la porteuse

9. Avec la fonction fir1, synthétisez un filtre passe-bande, centré sur $f_0/2$, permettant de restituer $\cos(2\pi f_0 t/2)$. Préciser l'ordre et en déduire le retard induit par le filtre.

10. Filtrer le signal $y(t)$ avec le filtre synthétisé. Tracer les représentations temporelle et fréquentielle du signal filtré.

11. Proposer une méthode pour doubler sa fréquence et obtenir ainsi $\cos(2\pi f_0 t)$. Tester votre méthode. (*Remarque* : $\cos^2(a)=[1+\cos(2a)]/2$).

V. Récupération du signal $x_2(t)$

Pour récupérer le signal $x_2(t)$, on filtre d'abord le signal $y(t)$ par un filtre passe-bande pour supprimer $x_1(t)$ et la porteuse.

12. Avec la fonction fir1, synthétisez un filtre passe-bande, centré sur f_0, permettant de réaliser cette opération. Préciser l'ordre et en déduire le retard induit par le filtre.

13. Filtrer le signal $y(t)$ avec le filtre synthétisé. Tracer la représentation fréquentielle du signal filtré.

Pour ramener le signal $x_2(t)$ en bande de base, on multiplie ensuite le signal filtré par $2\cos(2\pi f_0 t)$ obtenu ci-dessus, puis on filtre le résultat par un passe-bas (peut-on utiliser le même filtre passe-bas que celui synthétisé dans la partie III ? Justifiez votre réponse).

14. Réaliser ces opérations. Préciser l'ordre du filtre passe-bas et en déduire le retard induit par le filtre. Tracer la représentation fréquentielle du signal intermédiaire $v(t)$ (avant le filtrage passe-bas) et celle du signal $x_2(t)$ obtenu en sortie.

15. Ecouter le signal $x_2(t)$. Qu'est-ce que vous entendez?

VI. Récupération des signaux $g(t)$ et $d(t)$

On se rappelle que $x_1(t)=g(t)+d(t)$ et $x_2(t)=g(t)-d(t)$ où $g(t)$ et $d(t)$ sont les signaux enregistrés par les microphones gauche et droite.

16.† Proposer une méthode pour retrouver $g(t)$ et $d(t)$ à partir de $x_1(t)$ et $x_2(t)$.

17. Reconstruire les signaux $g(t)$ et $d(t)$ avec cette méthode. Ecouter ces signaux. Vous devriez entendre « droite » pour le signal $d(t)$ et « gauche » pour le signal $g(t)$. Qu'est-ce que vous entendez?

18. Avez-vous pris des précautions particulières quant au retard induit sur les signaux par les différents filtres ? Si non, proposer une solution pour avoir le même retard sur les différents

chemins (sinon les signaux x_1 et x_2 seront déphasés). Tester votre solution en écoutant les résultats.

VII. Questions supplémentaires

19. Est-ce qu'on aurait pu utiliser des filtres RII dans ce système ?

20. Est-ce que le filtre passe-bande sur f_0 est indispensable ?

21. Pourquoi l'information sur la porteuse est-elle située à $f_0/2$ au lieu de f_0 ?

Annexe A : Propriété de translation en fréquence de la transformée de Fourier

Si TF$\{x(t)\}=X(f)$, et sachant que TF$\{\cos(2\pi f_0 t)\}=[\delta(f - f_0)+\delta(f + f_0)]/2$, on peut alors montrer que TF$\{x(t)\cos(2\pi f_0 t)\}=[X(f - f_0)+ X(f + f_0)]/2$.

Annexe B : Fonction matlab fir1

FIR1 FIR filter design using the window method.

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B.
The cut-off frequency Wn must be between 0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. The filter B is real and has linear phase. The normalized gain of the filter at Wn is -6 dB.

B = FIR1(N,Wn,'high') designs an N'th order highpass filter. You can also use B = FIR1(N,Wn,'low') to design a lowpass filter.

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2. You can also specify B = FIR1(N,Wn,'bandpass'). If Wn = [W1 W2], B = FIR1(N,Wn,'stop') will design a bandstop filter.

B = FIR1(N,Wn,WIN) designs an N-th order FIR filter using the N+1 length vector WIN to window the impulse response.
If empty or omitted, FIR1 uses a Hamming window of length N+1.

Eviter au maximum les boucles, *e.g.*,

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

est équivalent à

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

Autre exemple :

```
>> r=1 : 10 ; A=[] ; % init. de A à vide
>> for k=1 : 5, A=[A ; r] ; end
```

peut être écrit

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

et même

```
>> r=1 : 10 ; A=r([1 1 1 1 1], :) ;
```

ou

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

Attention à ne pas utiliser `i` ou `j` comme indice de boucle car `i` et `j` prédéfinis à $\sqrt{-1}$.

