Introduction à la Synthèse sonore

Cours 2 Signaux : filtrage et modulation

Matthias Puech
matthias.puech@lecnam.net

STMN — Cnam ENJMIN, Angoulême

21-22 novembre 2017

Synthèse sonore

Un signal numérique est une approximation

Un signal numérique est un flux *discret* de valeurs (*échantillons*/*samples*) *discrètes* :

flux discret entre deux samples contigus dans le temps, il n'y a *pas* d'information

valeurs discrètes chaque sample ne peut prendre qu'un nombre fini de valeurs

Un signal numérique est une approximation

Un signal numérique est un flux *discret* de valeurs (*échantillons*/*samples*) *discrètes* :

flux discret entre deux samples contigus dans le temps, il n'y a *pas* d'information

valeurs discrètes chaque sample ne peut prendre qu'un nombre fini de valeurs

Codage PCM (Pulse Code Modulation)

- flux d'échantillons à intervalle régulier (ex : 44.1kHz/96kHz)
- chacun représente une mesure du signal à un instant
- et est codé dans le même espace (ex : 8/16/32 bits)

⇝ représentation dans le domaine temporel

Un signal numérique est une approximation

Conversion analogique→numérique

Deux étape :

- échantillonnage
  (mesure à intervalle régulier)
- quantification
  (déplacement à la valeur discrète la plus proche)

Échantillonnage

Mesure du signal analogique à intervalle régulier

Fréquence d'échantillonnage f_S

Exprimé en Hertz :

8kHz	téléphone
32kHz	radio
44.1kHz	CD
48kHz, 96kHz, 192kHz	*"Hi-Res"* : zéro différence audible avec CD (mais utile en synthèse)

Exemple

`https://xiph.org/video/vid1.shtml` (à partir de 9:30)

Échantillonnage

Mesure du signal analogique à intervalle régulier

Fréquence d'échantillonnage f_S

Exprimé en Hertz :

8kHz	téléphone
32kHz	radio
44.1kHz	CD
48kHz, 96kHz, 192kHz	*"Hi-Res"* : zéro différence audible avec CD (mais utile en synthèse)

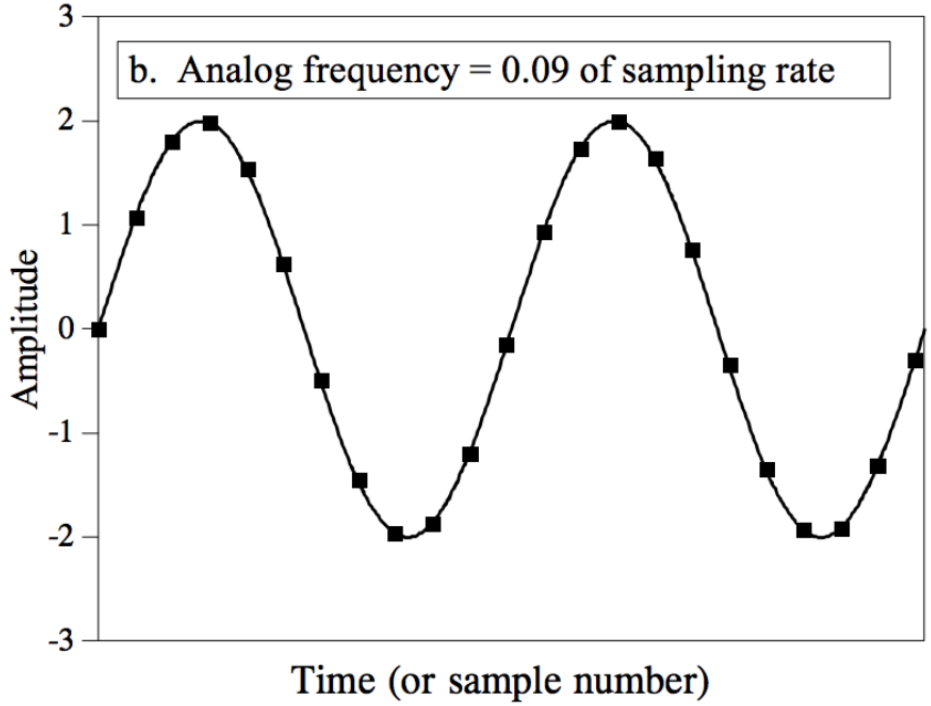
Exemple

`https://xiph.org/video/vid1.shtml` (à partir de 9:30)

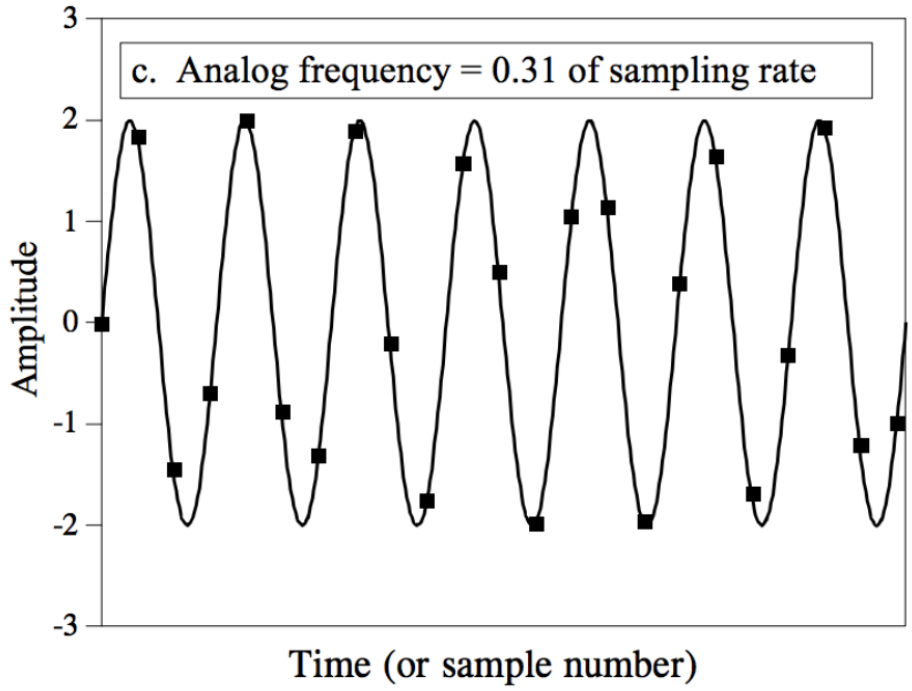
Dans Pure Data

Médias→ Paramètres audio (par défaut, 44.1kHz)

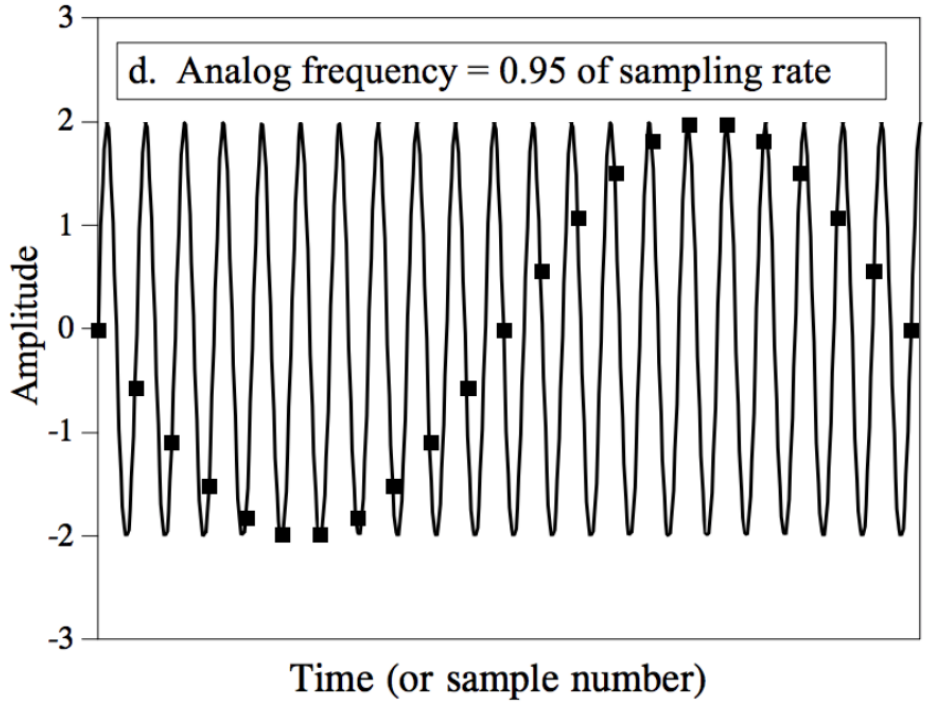
Le théorème de Shannon-Nyquist



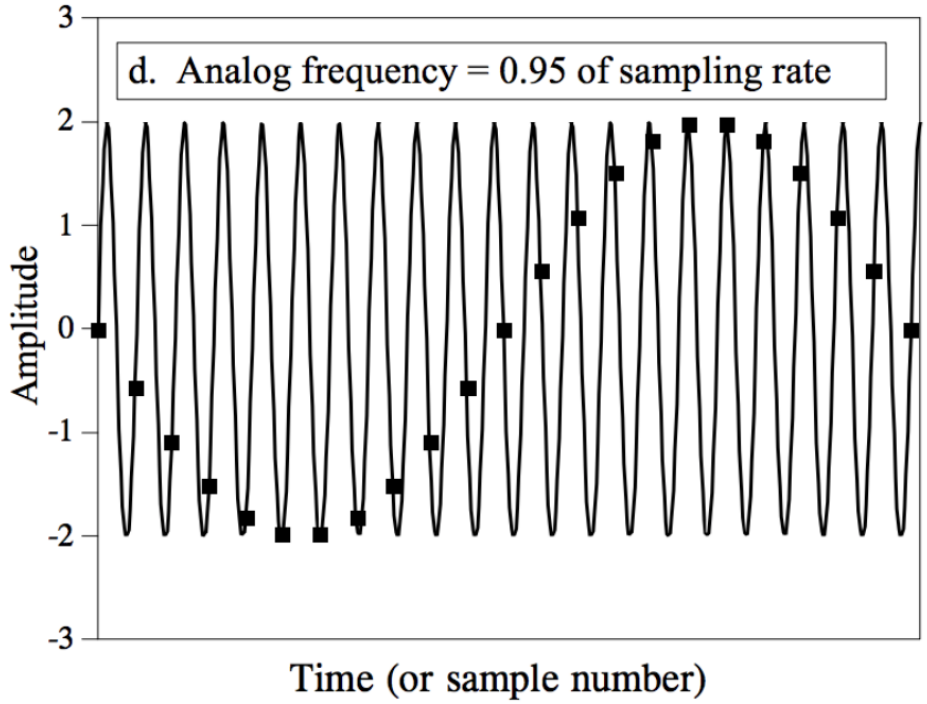
Le théorème de Shannon-Nyquist



Le théorème de Shannon-Nyquist



Le théorème de Shannon-Nyquist



Théorème

La représentation discrète d'un signal requiert une fréquence d'échantillonnage supérieure au double de la fréquence maximale présente dans ce signal.

