Séance 3: programmation de filtres IIR en virgule fixe,

Lors de cette séance, on va tout d'abord tester les performances obtenues par changement d'opérateur sur le filtre

1. analyse des effets de la méthode de changement d'opérateur

1.1. Analyse sous scilab

1.1.1. Cas du codage avec l'opérateur $q\left(z^{-1}\right)=\frac{2^{-L_q}}{1-\left[1-2^{-L_q}.z^{-1}\right]}$

☐Éditer le programme Scilab '*exemple_53.sce*' , l'enregister sous le nom *my_iir_with_op.sce* , modifier la ligne d'exécution conformément à ce nouveau nom,

☐modifier la définition du filtre dans le plan *W*, de façon à ce qu'elle corresponde à celle de votre filtre en exemple. (modifier également le nombre de bits de codage et la fréquence d'échantillonnage si besoin est)

☐Exécuter ce programme et vérifier que les résultats obtenus (coefficients entiers,quantifiés, facteurs d'échelles, décalage L_q) correspondent à ceux du travail préparatoire.

☐Relever la norme 1 de la fonction de transfert $H_{ex}\left(z^{-1}\right)$ entre l'entrée e et la variable interne x du filtre, et vérifier que la valeur du facteur d'échelle de signal $\lambda=2^L$ est correctement choisie (*scaling* en norme 1).

☐ Relever également la valeur maximale de la sortie due aux bruits (bruit de sortie maximum). Expliquer la façon dont elle est calculée dans le programme, ainsi que la raison pour laquelle cette valeur est de l'ordre de 1 (voir travail préparatoire).

☐Relever (sur la figure 0) l'erreur maximale entre la réponse fréquentielle du filtre quantifié, et la réponse fréquentielle du filtre idéal. Comparer cette valeur à l'erreur relative sur les coefficients (exprimée en db). Cette analyse correspond-elle au comportement attendu dans le travail préparatoire.

1.1.2. Cas du codage avec l'opérateur retard $q=z^{-1}$

☐Dans le programme Scilab '*exemple_53.sce*' , et modifier la variable *switch_z_1* pour que l'opérateur utilisé soit à présent : $q=z^{-1}$

☐Exécuter ce programme

☐Relever la norme 1 de la fonction de transfert $H_{ex}\left(z^{-1}\right)$ entre l'entrée e et la variable interne x du filtre, et vérifier que la valeur du facteur d'échelle de signal $\lambda=2^L$ est correctement choisie.

☐ Relever également la valeur maximale de la sortie due aux bruits (bruit de sortie maximum). La valeur trouvée est -elle cohérente avec l'analyse du travail préparatoire.

☐ Relever (sur la figure 0) l'erreur maximale entre la réponse fréquentielle du filtre quantifié, et la réponse fréquentielle du filtre idéal. Comparer cette valeur à l'erreur relative sur les coefficients (exprimée en db). Cette valeur correspond-elle au comportement prévu dans le travail préparatoire.

1.2. implémentation du filtre avec opérateur

1.2.1. réalisation et test sous *netbeans*

Pour intégrer le codage du filtre avec l'opérateur $q(z^{-1})$ au projet netbeans, on applique le même principe que précédemment : on ne détruit pas son travail, on l'enrichit en profitant de ce qui a déjà été réalisé =>

- ☐ créer un source ***my_iir_oq.c***, et le header ***my_iir_oq.h*** *{oq pour opérateur q}*
- ☐ copier le contenu de ***my_iir.c*** dans ***my_iir_oq.c***
- ☐ dans ***my_iir_oq.c,*** remplacer partout la chaîne de caractères *iir* par :*iir_oq*, et sauver le programme *my_iir_oq.c.*

☐ ajouter au header ***my_iir_oq.h*** la déclaration de la fonction externe *teste_iir_oq*

☐ modifier le programme principal de façon à inclure ce header, et à exécuter la fonction *teste_iir_oq()*

☐ modifier le source ***my_iir_oq.c*** pour y intégrer l'opérateur $q\left(z^{-1}\right)=\frac{2^{-L_q}}{1-\left[1-2^{-L_q}.z^{-1}\right]}$, il faudra