6 Signal et image

6.1 Matrices particulières

Matrices de Tœplitz :

```
>> c=[ 1 2 3 ] ; l=[ 1.5 2.5 3.5 4.5 5.5 ] ;
>> toeplitz(c,l)
   ans = 1.0 2.5 3.5 4.5 5.5
         2.0 1.0 2.5 3.5 4.5
         3.0 2.0 1.0 2.5 3.5
```

Matrices de Hankel, Hadamard, Hilbert, *etc...*

6.2 Filtrage et convolution

Fonction `filter` (2D : `filter2`) : filtrage rationnel par un filtre de coefficients dans `A` et `B` :

```
>> Y = filter(B,A,X) ;
```

Convolution `conv` (2D : `conv2`) :

```
>> Z = conv(X,Y) ;
```

La fonction `roots` : racines (zéros, pôles). Calcul des racines d'un polynôme donné par ses coefficients (ici $P(z) = z^3 - 6z^2 - 72z - 27$) :

```
>> p = [1 -6 -72 -27] ; r=roots(p)
```

La fonction `polyval` calcule la valeur d'un polynôme en certains points :

```
>> polyval(p,[ 1, exp(j*pi/4) ])
   ans = 1.0e+002 *
   -1.0400 -0.7862-0.5620i
```

La fonction `poly` inverse la fonction `roots`.

```
>> p=poly(r)
   p = 1.000 -6.000 -72.000 -27.000
```

6.3 Transformation de Fourier

`fft`, `ifft`. Calcul de la FFT du signal `x`, sur 1024 points régulièrement distribués dans $[0,1]$:

```
>> TF=fft(x,1024) ;
```

Pour $[-0.5,+0.5]$ utiliser `fftshift`. En dimension 2 : `fft2` et `ifft2`.

6.4 Un peu d'aléa

`rand` : génération pseudo-alétoire uniforme sur $[0,1]$
`randn` : génération gaussienne $\mathcal{N}(0,1)$
`mean` : moyenne empirique
`std` : écart-type empirique
`xcorr` : corrélation empirique
`cov` : covariance empirique

6.5 Fonctions mathématiques spéciales

`besselj`, `bessely`, `gamma`, `erf`, `erfinv`, *etc...*

6.6 Exemple de fichier de commande

Problème : On possède N mesures (x_n, y_n) bruitées censées correspondre à des points d'une droite. On va estimer la pente a et la valeur à l'origine b de cette droite par moindres carrés, c'est-à-dire minimisant :

$$J(a,b) = \sum_{k=1}^{N} (y_k - a x_k - b)^2$$

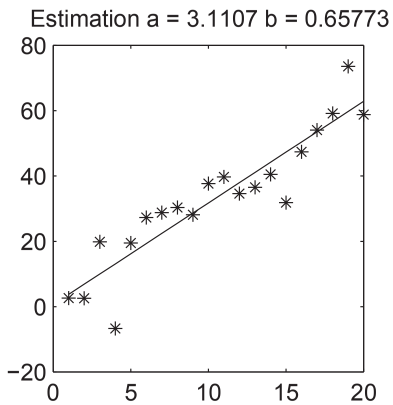
Solution : En annulant les dérivées partielles par rapport à a et b on trouve (un seul minimum) :

$$a = \frac{\sum_{k=1}^N x_k y_k - \frac{1}{N}(\sum_{k=1}^N x_k)(\sum_{k=1}^N y_k)}{\sum_{k=1}^N x_k^2 - \frac{1}{N}(\sum_{k=1}^N x_k)^2}$$

$$b = \frac{1}{N}(\sum_{k=1}^N y_k - a \sum_{k=1}^N x_k)$$

Script Matlab :

```
% Simulation
   a = pi ; b = 1/3 ;
% Tirage d'un ensemble de mesures
   N = 20 ; x = 1 :N ;
   bruit = 10*randn(1,N) ; y = a*x+b+bruit ;
% Estimation
   aest = (sum(x.*y) - sum(x)*sum(y)/N)/...
        (sum(x.^2) - sum(x)^2/N) ;
   best = (sum(y) - aest*sum(x))/N ;
% Affichage
   plot(x,y,'*',x,aest*x+best)
   title(['Estimation a=',num2str(aest),...
        ' b=',num2str(best)])
```



— Matlab en bref —

Table des matières

1 Généralités

1.1 Introduction

Matlab pour « Matrix Laboratory » : **tout est matrice**. Environnement de calcul matriciel, adapté à l'automatique et au traitement du signal et de l'image : facilité d'emploi, possibilités graphiques, boites à outils : signal processing, image processing, optimization, *etc...*

1.2 Démarrer

Cliquez sur l'icône Matlab ou tapez `matlab` dans un shell. Tapez les commandes Matlab ou le nom d'un fichier de commandes à la suite des chevrons `>>`.