Cette limite est appelée *fréquence de Nyquist* $f_N = \frac{f_S}{2}$

Aliasing [08.aliasing.pd]

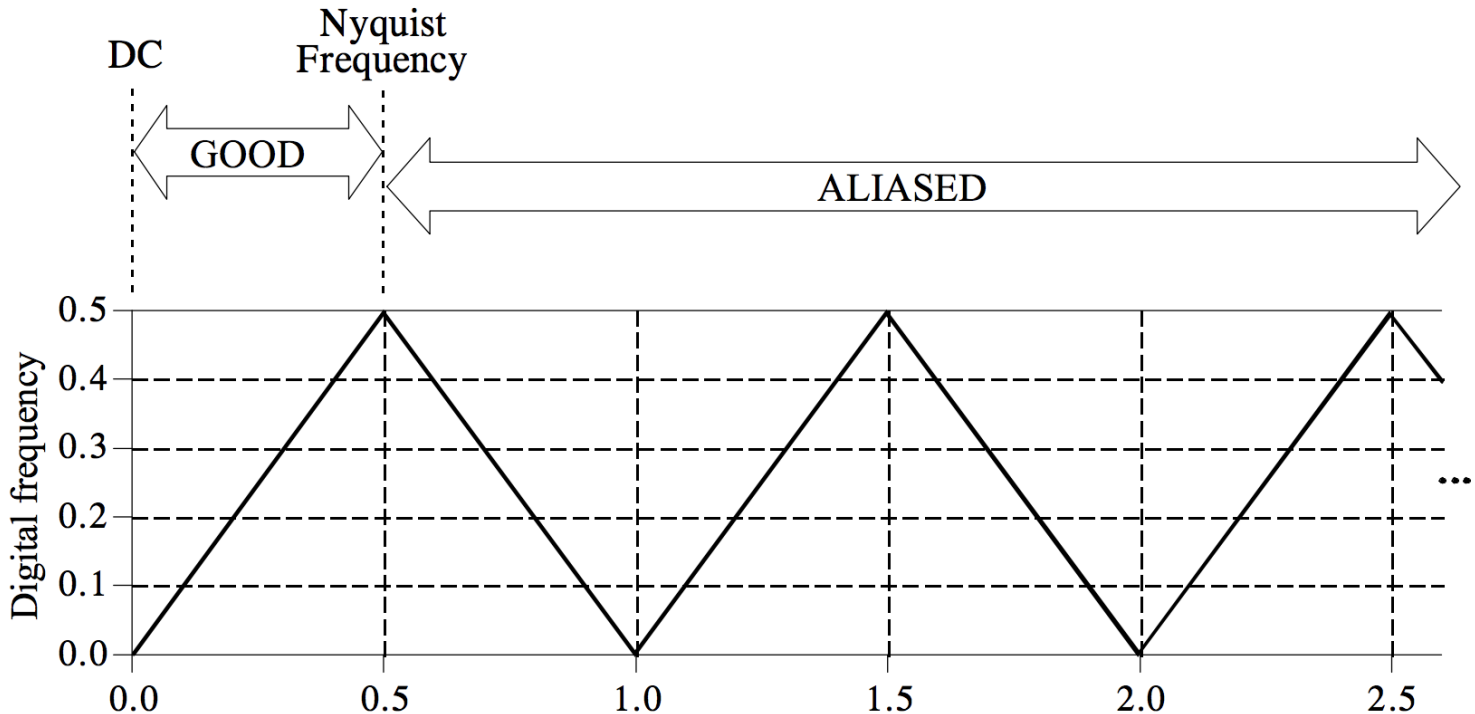
Que se passe-t-il si on échantillonne un signal qui contient des fréquences supérieures à f_N ?

Aliasing [08.aliasing.pd]

Que se passe-t-il si on échantillonne un signal
qui contient des fréquences supérieures à f_N ?

Réponse

Les fréquences au-delà sont “repliées” en dessous de Nyquist



(et s’entendent comme des artefacts désagréables)

Aliasing [08.aliasing.pd]

Que faire pour le prévenir en synthèse ?

Solutions

- ne jamais générer de sinusoïde $> f_N$
  (e.g. en synthèse additive)
- augmenter f_S ou faire du sur-échantillonnage
  ($\rightsquigarrow$ plus de calcul pour le processeur)
- utiliser des algorithmes de synthèse *band-limités*
  (DPW, minBlep, polyBlep...)

Quantification

Choix de la valeur discrète la plus proche de la valeur “réelle”.

Résolution `[07.quantize.pd]`

Exprimé en bits

- 8 bits : 256 valeurs réparties linéairement entre *min* et *max* (ajoute beaucoup d’artefacts)
- 16 bits : 65535 valeurs . . .
- 32 bits flottant : 2 milliards de valeurs réparties exponentiellement entre $-\infty$ et $+\infty$

Quantification

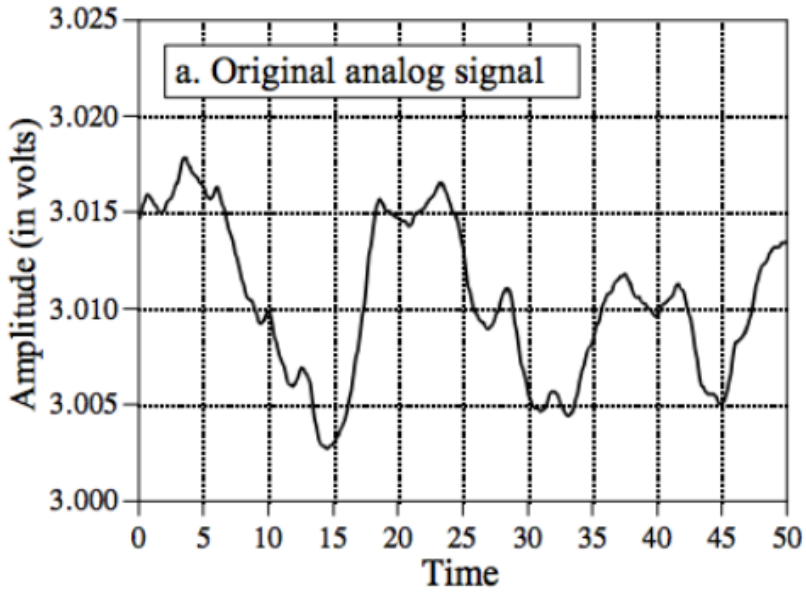
Choix de la valeur discrète la plus proche de la valeur “réelle”.

Résolution [07.quantize.pd]

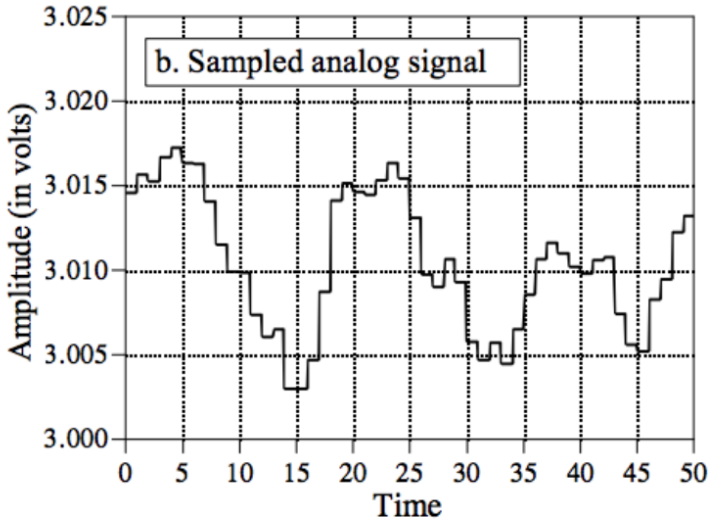
Exprimé en bits

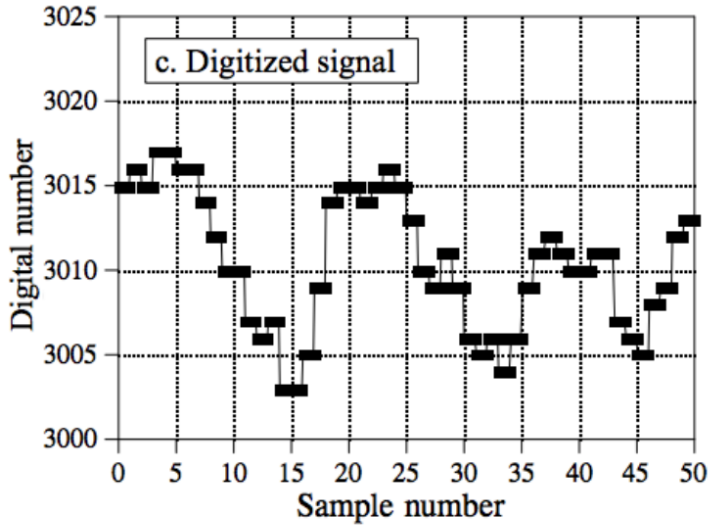
- **8 bits** 256 valeurs réparties linéairement entre *min* et *max* (ajoute beaucoup d’artefacts)
- **16 bits** 65535 valeurs . . .
- **32 bits flottant** 2 milliards de valeurs réparties exponentiellement entre $-\infty$ et $+\infty$

