TELECOM SudParis
Cours de traitement du signal SIC3501.

TP 2

Filtrage

Ce TP aborde le problème du filtrage numérique. On souhaite filtrer un signal noté x_n par un filtre passe-bas de fréquence de coupure normalisée f_0. Le résultat du filtrage est noté y_n.

On utilisera le langage Python et on chargera le module PyLab à l'aide de :

```
from pylab import * # importe le module pylab
ion() # mode interactif pour les tracés
```

1 Génération d'un signal

Dans tout le TP, on générera un signal aléatoirement.

1– *On note $(x_n)_{n=0...T-1}$ des échantillons d'un bruit blanc numérique de puissance unité. Le nombre d'échantillons T est à choisir judicieusement. Générer un vecteur* `x` *contenant ces échantillons.*

```
T = 1000 #nombre d'échantillons (on constatera ensuite que T pair est pratique)
x = randn(T)
plot(x)
```

2– *Prendre la transformée de Fourier discrète de* `x`*, que l'on appellera* `X`*, puis afficher le spectre du signal en fonction de la fréquence normalisée.*

```
X = fft(x, T)
plot(arange(T)/T, abs(X))
```

2 Filtrage dans le domaine des fréquences

Dans un premier temps, nous souhaitons utiliser la transformée de Fourier discrète pour effectuer le filtrage. Le filtre appliqué est un filtre passe-bas idéal de fréquence de coupure f_0 (réponse en fréquence égale à un dans la bande et zéro en dehors).

3– *A partir de* `X`*, calculer* `Y`*, transformée de Fourier discrète du signal filtré y_n.*

```
f0 = 0.05 # a choisir...
Nbf = floor(T*f0)
Masque = hstack((1, ones(Nbf), zeros(T-(2*Nbf+1)), ones(Nbf)))
Y = X*Masque
```

4– *Retrouver et afficher le signal y_n.*

```
y = ifft(Y)
plot(y)
```

5– *Faire varier f_0, observer et commenter.*

3 Filtre RIF

On souhaite maintenant effectuer le filtrage passe-bas par un filtre RIF. Le calcul d'un filtre RIF qui approxime le filtre passe-bas idéal a été fait en TD (exercice 12). On notera L la longueur de la réponse impulsionnelle et on supposera le filtre causal.

3.1 Filtrage du signal

6– *Prendre dans un premier temps $L = 5$ et calculer la réponse impulsionnelle du filtre que l'on stockera dans un vecteur noté* `h`.

```
L = 5 # longueur filtre RIF (prendre L impair)
tmp = arange(-floor(L/2), ceil(L/2), 1)
h = 2*f0*sinc(2*tmp*f0)
```

7– *Calculer et tracer la filtrée y_n du signal x_n par le filtre dont la réponse impulsionnelle est dans* `h`. *Pour ce faire, on pourra utiliser et comparer les méthodes suivantes :*

- *Utiliser la fonction `lfilter` du module `scipy.signal`.*
  ```
  from scipy.signal import lfilter
  y = lfilter(h, 1, x)
  ```
- *Utiliser la fonction `convolve`.*
  ```
  y = convolve(h, x, mode='full')
  y = y[:-L+1:]
  ```
- *Mettre l'équation de convolution sous forme d'un produit matriciel (fonction utile : `toeplitz` dans le module `scipy.linalg`).*
  ```
  from scipy.linalg import toeplitz
  y = h.dot(toeplitz(hstack((x[0], zeros(L-1))), x))
  ### autre solution:
  ### y = x.dot(toeplitz(hstack((h[0], zeros(T-1))), hstack((h, zeros(T-L)))))
  ```

8– *Tracer le spectre du signal y_n.*

```
plot(arange(T)/T, abs(fft(y, T)))
```

9– *Faire varier f_0 et L, observer et commenter.*

3.2 Filtrage dans le domaine des fréquences

On souhaite ici vérifier que, à des erreurs de calcul machine près, l'opération de convolution du filtre correspond bien à une multiplication dans le domaine des fréquences (multiplication de la réponse en fréquence et de la transformées de Fourier). On utilisera pour cela la technique du bourrage de zéros afin d'avoir un nombre suffisant de points dans le domaine des fréquences.

10– *Prendre la transformée de Fourier discrète du signal x_n.*

```
Tpad = 2**13 # nombre de points pour bourrage de zéros
X = fft(x,Tpad)
```

11– *Calculer et tracer la réponse en fréquence du filtre dont la réponse impulsionnelle est dans* `h`. *Cette réponse en fréquence est à calculer en les mêmes fréquences que celles de la transformée de Fourier discrète précédente :*

```
H = fft(h, Tpad)
plot(arange(Tpad)/Tpad, abs(H), color='red', linestyle='-.')
```

12– *Calculer la transformée de Fourier discrète du signal filtré et comparer avec la sortie du filtre lorsqu'elle est calculée par l'équation de convolution.*

```
Y_calculFreq = H*X
ytmp = ifft(Y_calculFreq)
y_calculFreq = ytmp[:T:]
print('La norme de (y-y_calculFreq) vaut {}'.format(norm(y-y_calculFreq)))
# le résultat doit être proche de zéro
```

4 Filtre RII

Il existe de nombreuses fonctions intégrées pour la synthèse de filtres. A l'aide de l'une de ces fonctions, un filtre passe-bas, de fréquence de coupure $f_0 = 0.1$ a été synthétisé et sa fonction de transfert en z est donnée par :

$$\frac{0.0539 + 0.0074s^{-1} + 0.0539z^{-2}}{1 - 1.5259z^{-1} + 0.6479z^{-2}}$$

13– *Représenter les pôles et les zéros de ce filtre. On utilisera pour cela la fonction `zplane` du module `plot_zplane` fournis (sur ma page web ou par l'enseignant). Que peut-on dire de la stabilité ?*

```
from plot_zplane import zplane
B = [0.0539, 0.0074, 0.0539]
A = [1, -1.5259, 0.6479]
zplane(B,A)
```

14– *Tracer la réponse en fréquence du filtre (on pourra utiliser la fonction `freqz` du module `scipy.signal`).*

```
from scipy.signal import freqz
w, Hsynth = freqz(B, A, worN=Tpad, whole=True)
plot(arange(Tpad)/Tpad, abs(Hsynth))
```

15– *Filtrer et tracer le signal x_n précédent par le filtre ci-dessus.*

```
y4 = sig.lfilter(B,A,x)
plot(y4)
```

16– *Comparer le filtre ci-dessus avec le filtre RIF précédent de même fréquence de coupure.*

17– *Au vu de l'exemple ci-dessus, que peut-on dire de façon générale à propos de la comparaison des filtres RII et RIF ?*