1.3 Aide en ligne

Commande `help` : `help NomFct`. Également `lookfor`. Avant d'écrire une fonction, assurez-vous qu'une fonction similaire n'existe pas déjà. De même, avant d'utiliser une fonction, vérifiez qu'elle réalise bien l'opération souhaitée. Vérifiez également sa syntaxe. En cas d'erreur, lisez le message d'erreur que Matlab a la gentillesse de vous indiquer.

1.4 Langage interprété

Pas de compilation, ni de déclaration de variable, ni de réservation mémoire. Résultat affiché et stocké dans la variable `ans`.

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

Fonctions usuelles : `sin`, `cos`, `exp`, `log`, *etc...*

1.5 Variables

Les noms de variable commencent par une lettre. Attention, Matlab fait la différence entre `X` et `x`.

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

Point virgule « `;` » évite l'affichage du résultat. Plusieurs commandes par ligne : séparées par « `;` » ou « `,` » :

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

1.6 Variables complexes

Matlab gère réels et complexes. `i` et `j` initialisés à $\sqrt{-1}$.

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

Fonctions prédéfinies : `real`, `imag`, `abs`, `angle`, *etc...*

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

1.7 Vecteurs, matrices et manipulations

Matrice sous forme d'un tableau :

```
>> A = [ 1 2 3; 4 5 6 ]
   A = 1 2 3
       4 5 6
```

Expressions à évaluer possible :

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

Éléments référencés par leurs indices :

```
>> x(2)
   ans = 1.7321
>> x(5) = abs(x(1))
   x = -1.300 1.732 4.800 0.000 1.300
```

Ajout de lignes à une matrice :

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

Ajout de colonnes : « `,` » à la place de « `;` »

Fonctions `zeros`, `ones`, et `eye` :

```
>> eye(2) % eye comme I : Identity
   ans = 1 0
         0 1
>> x = ones(1,5)
   ans = 1 1 1 1 1
```

Taille d'une matrice : `size`, `length` :

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

Transposée, conjuguée, retournement, rotation :

```
>> A' % Transposée conjuguée de A
>> A.' % Transposée de A
>> flipud(A) % Up - Down
>> fliplr(A) % Left - Right
>> rot90(A) % Rotation de 90°
```

1.8 L'opérateur d'énumération « : »

Pas à pas, incrément :

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

Incrémentation par défaut : 1 (voir aussi `linspace`) :

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

Sélection d'éléments :

```
>> A(1,3) ; A(1,1 : 3) ; A( : ,3) ;
```

Si on a :

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

alors, extraction lignes 1 à 2 et toutes les colonnes :

```
>> A(1 : 2, : )
   ans = 1 2 3
         4 5 6
```

1.9 Chaînes de caractères

Variables contenant des chaînes de caractères :

```
>> mess = 'Bienvenue sur Matlab' ;
```

Manipulations de même type que pour les vecteurs :

```
>> mess = [mess ' v 5']
   mess = Bienvenue sur Matlab v 5
```

Conversion de nombres en chaînes de caractères : `num2str`, `int2str`, `sprintf` :

```
>> mess = ['pi vaut ', num2str(pi)]
   mess = pi vaut 3.142
```

Évaluation d'une chaîne : `eval` et `feval` :

```
>> nomvar = 'x' ;
>> eval([nomvar '1 = sqrt(-1)'])
   x1 = 0.0000 + 1.0000i
>> NomFct = 'sin' ;
>> feval(NomFct,pi)
   ans = 1.2246e-16 % sin(pi) 0 !
```

2 Opérations matricielles

Opérations usuelles définies de façon naturelle :

```
>> 2*A
>> A*B % Produit matriciel
>> A^p
>> inv(A)
```

Résolution de systèmes linéaires :

```
>> X = A\B % solution de A*X = B
>> X = B/A % solution de X*A = B
```

Attention :

```
>> A.*B      % Produit terme à terme
>> X = A./B  % Division terme à terme
>> A.^p      % Puissance terme à terme
```

Autres fonctions utiles sur les matrices :

```
>> poly(A)       % Polyn. caract. de A
>> det(A)        % Déterminant de A
>> trace(A)      % Trace de A
>> [V,D] = eig(A) % Val. et vect. propres
```

Matrices creuses : gain en mémoire et en coût de calcul

```
>> A = eye(2) ; B = sparse(A) ;
>> A*[1 ;1] ; % 4 multiplications
>> B*[1 ;1] ; % 2 multiplications
```