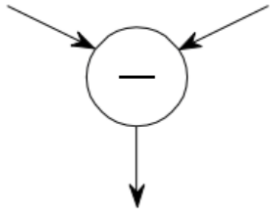
Exemple

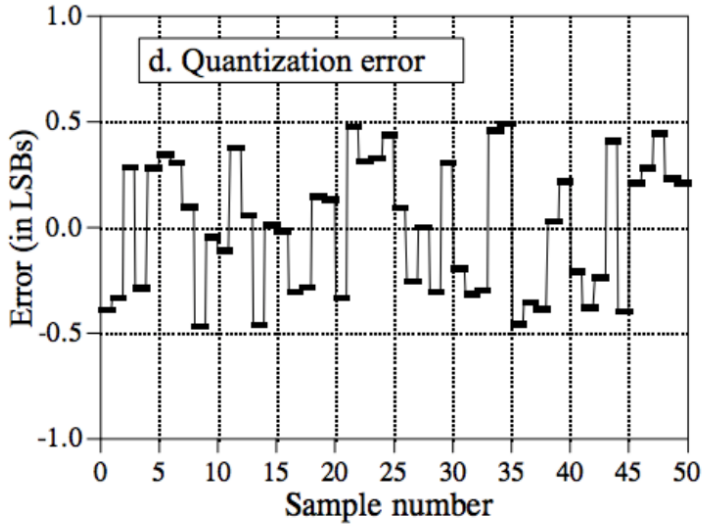


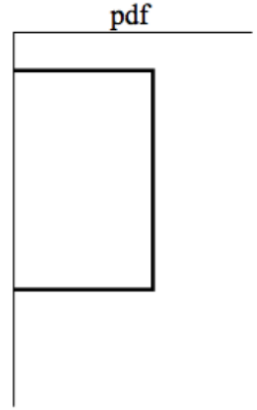
Quantification



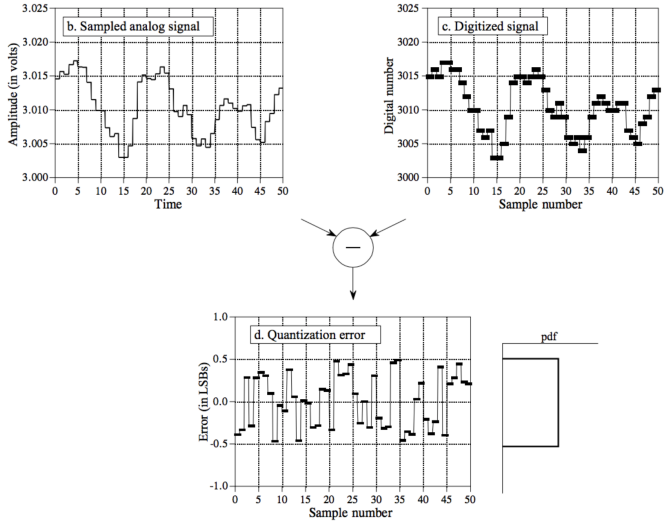








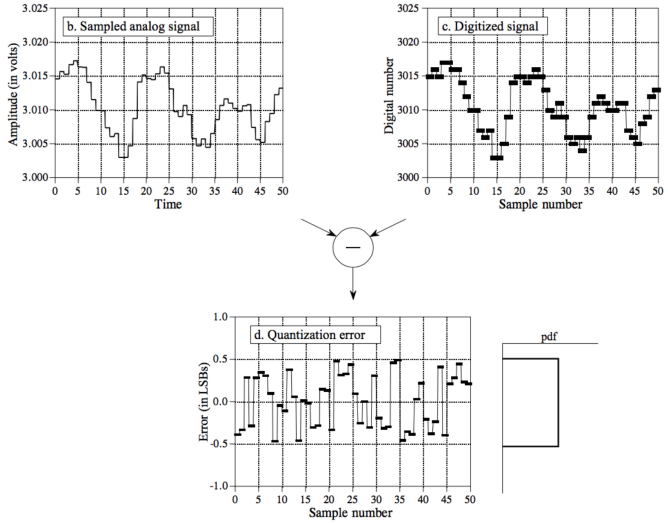
Quantification



Conclusion

La quantification ne fait que rajouter du **bruit** au signal

Quantification



Conclusion

La quantification ne fait que rajouter du **bruit** au signal

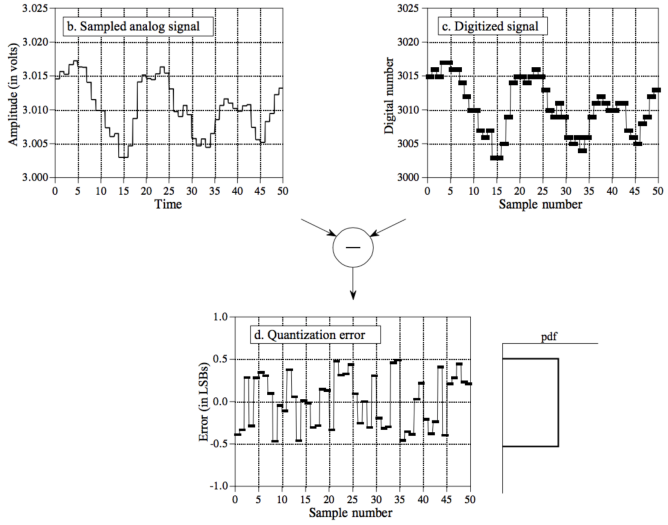
Anecdote

On peut mesurer le rapport signal/bruit d'un medium en bits

cassette audio 9 bits

disque vynile 10 bits

Quantification



Conclusion

La quantification ne fait que rajouter du **bruit** au signal

Anecdote

On peut mesurer le rapport signal/bruit d'un medium en bits

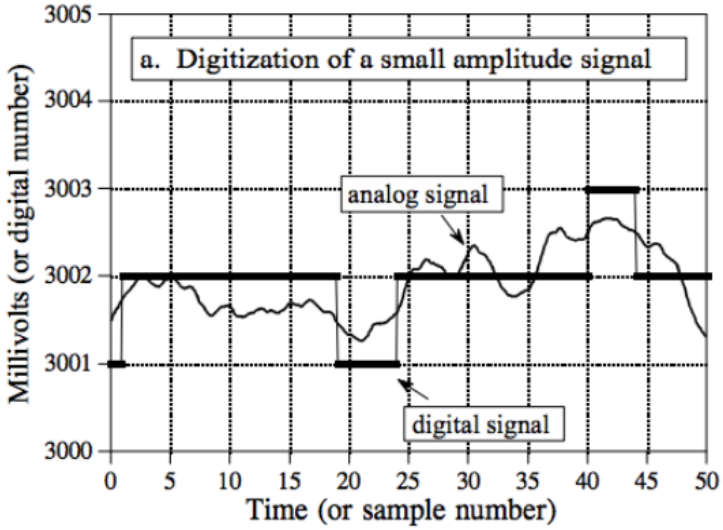
cassette audio 9 bits

disque vynile 10 bits

Pour aller plus loin : `https://xiph.org/video/vid2.shtml`

Dithering

En ajoutant du bruit de façon *maîtrisé* avant quantification, on peut améliorer sa fidélité



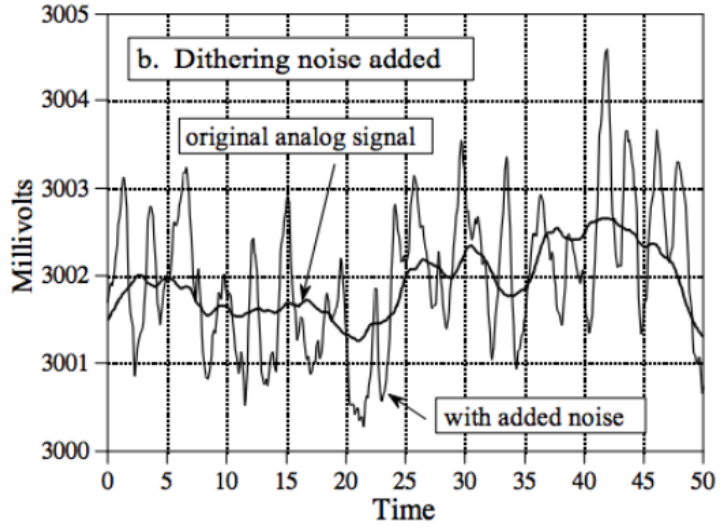
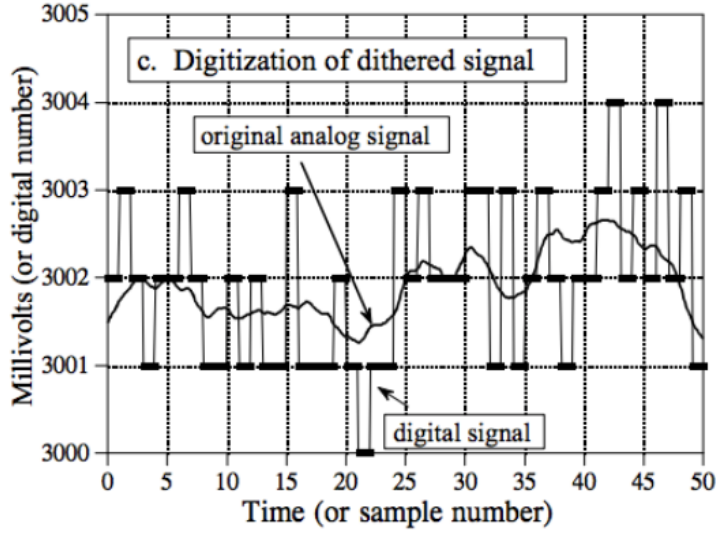


FIGURE 3-2
Illustration of dithering. Figure (a) shows how an analog signal that varies less than ±½ LSB can become *stuck* on the same quantization level during digitization. Dithering improves this situation by adding a small amount of random noise to the analog signal, such as shown in (b). In this example, the added noise is normally distributed with a standard deviation of 2/3 LSB. As shown in (c), the added noise causes the digitized signal to toggle between adjacent quantization levels, providing more information about the original signal.



