

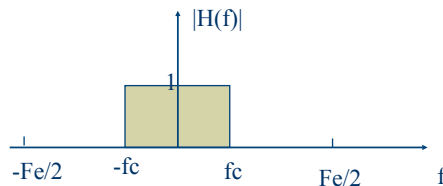
Conception de filtres LTI

- ♦ Un filtre LTI avec une réponse en fréquence : $H(f)$
agit comme une fonction de pondération des différentes composantes fréquentielles du spectre du signal d'entrée $X(\omega)$, pour conduire à un signal de sortie :
$$Y(f) = H(f) \cdot X(f) \quad \longleftrightarrow \quad y(t) = h(t) * x(t)$$

Les filtres LTI sont classés suivant l'amplitude de leur réponse $|H(f)|$, comme passe-bas, passe-haut, passe-bande ou coupe-bande.

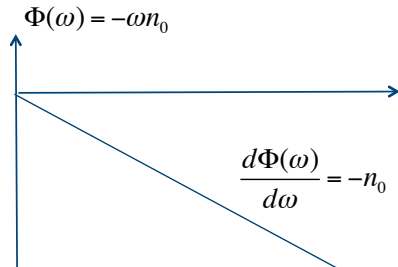
Filtre LTI passe-bas idéal

- ♦ Gain unité dans la bande passante $[0 F_c]$, et 0 dans la bande coupée $[F_c F_s/2]$
- ♦ Phase linéaire dans la bande passante, i.e.



Filtre LTI passe-bas idéal

- En général le délai temporel subi par une composante du signal à la fréquence ω lorsqu'elle passe dans une filtre est donnée par :

$$Tg = -\frac{d\Phi(\omega)}{d\omega}$$

$$\Phi(\omega) = -\omega n_0$$
$$\frac{d\Phi(\omega)}{d\omega} = -n_0$$

Conception de filtres LTI

- On peut définir les filtres LTI par leur sortie en fonction de l'entrée, qui donne une équation aux différences. Ils sont alors déterminés par les coefficients a_k et b_k :

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} - \sum_{k=1}^{M-1} a_k y_{n-k} \quad (1)$$

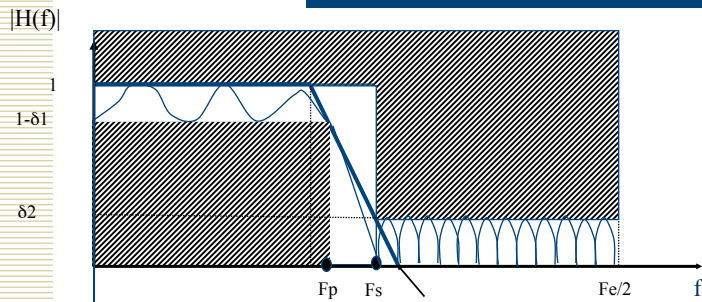
Conception de filtres LTI

- ♦ En choisissant correctement les coefficients a_k et b_k ainsi que les ordres P et Q , il est possible de réaliser **des filtres causaux** qui approximent les filtres idéaux, c.à.d. ayant une réponse en amplitude et une réponse en phase désirées.
- ♦ Les contraintes suivantes doivent être respectées :
 - **Stabilité** : Les pôles doivent être placés dans le cercle unité
 - Les zéros et les pôles complexes doivent apparaître par paires complexes-conjugués de telle façon que les **coefficients a_k et b_k soient réels**.

Synthèse de filtres numériques

- ♦ **Respect de la causalité** : implique de gérer un certain nombre de contraintes :
 - l'amplitude $|H(\omega)|$ ne peut être constante sur une bande de fréquences finie, et la transition de la bande-passante à la bande-coupée ne peut pas être infiniment raide.
 - L'amplitude et la réponse en phase du filtre sont interdépendantes et ne peuvent donc pas être spécifiées indépendamment.

Filtre LTI passe-bas réalisable



δ_1 : ondulation dans la BP
 δ_2 : ondulation dans la BC
fréquence limite de la BP : F_p
fréquence limite de la BC : F_s

Conception de filtres LTI

- ♦ Il existe différentes méthodes de synthèse de filtres LTI qui permettent d'approcher de manière optimale ces spécifications ; il s'agit alors de déterminer les paramètres a_k et b_k qui permettent de les créer (concevoir).
 - Filtres FIR : filtres à réponse impulsionnelle finie
 - réponse en phase linéaire dans la bande passante (BP)
 - Filtres IIR : filtres à réponse impulsionnelle infinie
 - réponse en phase non linéaire dans la BP
 - si la distorsion de phase est tolérable, les filtres IIR sont préférables car leur implémentation demande moins de paramètres, moins de mémoire et a une complexité algorithmique plus petite.