Pour certaines fonctions, matrice = tableau de valeurs :

```
>> exp(A) ; log(A) ; sqrt(A) ;
>> round(A) ; sign(A) ; rem(A,2) ;
```

Pour d'autres, matrice = suite de vecteurs colonnes

```
>> min(A) ; % Minima des colonnes
>> max(A) ; % Maximum
>> sum(A) ; % Somme
>> prod(A) ; % Somme
>> cumsum(A) ; % Sommes cumulés
>> cumprod(A) ; % Produits cumulés
>> sort(A) ; % Tri des colonnes
>> median(A) ; % Val. médiane des col
```

Décomposition de matrices :

```
>> [U,S,V] = svd(X,0) ; % Valeurs singulières
>> R = chol(X) ; % Cholesky
>> [Q,R] = qr(X) ; % QR
>> [L,U] = lu(X) ; % LU (Lower, Upper)
```

3 Affichages

3.1 Affichage alpha-numérique

Façon Matlab ou façon C

```
>> disp(mess)
   pi vaut 3.142
>> fprintf('pi vaut %1.3e \n',pi)
   pi vaut 3.141e+000
```

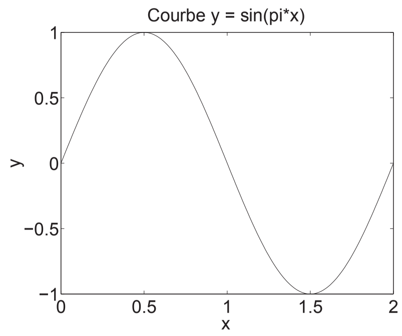
Saisie d'une valeur :

```
>> rep=input('Valeur de lambda : ') ;
```

3.2 Graphiques 1D : plot

Tracé d'une fonction avec axes et titre :

```
>> x = (0 : 0.1 : 2) ; y = sin(x*pi) ;
>> plot(x,y) ; % abscisse, ordonnée
>> title('Courbe y = sinus(pi*x)')
>> xlabel('x') ; ylabel('y')
```

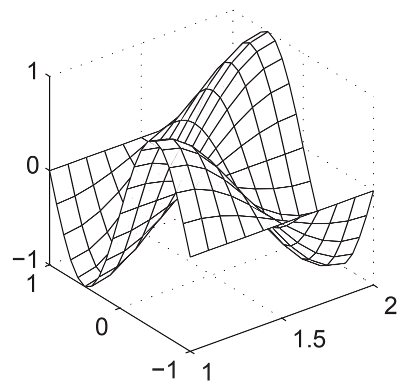



3.3 Graphiques 2D : mesh

Pour une fonction de deux variables :

```
>> x = 1 : 0.1 : 2 ; y = -1 : 0.1 : 1 ;
>> [X,Y] = meshgrid(x,y) ;
>> mesh(X,Y,cos(pi*X).*sin(pi*Y))
```

Mais aussi `meshc`, `contour`, `image`, `surf`, *etc.*..



3.4 Affichage de plusieurs courbes

`subplot` divise la fenêtre graphique :

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

1	2
3	4

La commande `figure` crée une nouvelle fenêtre graphique. `figure(2)` permet d'adresser la figure n° 2.

4 Fichiers, scripts, fonctions

4.1 Répertoire de travail

Commandes `cd` et `dir` (*idem* DOS) ; `cd` et `ls` (*idem* Unix).

4.2 Sauvegarde et chargement

Sauvegarde :

```
>> save NomFichier NomsVariables
```

Exemple :

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

Par défaut sauvegarde dans `matlab.mat`.
Chargement :

```
>> load NomFichier
```

Voir aussi les commandes `who`, `whos`, `pack` et `clear`.

4.3 Scripts

Script : suite de commandes dans un fichier. Commentaires par : « % ». Nom de fichier avec extension « .m » (`nomfich.m`). Exécution sous Matlab : nom du fichier, sans « .m ». Variables de l'espace de travail visbibles dans le script et réciproquement.

4.4 Fonctions

Définition :

```
function y = sinuscardinal(x)
   z = sin(x) ; % Variable de stockage
   y = z./x ; % Variable de sortie
```

Appel identique aux fonctions Matlab :

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

Autre exemple :

```
function [min, max] = minetmax(x)
   min = min(x) ; max = max(x) ;
```

Appel :

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

Passage par valeur seulement. Nombres d'arguments en entrée et en sortie acessibles dans `nargin` et `nargout`. Variables de l'espace de travail invisible dans les fonctions et réciproquement. Outils de mise au point (déboggage) : plus simple : `keyboard` ; plus évolués : débogueur intégré...