Traitement par bloc

Processus possible de calcul de Pure Data (ultra-simplifié) :

```
toute les 1/44100s {
  pour chaque objet {
    lire ses entrées;
    calculer le sample suivant;
    écrire dans ses sorties;
  }
}
```

Traitement par bloc

Processus possible de calcul de Pure Data (ultra-simplifié) :

```
toute les 1/44100s {
  pour chaque objet {
    lire ses entrées;
    calculer le sample suivant;
    écrire dans ses sorties;
  }
}
```

- \+ simple
- \+ rapide (ce qui rentre sort après 1/44100s)
- – coûteux (le temps processeur à réserver doit être > au temps de calcul du sample le plus coûteux)

Traitement par bloc

Processus effectif de calcul de Pure Data (ultra-simplifié) :

```
toutes les 512/44100s {
  pour chaque objet {
    lire 512 samples dans les entrées;
    les traiter en bloc;
    écrire 512 samples dans les sorties;
  }
}
```

Traitement par bloc

Processus effectif de calcul de Pure Data (ultra-simplifié) :

```
toutes les 512/44100s {
  pour chaque objet {
    lire 512 samples dans les entrées;
    les traiter en bloc;
    écrire 512 samples dans les sorties;
  }
}
```

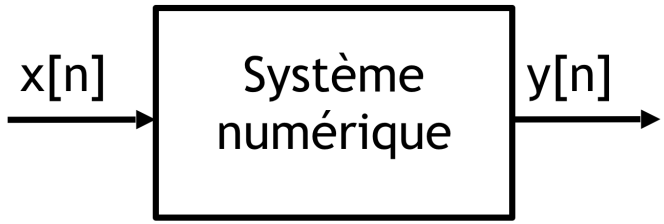
- \+ moins coûteux (le temps de calcul est amorti sur 512 samples)
- – latence de 512/44100s
- ⇝ 512 samples = un *bloc*

Synthèse sonore

Filtrage

Définition

Un filtre est un système numérique linéaire invariant dans le temps



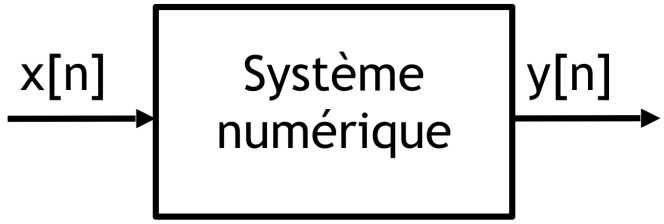
linéaire $A \cdot x[n] \xrightarrow{\text{filtre}} A \cdot y[n]$

invariant dans le temps $x[n-T] \xrightarrow{\text{filtre}} y[n-T]$

Filtrage

Définition

Un filtre est un système numérique linéaire invariant dans le temps



linéaire $A \cdot x[n] \xrightarrow{\text{filtre}} A \cdot y[n]$

invariant dans le temps $x[n-T] \xrightarrow{\text{filtre}} y[n-T]$

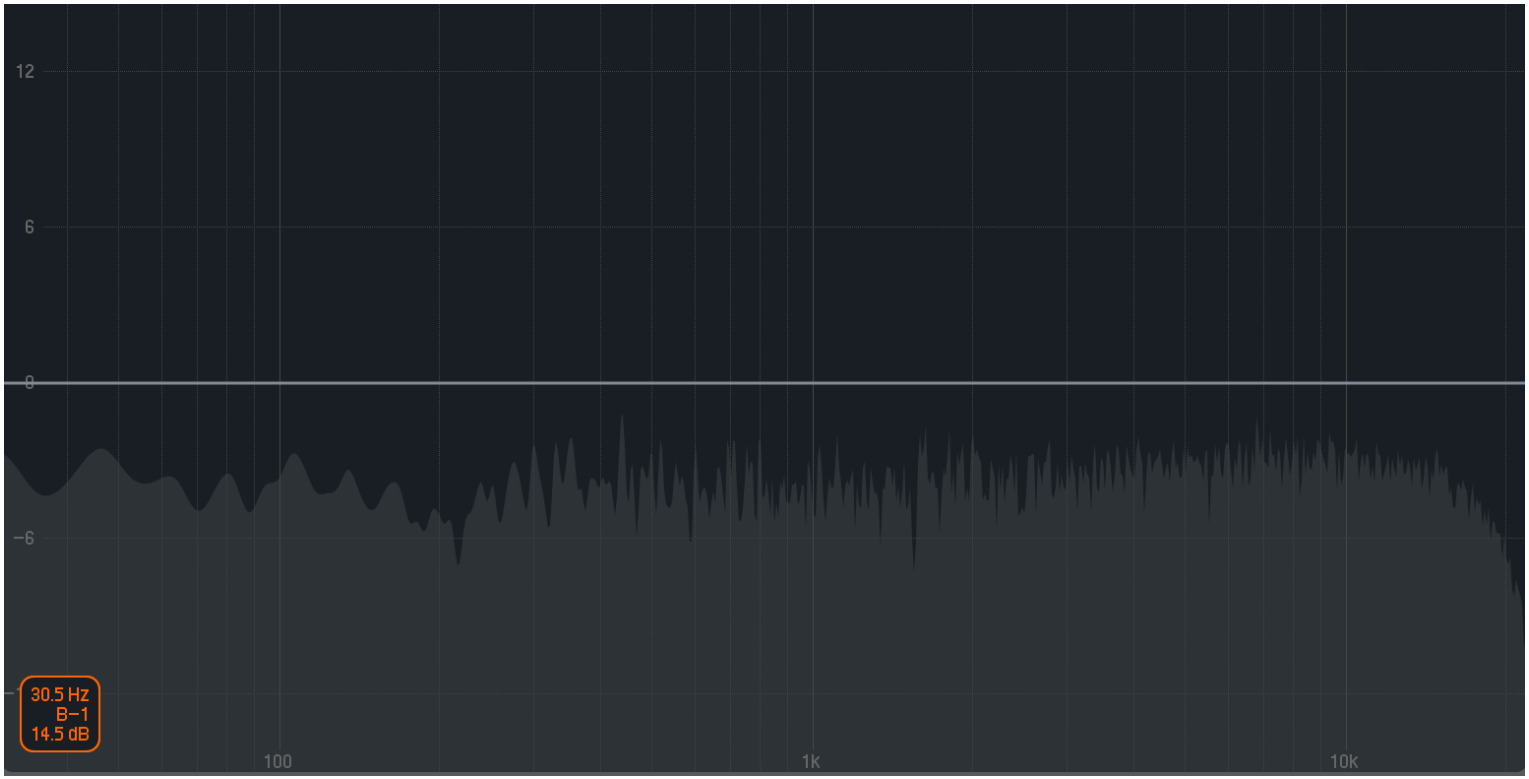
En pratique

Un filtre modifie le contenu spectral d'un signal en atténuant/amplifiant des fréquences données.