1- modifier la structure *my_iir_oq*, pour y intégrer
-la mémoire interne *mq_32* de l'opérateur
- le décalage à droite L_q
2- modifier la routine d'initialisation, pour initialiser ces 2 quantités
3- modifier la routine appelée à chaque pas d'échantillonnage, pour y intégrer la programmation sous forme directe 2 de votre filtre, en arithmétique 16/32bits
4- modifier la définition des coefficients du filtre ' idéal' (pour qu'ils correspondent aux coefficients en z^{-1} du filtre quantifié)

simulation et test

☐exécuter le programme dans les mêmes conditions qu'à la séance 2, et comparer les niveaux de bruits à ceux obtenus séance 2, avec et sans *noise shaping*. Comparer également la valeur efficace de la sortie du filtre idéal (dont les coefficients sont quantifiés) à la valeur efficace que l'on devrait obtenir si l'on n'avait pas quantifié les coefficients)
- commentaires...

2. codage en virgule fixe d'un filtre iir d'ordre élevé, emploi d'un gabarit, génération automatique de code

Dans cette partie on va illustrer les étapes de synthèse d'un filtre numérique iir à l'aide d'un outil cao. Ce filtre devrait ensuite être implémenté sur un processeur spécialisé en traitement de signal (Digital Signal Processor ou DSP), en arithmétique 16/32 bits.

Les différentes étapes de synthèse se décomposent généralement comme suit

1- détermination du gabarit et du modèle de filtre

2- choix d'une structure d'implémentation (cascade, parallèle , passe-tout en parallèle,...)

3- choix d'une forme d'implémentation pour chaque cellule (*df1*,*df2*,...)

4- scaling et analyse des performances avec les schémas standard(bruits , réponse fréquentielle)

5- lorsque les performances sont satisfaisantes, génération du code

Généralement on doit disposer d'un outil CAO pour réaliser ces étapes: il est inenvisageable d'écrire manuellement le code, ou de déterminer manuellement les coefficients d'un filtre d'ordre élevé. Dans cette partie vous aurez juste à adapter le contenu du programme scilab ***iir .sce****, qui se charge ensuite du calcul et de la génération du code en langage c correspondant au filtre iir à générer.*

2.1. gabarit et modèle de filtre

Un gabarit de filtre est la description de l'ensemble des gains admissibles pour la réponse fréquentielles du filtre. Il existe 4 types de gabarits, selon les types de filtres que l'on veut créer

passe-bas ou low-pass, voir Dessin 1
passe-haut ou high-pass, voir Dessin 2
passe-bande ou band-pass, voir Dessin 3
coupe-bande ou band-stop, voir Dessin 4

plus le gabarit est étroit, plus l'ordre du filtre (degré du dénominateur de la fonction de transfert) est élevé. Pour un gabarit donné, on cherche généralement la fonction de transfert de plus faible degré qui respecte les spécifications. Les mathématiciens ont défini des ***modèles de filtre*** *qui permettent de déterminer rapidement les fonctions de transfert de degré minimal répondant au gabarit, pour différentes spécifications typiques*

*les **modèles type** de filtres classiquement employés sont décrits ci-dessous*

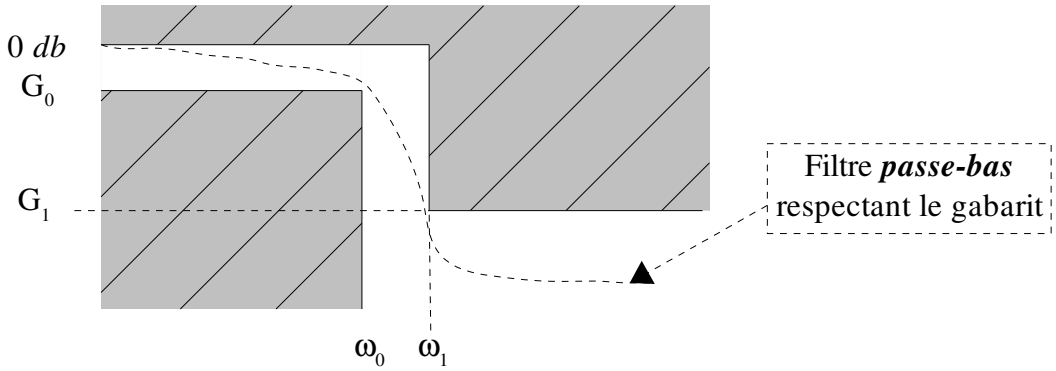
butterworth ou maximally flat : le plus plat dans la bande passante, pas d'ondulations (degré très élevé)

legendre : coupure la plus raide, sans ondulations (degré élevé)