5 Boucles et contrôles

5.1 Test « if » :

Structure générale :

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

Opérateurs logiques : et (`&`), ou (`|`, `xor`), égal (`==`), différent (`~=`), supérieur (`>`, `>=`), inférieur (`<`, `<=`), non (`~`). Fonctions logiques : `exist`, `any`, `find`, `isinf`, `isnan`, *etc.*.. Expression considérée fausse si `0`, et vraie sinon.

5.2 Boucle « for » :

Structure générale :

```
for k=1 : 10
   suite d'instructions ;
end
```

– Wn bande passante du filtre (fréquence haute de la bande passante pour un passe-bas ; fréquence basse de la bande passante pour un passe-haut ; fréquences basse et haute de la bande passante pour un passe-bande ; fréquences basse et haute de la bande coupée pour un coupe-bande). Les fréquences de Wn sont normalisées par rapport à la fréquence de Nyquist.

– Rp atténuation maximale (en dB) dans la bande passante.

– Rs atténuation minimale (en dB) dans la bande coupée.

3.3.5 Estimation de l'ordre des filtres

Enfin, Matlab, possède des fonctions permettant d'estimer l'ordre minimal nécessaire pour la construction d'un filtre passe-bas ou passe bande entrant dans un gabarit donné :

```
>> [n, Wn] = buttord(Wp,Ws,Rp,Rs) ;
>> [n, Wn] = cheb1ord(Wp,Ws,Rp,Rs) ;
>> [n, Wn] = ellipord(Wp,Ws,Rp,Rs) ;
```

– Wp bande passante.

– Ws bande coupée.

– Rp atténuation maximale (en dB) dans la bande passante.

– Rs atténuation minimale (en dB) dans la bande coupée.

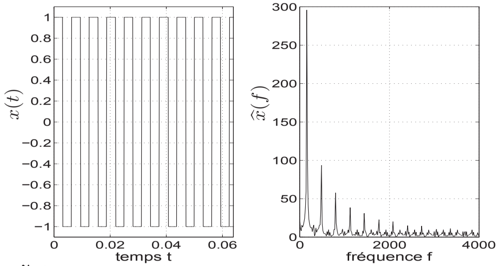
– n ordre du filtre.

– Wn fréquence propre du filtre numérique. Pour un filtre passe-bas Wp et Ws sont les fréquences hautes de la bande passante et basse de la bande coupée. Pour un filtre passe-bande, Wp contient les fréquences basse et haute de la bande passante et Ws les fréquences haute et basse de la bande coupée.
Attention, les fréquences sont normalisées par rapport à la fréquence de Nyquist $=\frac{f_e}{2}$.

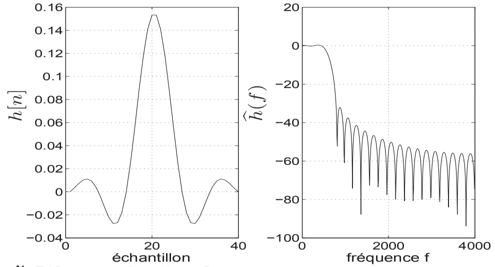
Pour les filtres passe-haut et coupe-bande, leur ordre peut être calculé de la même façon que pour les filtres passe-bas et passe-bande en renversant les fréquences de 0 vers 1 et de 1 vers 0. (e.g. l'ordre d'un passe-haut Wp=0.2, Ws=0.1 est le même que celui d'un passe-bas Wp=0.8, Ws=0.0).

4 Exemple

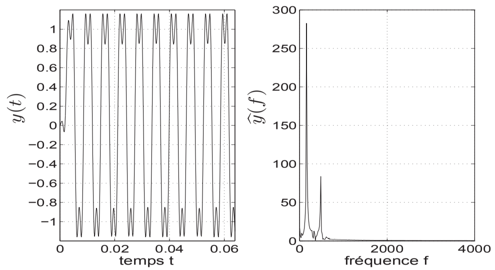
```
% Génération du signal
  Fe = 8e3;
  N = 512;
  t = (0:N-1)/Fe;
  x = square(2*pi*Fe*t/50);
% TFD sur [0, Fe]
  X = fft(x);
  f = (0:N-1)/N*Fe;
% Affichage
  subplot(1,2,1); plot(t,x);
  xlabel('temps t'), ylabel('x(t)');
  subplot(1,2,2); plot(f(1:N/2),abs(X(1:N/2)));
  xlabel('fréquence f'), ylabel('X(f)');
```




```
% Synthèse du filtre passe bas
% (RIF moindres carrés)
% Bande passante [0, 200 Hz]
% Bande coupée [400Hz, 4000Hz]
% Réponse impulsionnelle
  h = firls(39,[0 500 750 Fe/2]/Fe*2,[1 1 0 0]);
% Réponse en frquence
  [H, freq] = freqz(h,1,512,Fe);
% Affichage
  subplot(1,2,1); plot(h);
  xlabel('échantillon'), ylabel('h[n]');
  subplot(1,2,2); plot(freq,20*log10(abs(H)));
  xlabel('fréquence f'), ylabel('H(f)');
```




```
% Filtrage du signal
  y = filter(h,1,x);
  Y = fft(y);
% Affichage
  subplot(1,2,1); plot(t,y);
  xlabel('temps t'), ylabel('y(t)');
  subplot(1,2,2); plot(f(1:N/2),abs(Y(1:N/2)));
  xlabel('fréquence f'), ylabel('Y(f)');
```