Filtrage

Exemple

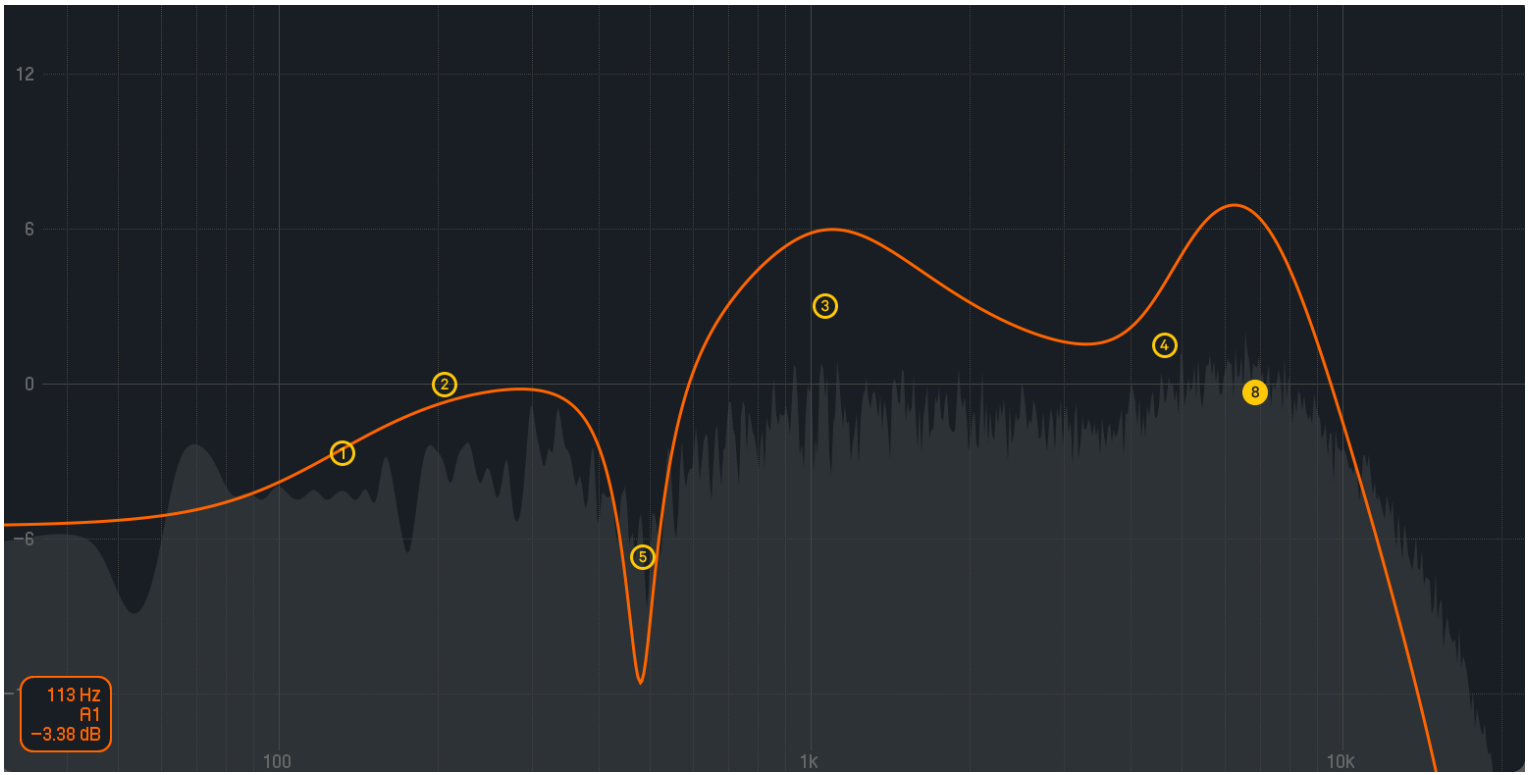
Filtrage de bruit blanc



Filtrage

Exemple

Filtrage de bruit blanc

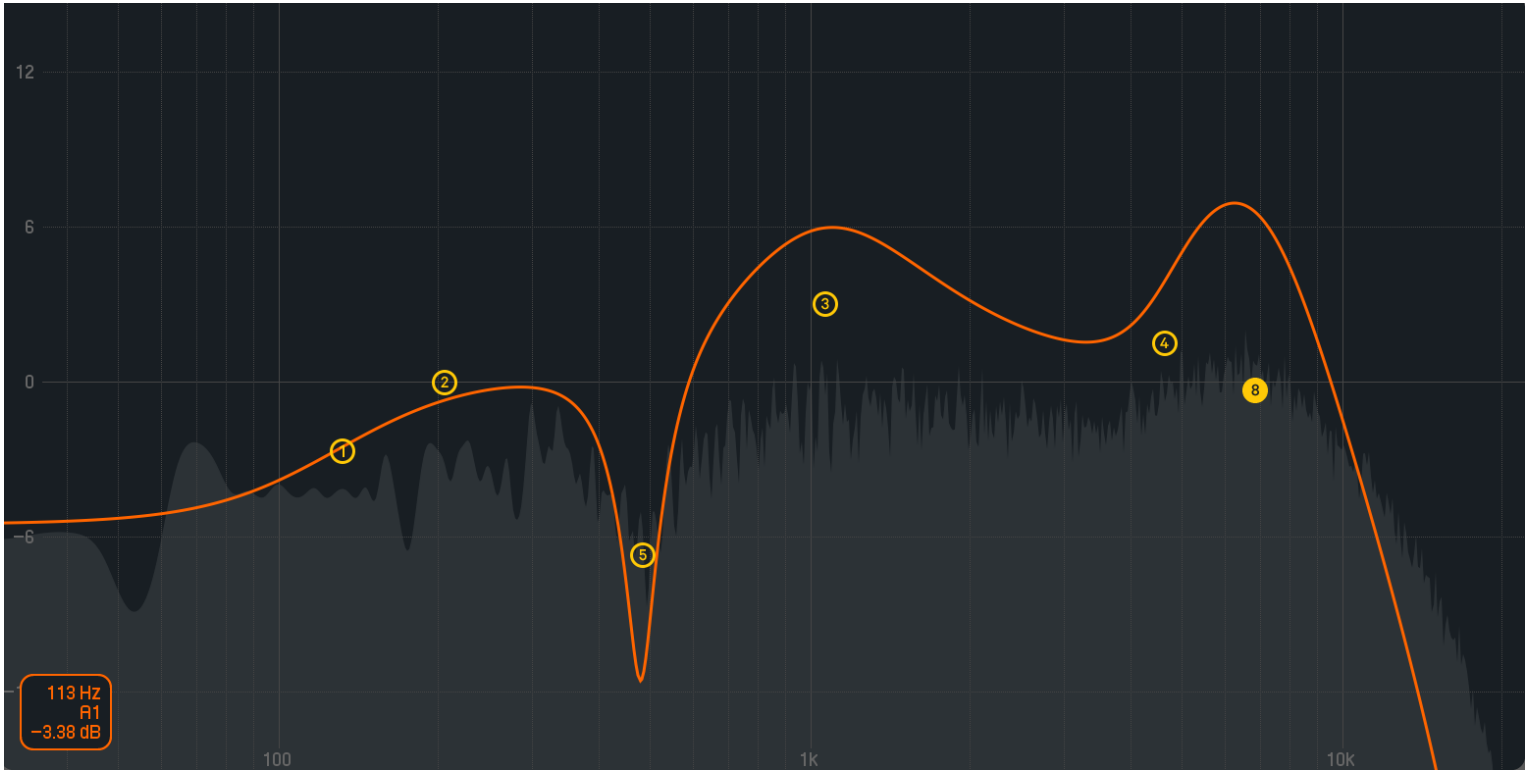


On multiplie son spectrogramme par une *enveloppe spectrale*

Filtrage

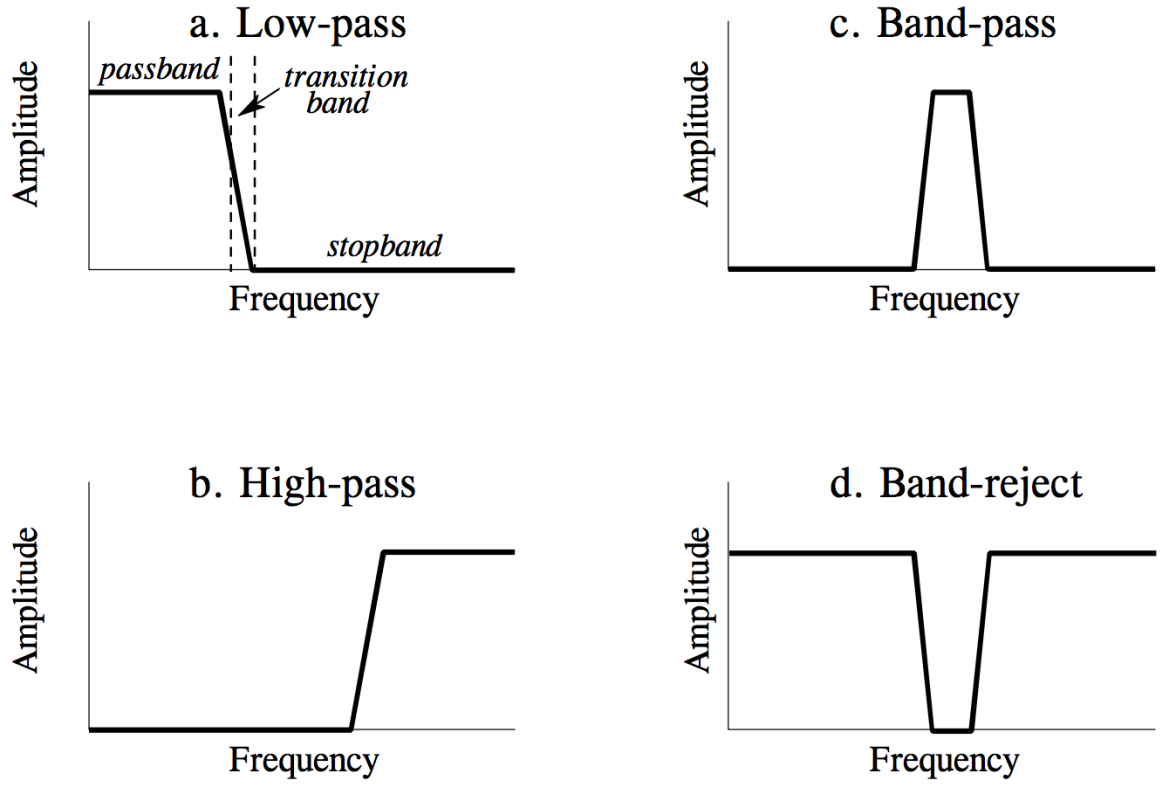
Exemple

Filtrage de bruit blanc



On multiplie son spectrogramme par une *enveloppe spectrale*
Un filtre = la donnée d'une enveloppe spectrale

Typologie des filtres usuels



- le *passe-bas* coupe les fréquences *au dessus* d'un seuil
- le *passe-haut* coupe les fréquences *en dessous* d'un seuil
- le *coupe-bande* coupe autour d'une fréquence
- le *passe-bande* coupe toutes les fréquences *sauf* autour d'une

Typologie des filtres usuels

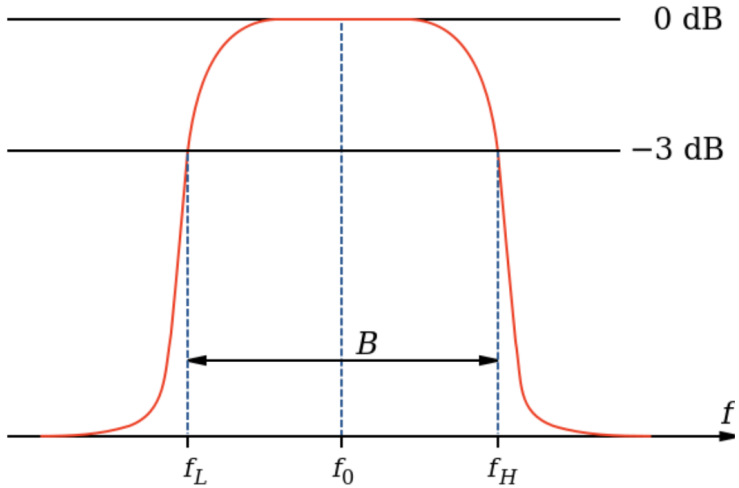
Caractéristiques d'un filtre

fréquence de coupure f_0 (ou *cutoff*) fréquence à partir de laquelle la transition est amorcée
(à -3dB après l'amorce de la descente)

pente "vitesse" de transition (mesurée en dB/Oct)

facteur de qualité pour les filtres passe/coupe-bande :

$$Q = \frac{f_0}{f_H - f_L}$$



Exemple historique : le filtre résonant

Invention de Bob Moog

- circuit analogique simple et omniprésent
- passe-bas + passe-bande
- deux contrôles :
  - fréquence de coupure f_0
  - facteur de qualite Q (ou *emphasis*)
- la *résonance* est une bosse à f_0 de hauteur proportionnelle à Q