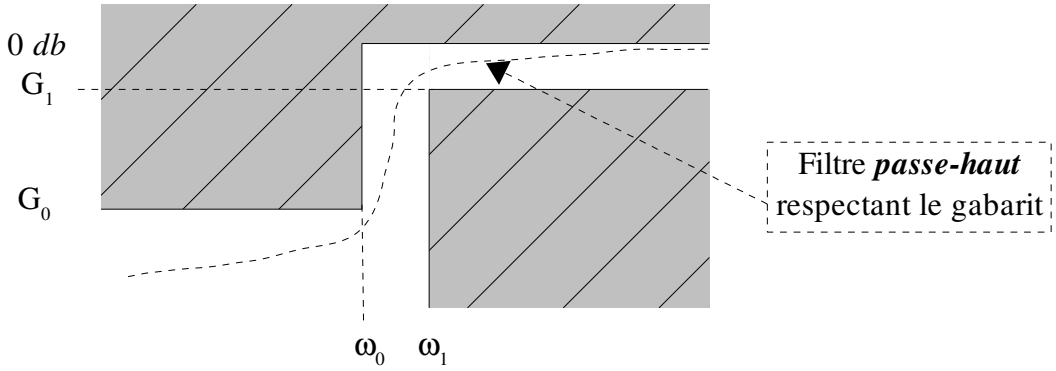
chebycheff 1 : coupure la plus raide, avec ondulations uniquement dans la bande passante (degré assez élevé)

chebycheff 2 : coupure la plus raide, avec ondulations uniquement dans la bande coupée (degré assez élevé)

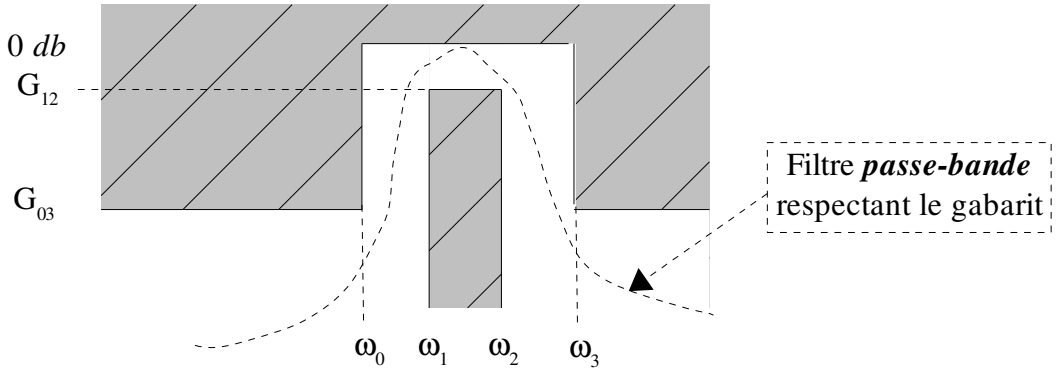
elliptique : coupure la plus raide, avec ondulations simultanées dans les 2 bandes(degré faible)