— Matlab et le traitement du signal —

Table des matières

Matlab et sa boîte à outils *Signal Processing*, contiennent un grand nombre de fonctionnalités concernant :

– la génération de signaux ;

– la représentation des signaux (Transformée de Fourier Discrète FFT, Transformée en Cosinus Discrets DCT. . .) ;

– l'analyse des signaux (statistique, analyse spectrale paramétrique. . .) ;

– la représentation des systèmes linéaires (fonction de transfert, pôles et zéros, espace d'état. . .) ;

– l'analyse des systèmes (réponse impulsionnelle, réponse en fréquence. . .) ;

– le filtrage et la synthèse de filtres.

Nous nous intéresserons ici uniquement aux fonctions utiles pour la représentation fréquentielle des signaux et des systèmes linéaires et aux fonctions de filtrage et de synthèse de filtres.

1 Représentation des signaux et systèmes

1.1 Temps

Un signal numérique échantillonné à la fréquence f_e se représente naturellement dans Matlab, comme un vecteur de N éléments (signal de durée $\frac{N}{f_e}$). Le vecteur des temps qui lui est associé est :

```
>> t = (0:N-1)/fe;
```

1.2 Autocorrélation

L'estimation de l'autocorrélation d'un signal ou de l'intercorrélation de deux signaux de longueur N peut être effectuée avec la fonction xcorr :

```
>> Cxy = xcorr(x,y,option);
```

C'est un vecteur de longueur 2N-1 tel que le Nième élément corresponde à la corrélation en 0. si option n'est pas donné, xcorr estime la corrélation non normalisée :

$$C^{\text{un}}_{x,y}[n] = \begin{cases} \sum_{k=1}^{N-n} x^*[k]y[k+n] & \text{si } n \geq 0 \\ C_{x,y}[-n]^* & \text{si } n < 0 \end{cases}$$

option peut prendre les valeurs :

– 'biased' pour l'estimateur biaisé de la corrélation :

$$C^{\text{b}}_{x,y}[n] = \frac{1}{N} C^{\text{un}}_{x,y}[n].$$

– 'unbiased' pour l'estimateur non biaisé de la corrélation :

$$C^{\text{nb}}_{x,y}[n] = \frac{1}{|N-n|} C^{\text{un}}_{x,y}[n].$$

– 'coeff' pour laquelle la corrélation est normalisée de façon à ce que $C^{\text{c}}_{x,y}(0) = 1$.

1.3 Fonction de transfert

– La fonction de transfert (transformée de Laplace de la réponse impulsionnelle) d'un filtre analogique s'écrit sous la forme :

$$F(s) = \frac{b_0 s^M + b_1 s^{M-1} + \cdots + b_{M-1}s + b_M}{s^N + a_1 s^{N-1} + \cdots + a_{N-1}s + a_N}$$

ou sous la forme :

$$F(s) == K\frac{\sum_{k=1}^{M}(s-z_k)}{\sum_{k=1}^{N}(s-p_k)}.$$

– La fonction de transfert (transformée en z de la réponse impulsionnelle) d'un filtre numérique s'écrit quant à elle :

$$F(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} = K\frac{\sum_{k=1}^{M}(z-z_k)}{\sum_{k=1}^{N}(z-p_k)}.$$

Ces systèmes peuvent donc se représenter dans Matlab, avec les vecteurs du dénominateur a=[1, $a_1, \ldots a_N$] et du numérateur b=[b_0, $b_1, \ldots b_M$] ou par le gain K et les vecteurs des poles p=[p_0, $p_1, \ldots p_M$] et des zéros z=[z_0, $z_1, \ldots z_M$].

Un filtre numérique à réponse impulsionnelle finie (RIF) ayant son dénominateur à 1 sera entièrement caractérisé par sa réponse impulsionnelle (h = b).