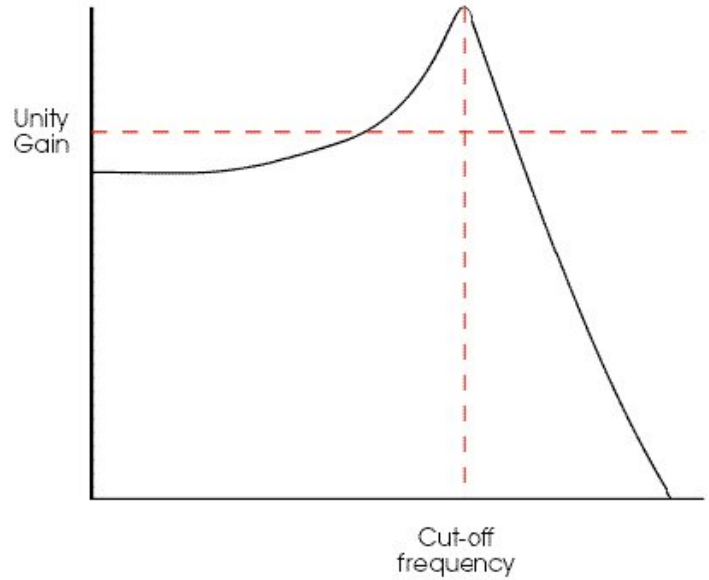


Figure 13: A typical resonant low-pass filter response.

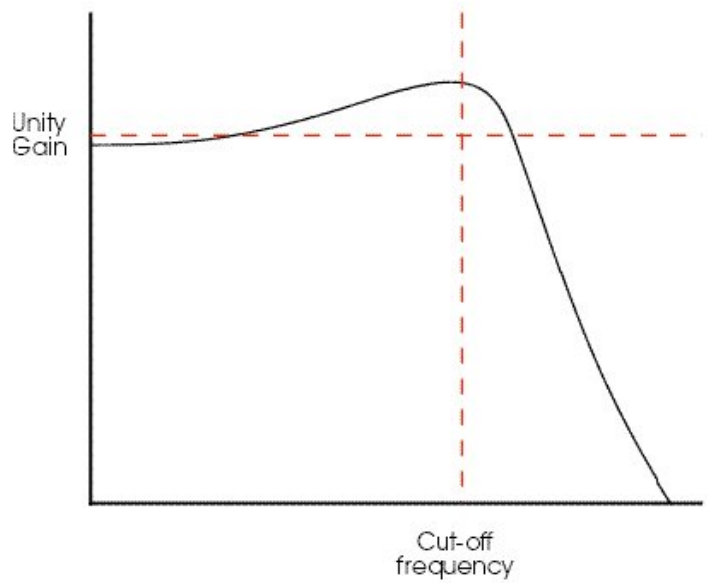


Figure 14: Resonant low-pass filter with low Q.

Filtres en Pure Data

`[09.filters.pd]` et `[10.vcf.pd]`

`lop~` passe-bas 6dB/Oct

- entrée 1 : signal à filtrer
- entrée 2 : (ou argument) fréquence de coupure
- sortie : signal filtré

`hip~` passe-haut 6dB/Oct

`bp~` passe-bande 6dB/Oct

`vcf~` filtre résonant 12dB/Oct

- entrée 1 : signal à filtrer
- entrée 2 : (ou argument) fréquence de coupure
- entrée 3 : facteur de qualité
- sortie 1 : signal filtré passe-bande
- sortie 2 : signal filtré passe-bas

Filtrage dans le domaine temporel

Quel est l'effet du filtrage sur e.g. une enveloppe ?

Filtrage d'un slider `[11. time-filter.pd]`

- un filtre passe-bas ajoute de l'inertie au mouvement
  (plus sa fréquence est basse, plus il y a d'inertie)
- un filtre passe-haut ramène le signal vers 0
  (plus sa fréquence est haute, plus le retour à 0 est rapide)

Filtrage dans le domaine temporel

Quel est l'effet du filtrage sur e.g. une enveloppe ?

Filtrage d'un slider `[11. time-filter.pd]`

- un filtre passe-bas ajoute de l'inertie au mouvement
  (plus sa fréquence est basse, plus il y a d'inertie)
- un filtre passe-haut ramène le signal vers 0
  (plus sa fréquence est haute, plus le retour à 0 est rapide)

Application

le filtre passe-bas "adoucit" un signal

- atténue le bruit
- atténue l'"effet d'escalier"
  ex : un contrôle capturé avec peu de résolution ou échantillonné lentement

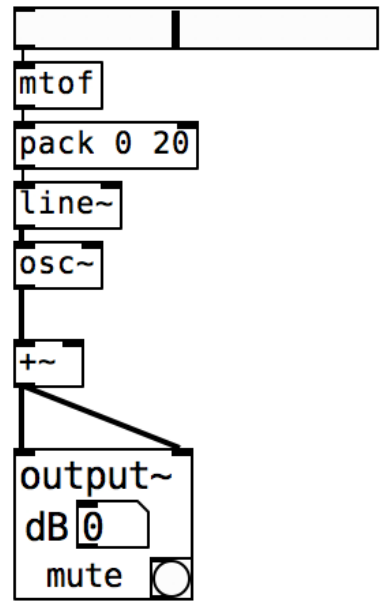
Filtrage dans le domaine temporel

Filtrage avec `line~` [12. line-filter.pd]

Alternative : générer un segment de droite entre chaque valeur.

`pack` fabrique une liste de n valeurs avec ses n arguments
(ici : émet la liste "x 20" pour chaque valeur x du slider)

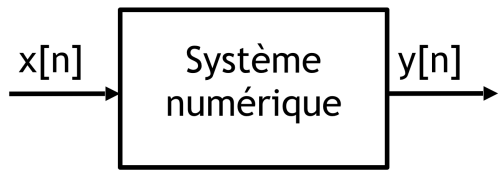
Chaque segment met 20ms à atteindre la valeur du slider



Bonus : Comment implémente-t-on un filtre ?

Rappel

Un filtre est un système numérique linéaire invariant dans le temps

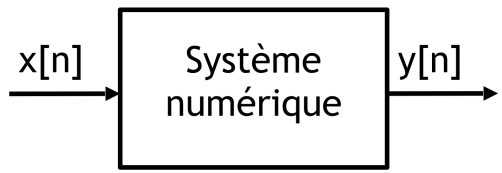


Il ne connaît que les valeurs successives de $x[n]$ et de $y[n]$.

Bonus : Comment implémente-t-on un filtre ?

Rappel

Un filtre est un système numérique linéaire invariant dans le temps



Il ne connaît que les valeurs successives de $x[n]$ et de $y[n]$.

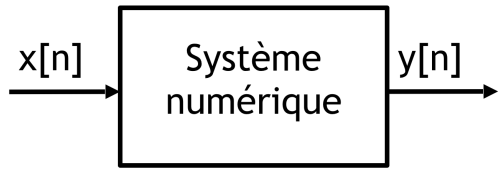
Définition

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + \dots \\ + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \dots$$

Bonus : Comment implémente-t-on un filtre ?

Rappel

Un filtre est un système numérique linéaire invariant dans le temps



Il ne connaît que les valeurs successives de $x[n]$ et de $y[n]$.

Définition

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + \dots \\ + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \dots$$

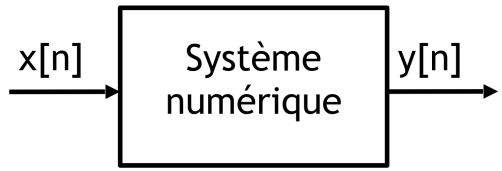
Comment trouve-t-on les coefficients a_i et b_i

pour une enveloppe spectrale donnée ?

Bonus : Comment implémente-t-on un filtre ?

Rappel

Un filtre est un système numérique linéaire invariant dans le temps



Il ne connaît que les valeurs successives de $x[n]$ et de $y[n]$.

Définition

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + \dots \\ + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \dots$$

Comment trouve-t-on les coefficients a_i et b_i

pour une enveloppe spectrale donnée ?