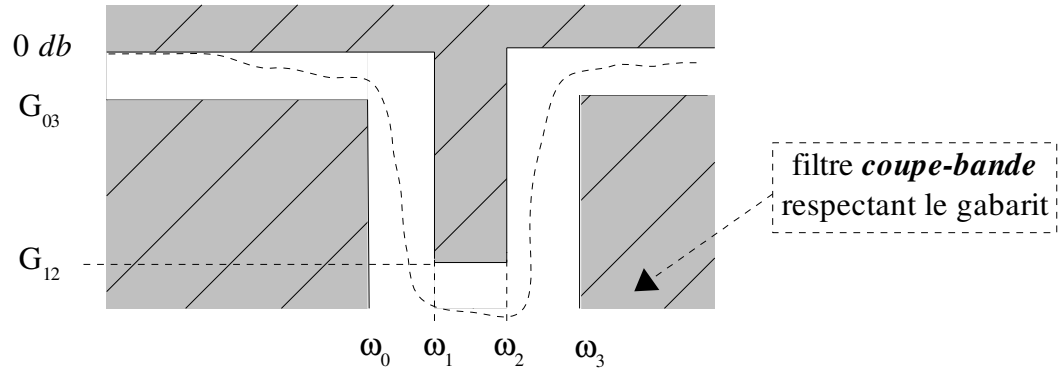
*Dessin 1: gabarit de filtre **passe-bas** caractérisé par 2 gains et 2 pulsations*
en scilab :[K,F_de_p] = low_pass_en_p(modele_flt,w0,w1,G0_db,G1_db)



*Dessin 2: gabarit de filtre **passe-haut** caractérisé par 2 gains et 2 pulsations*
en scilab :[K,F_de_p] = high_pass_en_p(modele_flt,w0,w1,G0_db,G1_db)



*Dessin 3: gabarit de filtre **passe-bande** caractérisé par 2 gains et 4 pulsations*
en scilab :[K,F_de_p] = band_pass_en_p(modele_flt,w0,w1,w2,w3,G03_db,G12_db)



*Dessin 4: gabarit de filtre **coupe-bande** caractérisé par 2 gains et 4 pulsations*
en scilab :[K,F_de_p] = band_stop_en_p(modele_flt,w0,w1,w2,w3,G03_db,G12_db)

☐ éditer le programme scilab ***iir.sce,*** l'enregistrer sous le nom ***iir_pb.sce***, modifier la première ligne, et exécuter le programme pour vérifier qu'il ne produit pas d'erreur.

*On veut générer un filtre **iir** ayant les mêmes caractéristiques que le filtre FIR de la séance 1, soit encore :*

un gain supérieur à 0.8 pour des fréquences inférieures à 90 hz

un gain inférieur à 0.2 pour des fréquences supérieures à 110 hz

de plus le filtre devra travailler dans la bande de fréquence 0 à 4 KHZ, et on travaillera en arithmétique 16/32 bits

☐dans le programme ***iir_pb.sce***, modifier la définition du gabarit du filtre, ainsi que la fréquence d'échantillonnage, conformément à ce cahier des charges. Exécuter le programme, puis observer les réponses fréquentielles obtenues, ainsi que l'ordre du filtre. Recommencer pour les différents modèles de filtre possibles, en modifiant la variable ***modele_flt*** . Vérifier que les caractéristiques des filtres correspondent bien à ce qui était annoncé en 2.1.

Pour la suite on retiendra un filtre de Chebychev de type 2 (pas d'ondulations dans la bande passante), car on juge que le butterworth (le plus platdans la bande) a un ordre trop élevé.

2.2. Choix d'une structure d'implémentation

Les cellules du filtre seront systématiquement programmées sous forme directe 2 (c'est la seule forme pour laquelle le programme génère automatiquement le code en langage c). Il reste à choisir si le filtre sera implémenté en cascade (factorisation) ou en parallèle (décomposition en éléments simples)

☐dans le programme, choisir la norme 1 pour le scaling, et la norme 2 pour l'analyse des niveaux de bruits. Modifier la variable ***params.switch_structure*** de telle façon que le filtre soit implémenté en cascade. Exécuter le programme et relever le niveau de bruit obtenu. Modifier à présent l'opérateur employé pour le codage en choisissant ***params.switch_operateur="x"***, et relever à nouveau le niveau de bruit obtenu.

☐Recommencer pour une décomposition parallèle (choisir ***params.switch_structure="cascade-to_paralell"*** , puisque ***F_de_w*** représente une cascade qui doit être implémentée en paralèlle).

On retiendra finalement pour ***params.switch_structure***, et ***params.switch_operateur*** les valeurs donnant les plus faibles niveaux de bruit.