Conception de filtres FIR

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}$$

- ♦ Il existe plusieurs méthodes de synthèse de filtres FIR

Synthèse de filtres FIR en Python bibliothèque signal

```
b1=signal.firwin(ordre,cutoff=[0.2],window='hann',nyq=0.5)
```

ordre : ordre du filtre (nombre de paramètres b_k , soit N)

cutoff : fréquence de coupure normalisée

btype : fonction du filtre

par défaut passe-bas *lowpass* (1 fréquence de coupure)

ou passe-bande (2 fréquences de coupure)

highpass : passe-haut

stopband : pour coupe-bande

window : fenêtre prise en compte par défaut *Hamming* (ici *Hanning*)

nyq : fréquence de Nyquist normalisée ($f_s/2$: ici 0.5 si on regarde la réponse de 0 à $f_s/2$, et 1.0 si on regarde la réponse de 0 à f_s)

Conception de filtres IIR

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} - \sum_{k=1}^{M-1} a_k y_{n-k} \quad (1)$$

- ♦ Il existe plusieurs méthodes de synthèse de filtres IIR

Synthèse de filtres FIR en Python bibliothèque signal

```
b,a = signal.iirfilter(N=2,Wn=[0.4],btype="highpass",ftype="butter", nyq = 0.5)
```

N: ordre du filtre (nombre de paramètres a_k et b_k) : ici 2

Wn: fréquence de coupure normalisée : ici 0.4

btype : fonction du filtre : ici *highpass*

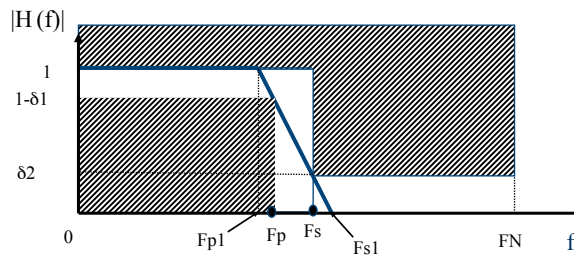
ftype : type du filtre : ici *butterworth*

nyq : fréquence de Nyquist normalisée

Synthèse de filtres IIR

Synthèse de filtres analogiques

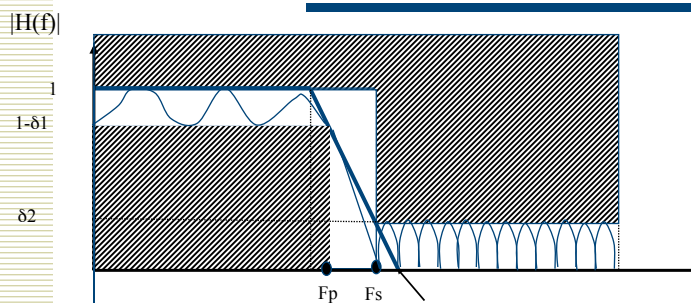
- ♦ Filtre analogique donné par son gabarit (module de la réponse en fréquence)



- ♦ On donne le gabarit :
 - $(0, F_{p1}, F_{s1}, F_N)$ pour les bandes de fréquence en échelle normalisée de préférence (de 0 à 1 ou de 0 à 0.5) ;
 - $(1, 1, 0, 0)$ pour les valeurs de l'atténuation aux points des bandes de fréquence

Synthèse de filtres IIR

Synthèse de filtres analogiques



- ♦ Parfois on donne la valeur d'ondulation dans la bande passante : δ_1 (rp)

Synthèse de filtres en Python avec gabarits bibliothèque signal

- ♦ Autres filtres FIR

firls, minimum phase, remez

- ♦ Exemple

`scipy.signal.firls(numtaps, bands, desired, weight=None, nyq=None, fs=None)`

Synthèse de filtres en Python avec gabarits bibliothèque signal

- ♦ Autres filtres IIR

- butter Filter design using order and critical points
- cheby1, cheby2, ellip, bessel
- buttord Find order and critical points from passband and stopband spec
- cheb1ord, cheb2ord, ellipord
- iirdesign General filter design using passband and stopband spec

- ♦ Exemple

`scipy.signal.butter(N, Wn, btype='low', analog=False, output='ba', fs=None)`

- Butterworth digital and analog filter design.
- Design an Nth-order digital or analog Butterworth filter and return the filter coefficients.

Démarche

- ♦ 1. Conception de filtre (synthèse):
Calcul des vecteurs b, a
- ♦ 2. Tracé de la réponse en fréquence
 $|H|, \Phi$ avec la fonction `freqz`
- ♦ 3. Application du filtre
Sortie y en fonction de x : $y = \text{lfilter}(b, a, x)$