⇝ par les maths (transformée Z)

Synthèse soustractive

Le principe derrière l'écrasante majorité des synthétiseurs

Face avant du Minimoog

La chaîne VCO→VCF→VCA

VCO oscillateur(s) modulé par un clavier
(formes d'onde riches en harmoniques : scie, carré, bruit)

VCF filtre résonant modulé par une enveloppe ADSR

VCA atténuateur modulé par une enveloppe ADSR

Synthèse soustractive en Pure Data `[13.saw-square.pd]`

Génération de formes d'ondes riches

Par convention les oscillateurs vont de -1 à +1 :

dent de scie

- `phasor~` (va de 0 à 1)
- multiplie par 2 (va de 0 à 2)
- enlève 1 (va de -1 à 1)

carrée

- `phasor~` (va de 0 à 1)
- compare à 0.5
  (1 si x<0.5, 0 sinon)
- multiplie par 2 (va de 0 à 2)
- enlève 1 (va de -1 à 1)

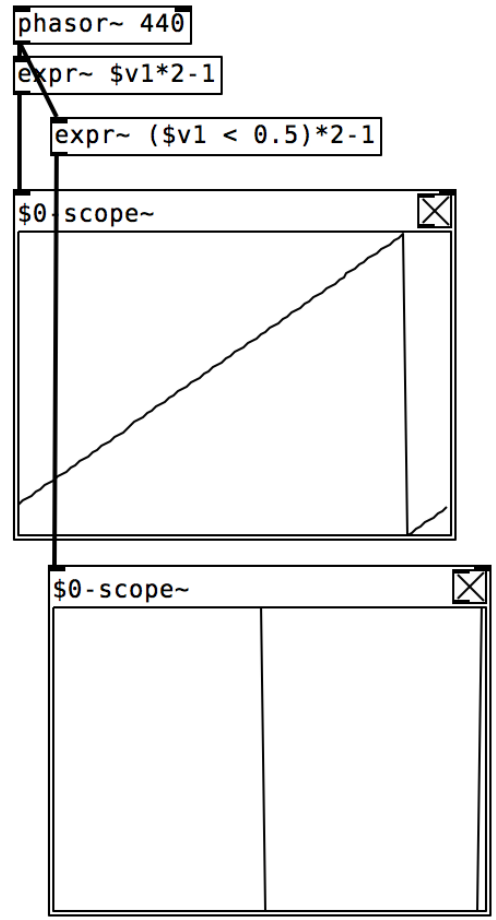
Synthèse soustractive en Pure Data [13.saw-square.pd]

Génération de formes d'ondes riches

Par convention les oscillateurs vont de -1 à +1 :

dent de scie
- `phasor~` (va de 0 à 1)
- multiplie par 2 (va de 0 à 2)
- enlève 1 (va de -1 à 1)

carrée
- `phasor~` (va de 0 à 1)
- compare à 0.5
  (1 si x<0.5, 0 sinon)
- multiplie par 2 (va de 0 à 2)
- enlève 1 (va de -1 à 1)



`expr~`

Calcul arithmétique sur un signal

argument l'expression à calculer, avec `$v1`, `$v2` etc. les entrées (signal)

Synthèse sonore

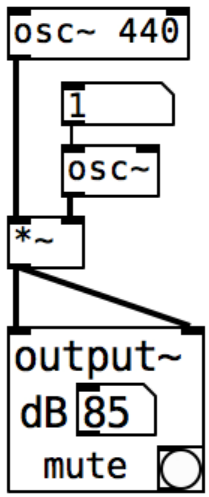
Modulations

Deux techniques issues des radiocommunications
pour créer des timbres *complexes* à partir de signaux *simples* :

- modulation d'amplitude (AM)
- modulation de fréquence (FM)

Modulation d'amplitude [14.mod-amp.pd]

Qu'entend-t-on si on module l'amplitude d'une sinusoïde à vitesse audio ?



Modulation d'amplitude [14.mod-amp.pd]

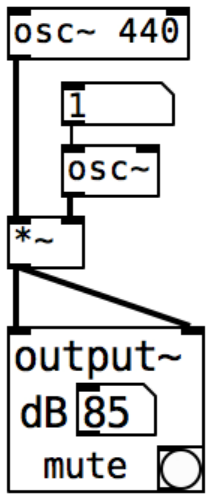
Qu'entend-t-on si on module l'amplitude d'une sinusoïde à vitesse audio ?

Exemple historique

"modulation en anneau" ou modulation de Bode

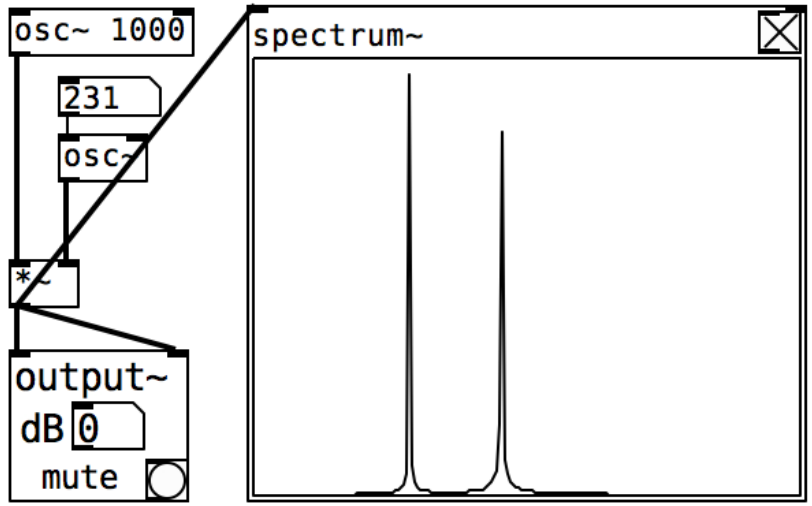
Karlheinz Stockhausen - Mantra (1970)

`https://www.youtube.com/watch?v=vQ4jQyTHU-4`



Spectre de la modulation d'amplitude

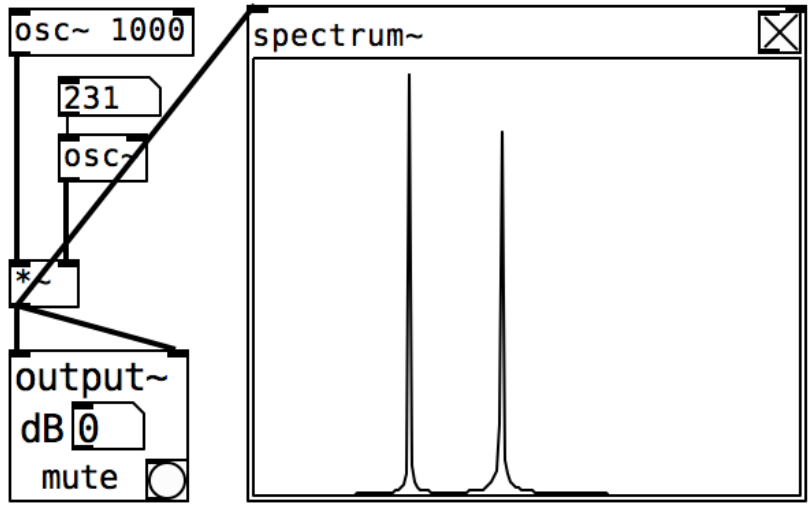
Quel est le spectre d'un signal modulé en amplitude ?



$$\cos(x)\cos(y) = \frac{1}{2}\left(\cos(x+y) + \cos(x-y)\right)$$

Spectre de la modulation d'amplitude

Quel est le spectre d'un signal modulé en amplitude ?



$$\cos(x)\cos(y) = \frac{1}{2}\left(\cos(x+y) + \cos(x-y)\right)$$

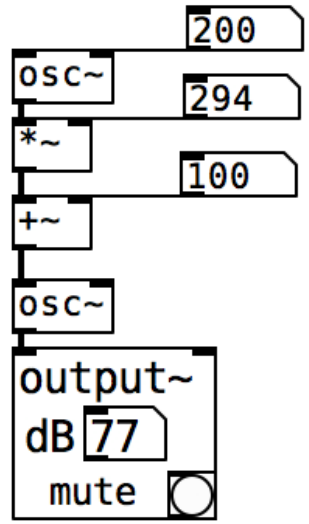
Bandes latérales

On obtient 2 copies du spectre du signal modulé (*bandes latérales*) :

- une décalée de la fréquence porteuse (1000Hz + y)
- une décalée et retournée (1000Hz - y)

Modulation de fréquence [15.mod-freq.pd]

Qu'entend-t-on si on module la **fréquence** d'une sinusoïde à vitesse audio ?



Modulation de fréquence [15.mod-freq.pd]

Qu'entend-t-on si on module la **fréquence** d'une sinusoïde à vitesse audio ?

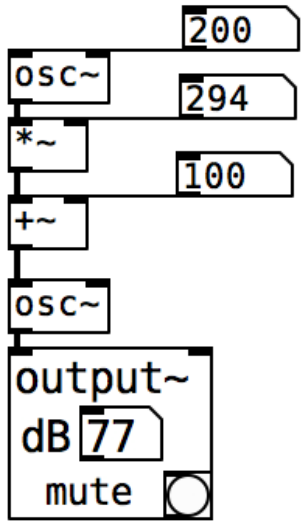
Exemple historique

John Chowning - Turenas (1972)

`https://www.youtube.com/watch?v=kSbTOB5ft5c`

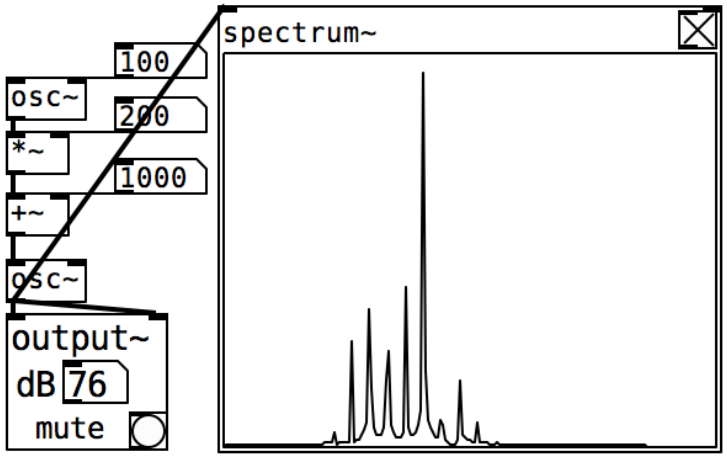
Avantages

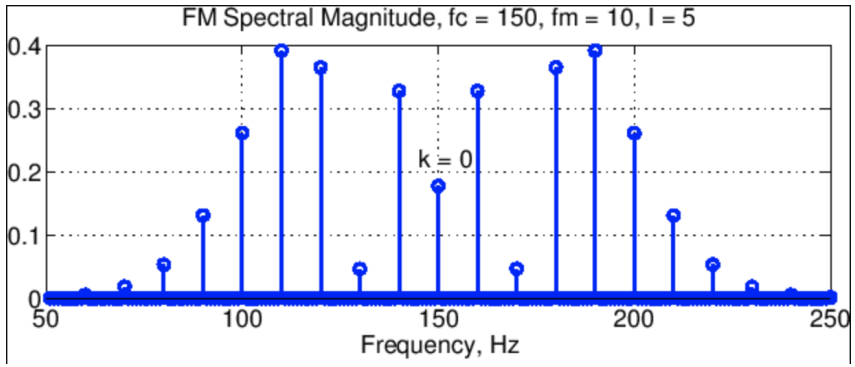
- timbres riches avec peu de calcul
- grande variété de timbres