2 Représentations fréquentielles

2.1 Signaux

La Transformée de Fourier Discrète d'un signal de `N` points est calculée par un algorithme rapide (*Fast Fourier Transform* FFT) :

```
>> X = fft(x) ;
```

C'est également un signal (à valeurs complexes) de `N` points échantillonnés à la fréquence $\frac{N}{f_e}$. Le vecteur des fréquences qui lui est associé est :

```
>> f = (0:N-1)/N*fe;
```

Rappelons que ce signal est de période f_e ; on peut le représenter sur l'intervalle $[-\frac{f_e}{2}, \frac{f_e}{2}]$ grâce à la fonction `fftshift` (qui ne fait qu'un décalage des vecteurs et aucun calcul de fft) :

```
>> Y = fftshift(X);
```

Le vecteur des fréquences qui lui est associé est alors :

```
>> f = (0:N-1)/N*fe - fe/2;
```

2.2 Systèmes

La réponse en fréquence d'un système analogique est donnée par :

```
>> H=freqs(b,a,w);
```

`H` est la réponse en fréquence aux pulsations données dans le vecteur `w` (en radian par seconde).

La réponse en fréquence d'un système numérique est donnée par :

```
>> H=freqz(b,a,f,fe);
```

`H` est la réponse du système aux fréquences données dans le vecteur `f` (en Hertz) et `fe` la fréquence d'échantillonnage.

Voir l'aide en ligne pour plus de détails...

Remarque : Matlab, travaille en pulsation pour les systèmes analogiques et en fréquence (et même en fréquence normalisée) pour les systèmes numériques.

3 Filtrage et synthèse de filtres

3.1 Filtrage

Le filtrage du vecteur `x` par le filtre numérique défini par `a` et `b` est effectué par :

```
>> y = filter(b,a,x);
```

Remarque : Les conditions initiales de l'équation de récurence peuvent être données en entrée de la fonction `filter`. Elles se calculent par la fonction `filtic`.

3.2 Synthèse filtres RIF

Il existe différentes méthodes de synthèse de filtres RIF approchant un filtre idéal :

3.2.1 Troncature de la Réponse Impulsionnelle

La fonction `fir1` synthétise un filtre RIF simple (défini par une seule bande passante ou coupée) par troncature et fenêtrage de la réponse impulsionnelle du filtre numérique idéal :

```
>> h = fir1(n,fn,type,window) ;
```

- `n` est l'ordre du filtre (longueur de la RI moins un).
- Les fréquences `fn` sont normalisées par rapport à la fréquence de Nyquist (`fn`$= f/\frac{f_e}{2}, 0 \leq$ `fn` ≤ 1). `fn` indique la fréquence de coupure pour les passe-bas et passe-haut, et les fréquences de coupures basse et haute pour les passe-bande et coupe-bande.
- La chaîne de caractère `type` précise le type de filtre. `'high'` pour passe-haut, `'stop'` pour coupe-bande, type omis pour les passe-bas et passe-bande.
- Le vecteur `window` de longueur `n+1`, correspond à la fenêtre prise en compte (par défaut fenêtre de Hamming). Les fonctions Matlab, disponibles pour créer des fenêtres sont : `bartlett`, `blackman`, `boxcar` (rectangulaire), `chebwin` (chebychev), `Hamming`, `hanning`, `kaiser`, `triang` (triangulaire).

Voir l'aide en ligne pour plus de détails...

3.2.2 Échantillonnage de la Réponse en Fréquence

La fonction `fir2` synthétise un filtre RIF par échantillonnage de la réponse en fréquence du filtre analogique idéal et fenêtrage de la réponse impulsionnelle du filtre ainsi construit.

```
>> h = fir2(n,fn,m,window) ;
```

- `n` est l'ordre du filtre (longueur de la RI moins un).
- `fn` est le vecteur des fréquences normalisées ($0 \leq$ `fn` ≤ 1) définissant le filtre idéal comme linéaire par morceaux.
- `m` est le vecteur des amplitudes, aux fréquences données par `fn`, de la réponse en fréquence du filtre idéal.

Voir l'aide en ligne pour plus de détails...

3.2.3 Moindres Carrés

La fonction `firls` synthétise un filtre RIF approchant au mieux, au sens des moindres carrés (norme L_2), la réponse en fréquence du filtre analogique idéal.

```
>> h = firls(n,fn,m) ;
```

- `n` est l'ordre du filtre (longueur de la RI moins un).
- `fn` est le vecteur des fréquences normalisées ($0 \leq$ `fn` ≤ 1) définissant le filtre idéal.
  Attention, contrairement à `fir2`, ces fréquences sont prises deux par deux dans `firls`, permettant ainsi de définir des bandes de fréquences ou le filtre idéal n'est pas précisé (bandes de transition).

- `m` est le vecteur des amplitudes de la réponse en fréquence du filtre idéal aux fréquences `fn`.

Voir l'aide en ligne pour plus de détails...