2.3. Génération automatique de code, et intégration sous netbeans

☐ Sachant que l'on a employé la norme 1 pour le scaling, doit-on saturer les variables internes du filtre? En déduire la valeur de la variable ***switch_saturate*** dans le programme, puis l'exécuter.

☐Editer le fichier source ***toto.c*** généré automatiquement par le programme scilab. Lancer netbeans et ajouter à votre projet un fichier source ***iir_pb.c*** et le header correspondant ***iir_pb.h***.

☐ Copier le contenu de ***toto.c*** dans le fichier ***iir_pb.c***. Construire le projet et vérifier qu'il n'y a pas d'erreur.

☐ Sous netbeans, copier (dupliquer) la fonction *teste_iir_ns()* (dans le fichier *my_iir_ns.c)* à la fin du fichier ***iir_pb.c,*** puis la renommer ***teste_iir_pb()***.

☐Modifier la fonction ***teste_iir_pb()*** de façon à ce qu'elle permette de comparer le comportement du filtre *iir* en nombre réels, à celui du filtre *iir* en nombres entiers. (vous pouvez vous inspirer des deux fonctions générées automatiquement : ***teste_real_Fz***(), et ***teste_16bits_filter_Fz***() , pour modifier le code de **teste_iir_pb**()).

☐ Ajouter la déclaration de ***teste_iir_pb*** dans le header, modifier le programme principal pour qu'il exécute ***teste_iir_pb()***. Puis tester le comportement du filtre pour des entrées sinusoïdales d'amplitude maximale, et de fréquence: 1 hertz, 90 hertz, 110 hertz. Comparer les caractéristiques de l'erreur à celles prévues....

☐ Soyons prudents : Une fois que la fonction ***teste_iir_pb*** est au point : la sauver sous votre éditeur de texte (pas sous netbeans) dans un fichier nommé ***code_de_teste_iir.c.****(sinon chaque fois que vous générerez le code, il faudra réécrire cette fonciion...)*

2.4. Adaptation du code, pour implémentation sur carte DSP

Le CAN de la carte utilisée (pour le moment), est un CAN 8 bits non signé, pour des tensions mesurées entre 0 et 5 volts. Le CNA est un CNA 8 bits non signé, pour des tensions de sortie entre 0 et 2,5 volts. Le filtre aura pour objectif de filtrer des signaux entre 0 et 2.5volts, avec un gain statique en tension égal à 1. Le filtre numérique devra donc avoir un gain statique égal à 2. Et on devra modifier le code de ***one_step_16bits_filter_Fz()*** *conformément à ces spécifications*

☐Modifier le programme scilab ***iir_pb.sce*** pour qu'il calcule un filtre de gains statique = 2 (multiplier le gain *K* par 2, juste avant la ligne :*F_de_w=distribute_gain(K,F_de_w);*). Exécuter le programme et copier le contenu de *toto.c* dans le programme *iir_pb.c* (le nouveau contenu doit remplacer l'ancien, sauf la fonction que vous avez écrite : ***teste_iir_pb()*** *)*

2.5. rétro-ingénierie

☐ Editer le code de la fonction ***one_step_real_filter()***, et en déduire le schéma de principe du filtre.

☐Editer le code de la fonction ***one_step_16bits_filter_Fz()***, et en déduire le schéma d'implémentation détaillé du filtre. (demander l'aide du prof. Ce travail demande d'être rigoureux et méthodique).

☐En déduire la valeur des facteurs d'échelle utilisés, ainsi que les opérateurs employés pour chaque cellule.

2.6. Adaptation du code

☐Modifier manuellement la fonction ***one_step_16bits_filter_Fz()*** pour l'adapter à la carte employée => vous devrez

1-multiplier l'entrée par 2^7 (pour l'adapter à la pleine échelle sur 16 bits), en modifiant un décalage

2-diviser la sortie par 2^7 (pour l'adapter à la pleine échelle du CNA sur 8 bits), en modifiant un décalage

3-saturer la sortie entre 0 et 255 (entre 0 et 2,5 volts)

☐exécuter le code pour vérifier que le programme fonctionne correctement, pour des entrées analogiques entre 0 et 2,5 volts (<=> valeurs numériques de l'entrée entre 0 et 127)