3.2.4 Méthode de Rémez

La fonction `remez` synthétise un filtre RIF approchant au mieux, au sens du minimax (norme L_∞), la réponse en fréquence du filtre idéal.

```
>> h = remez(n,fn,m) ;
```

Les paramètres sont les mêmes que pour `firls`.
La fonction `remezord` permet de plus d'estimer l'ordre nécessaire à la méthode de remez pour construire un filtre de déviation maximale donnée.
Voir l'aide en ligne pour plus de détails...

3.3 Synthèse de filtres RII

Les principales méthodes de synthèse de filtres à réponse impulsionnelle infinie (RII) procèdent par discrétisation d'un filtre analogique.

3.3.1 Synthèse de filtres analogiques passe-bas

Les fonctions suivantes renvoient les pôles (`p`) zéros (`z`) et gain (`k`) des filtres analogiques passe-bas normalisés (pulsation de coupure unité) :

```
% Butterworth
>> [z, p, k] = buttap(n) ;
% Chebychev
% Oscillations inférieures à Rp dB
% en bande passante
>> [z, p, k] = cheb1ap(n, Rp) ;
% Oscillations au deçà de Rs dB
% en bande coupée
>> [z, p, k] = cheb2ap(n, Rs) ;
% Elliptique : oscillations inférieures
% à Rp dB en bande passante et au deçà
% de Rs dB en bande coupée
>> [z, p, k] = ellipap (n, Rp, Rs) ;
```

Pour obtenir une représentation de ces filtres analogiques en terme des numérateurs et dénominateurs de leur fonction de transfert (transformée de Laplace de leur réponse impulsionnelle) :

```
>> [b, a] = zp2tf(z,p,k) ;
```

3.3.2 Transformation des fréquences

Pour transformer les filtres passe-bas en tout type de filtres :

```
% Lowpass -> Lowpass
```
$(p \longrightarrow \frac{p}{\omega_0})$
```
>> [bt, at] = lp2lp(b,a,W0) ;
% Lowpass -> Highpass
```
$(p \longrightarrow \frac{\omega_0}{p})$
```
>> [bt, at] = lp2hp(b,a,W0) ;
% Lowpass -> Bandpass
```
$(p \longrightarrow \frac{1}{B}(\frac{p}{\omega_0} + \frac{\omega_0}{p}))$
```
>> [bt, at] = lp2bp(b,a,W0,Bw) ;
% Lowpass -> bandstop
```
$(p \longrightarrow B\frac{1}{\frac{p}{\omega_0} + \frac{\omega_0}{p}})$
```
>> [bt, at] = lp2bs(b,a,W0,Bw) ;
```

Si ω_b est la pulsation basse de coupure et ω_h la pulsation haute de coupure, alors la pulsation propre du filtre ω_0 et la largeur de bande du filtre sont donnés par : $B = \omega_h - \omega_b$ et $\omega_0 = \sqrt{\omega_h \omega_b}$.

3.3.3 Discrétisation des filtres

La discrétisation des filtres analogiques permet d'obtenir les coefficients des filtres numériques à partir de ceux du filtre analogiques. Deux techniques sont disponibles à cette fin dans Matlab :

- Discrétisation par invariance de la réponse impulsionnelle :
  ```
  >> [bd, ad] = impinvar(b, a, fe) ;
  ```
  Où `fe` est la fréquence d'échantillonnage.
- Discrétisation par transformation bilinéaire : (approximation $p \approx \frac{2}{T_e}\frac{1-z^{-1}}{1+z^{-1}}$)
  ```
  >> [bd, ad] = bilinear(b, a, fe) ;
  ```
  Attention, la transformation bilinéaire provoque une déformation des fréquences (soit f_a la fréquence analogique et f_n la fréquence numérique) :
  $$f_n = \frac{f_e}{\pi} \arctan(\frac{\pi f_a}{f_e})$$
  et
  $$f_a = \frac{f_e}{\pi} \tan(\frac{\pi f_n}{f_e}).$$
  Il est donc nécessaire de pré-déformer le gabarit du filtre analogique pour obtenir le filtre numérique désiré.

3.3.4 Synthèse complète des filtres

Matlab, propose des fonctions dans lesquelles la synthèse complète du filtre numérique est effectuée :

```
% Butterworth
>> [b, a] = butter(n,Wn,type) ;
% Chebychev
% Oscillations de Rp dB en bande
% passante
>> [b, a] = cheby1(n,Rp,Wn,type) ;
% Oscillations au deçà de Rs dB
% en bande coupée
>> [b, a] = cheby2(n,Rs,Wn,type) ;
% Elliptique : oscillations de Rp dB en
% bande passante et au deçà de Rs dB en
% bande coupée
>> [b, a] = ellip(n,Rp,Rs,Wn,type) ;
```

Ces fonctions donnent directement les coefficients `a` et `b` du filtre numérique à partir de :
- `n` ordre du filtre.