Spectre de la modulation de fréquence

Quel est le spectre d'un signal modulé en fréquence ?





$$\cos(f_c t + B\sin(f_m t)) = \sum_{n=-\infty}^{+\infty} J_n(B)\cos((f_c + nf_m)t)$$

Fonction de Bessel $J_n(B)$

Son enveloppe spectrale est la fonction de Bessel

Synthèse FM

1967 J. Chowning invente la synthèse FM

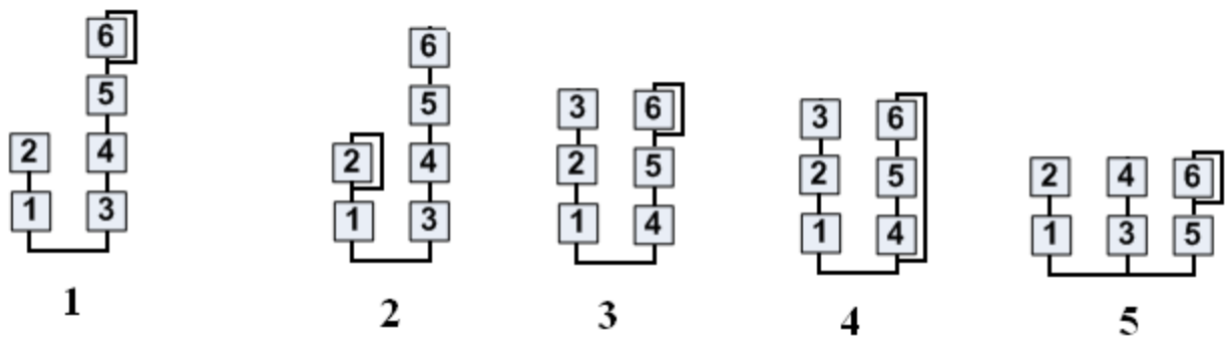
1974 Stanford vend le brevet à Yamaha

1981 Yamaha sort le GS1 (échec commercial)

1983 Yamaha sort le DX7

- synthé le plus populaire au monde
- 6 sinusoïdes modulant/modulées
- 16 voix de polyphonie

Opérateurs et algorithmes



Synthèse sonore

Pure Data : à propos de polyphonie

Pour l'instant, on n'a qu'une seule voix de synthèse :
Un seul son à chaque instant (monophonie)
Pas de notion intégrée à pd de polyphonie

Pure Data : à propos de polyphonie

Pour l'instant, on n'a qu'une seule voix de synthèse :
Un seul son à chaque instant (monophonie)
Pas de notion intégrée à pd de polyphonie

⇝ On doit faire à la main !

(dupliquer notre voix n fois, répartir les notes entrantes sur les n voix)

Pure Data : à propos de polyphonie

Pour l'instant, on n'a qu'une seule voix de synthèse :
Un seul son à chaque instant (monophonie)
Pas de notion intégrée à pd de polyphonie

⇝ On doit faire à la main !

(dupliquer notre voix n fois, répartir les notes entrantes sur les n voix)

Ingrédients

abstraction	faire de patches entiers des objets maison
`trigger`	contrôler l'ordonnancement des message
`route`	envoie des messages dans des sorties séparées
`float`	stocke un nombre

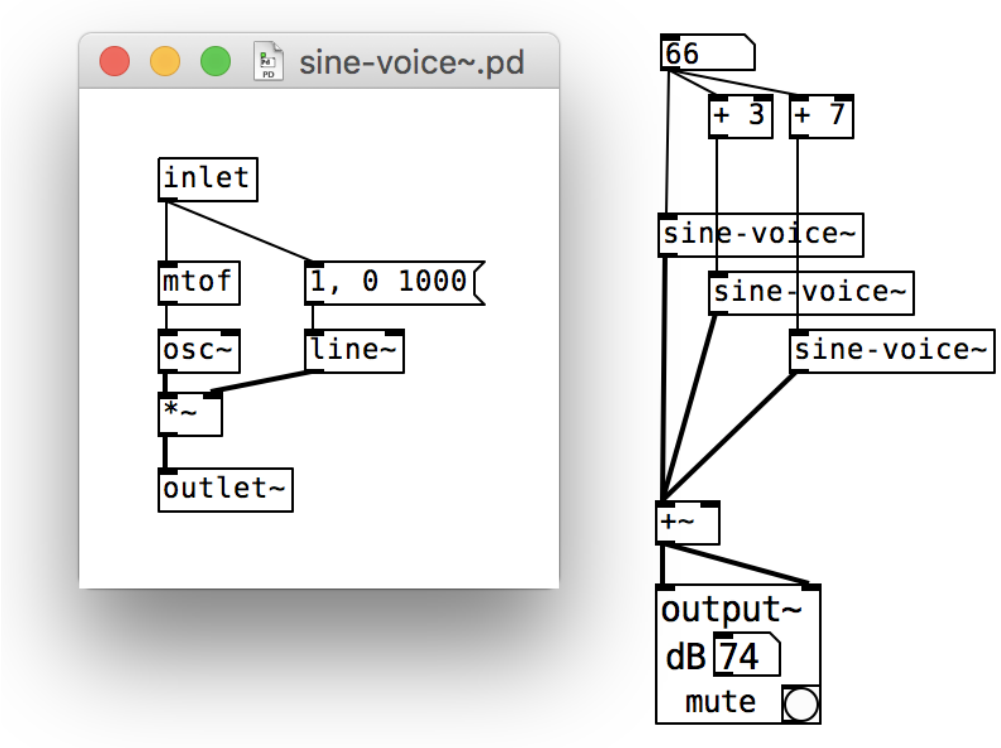
Abstractions 16.abstraction.pd

Tout fichier `fichier.pd` dans le répertoire courant peut être utilisé comme un objet

`inlet(~)` entrée message/signal de l'objet

`outlet(~)` sortie message/signal de l'objet

Exemple



Nombres, listes et routage

+, –, *, / opérations arithmétiques sur les messages nombres

entrée 2 stocke l'opérande 2 (entrée *froide*)

entrée 1 déclenche le calcul (entrée *chaude*)

Nombres, listes et routage

`+`, `-`, `*`, `/` opérations arithmétiques sur les messages nombres

- entrée 2 stocke l'opérande 2 (entrée *froide*)
- entrée 1 déclenche le calcul (entrée *chaude*)

`float` stocke un nombre

- entrée 1 `bang` émet le nombre stocké ;
  [nombre] change le nombre et l'émet
- entrée 2 change le nombre stocké (n'émet rien)

Nombres, listes et routage

`+`, `-`, `*`, `/` opérations arithmétiques sur les messages nombres

- entrée 2 stocke l'opérande 2 (entrée *froide*)
- entrée 1 déclenche le calcul (entrée *chaude*)

`float` stocke un nombre

- entrée 1 `bang` émet le nombre stocké ;
  [nombre] change le nombre et l'émet
- entrée 2 change le nombre stocké (n'émet rien)

`route` route message de la forme "`n m`" vers sa *n*-ième sortie

Nombres, listes et routage

`+, -, *, /` opérations arithmétiques sur les messages nombres

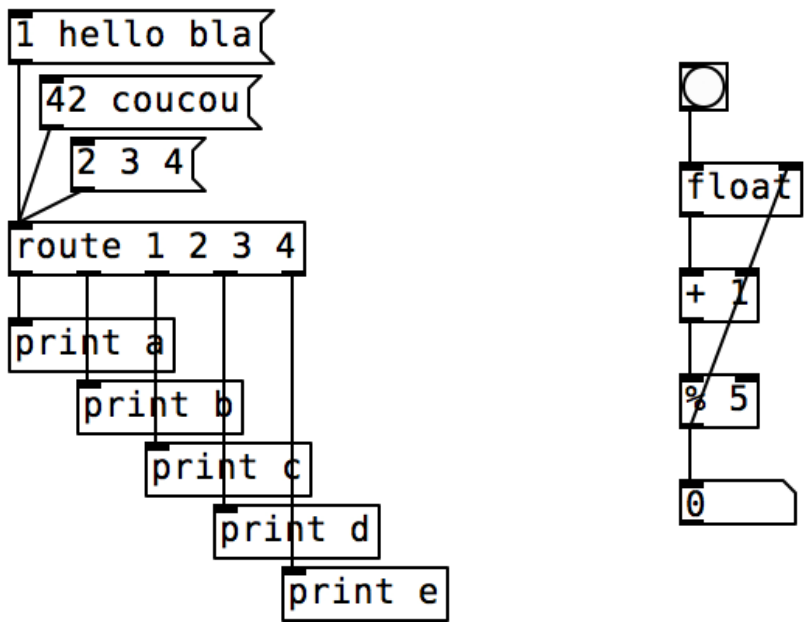
- entrée 2 stocke l'opérande 2 (entrée *froide*)
- entrée 1 déclenche le calcul (entrée *chaude*)

`float` stocke un nombre

- entrée 1 `bang` émet le nombre stocké ;
  [nombre] change le nombre et l'émet
- entrée 2 change le nombre stocké (n'émet rien)

`route` route message de la forme "`n m`" vers sa *n*-ième sortie

Exemple `[17.float-route.pd]`



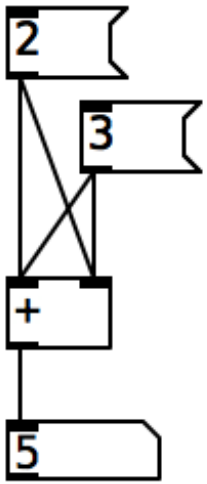
Ordonnancement des messages

Problème

On ne choisit pas l'ordre d'arrivée des messages

Exemple

Comment peut-on arriver à la situation ci-contre ?



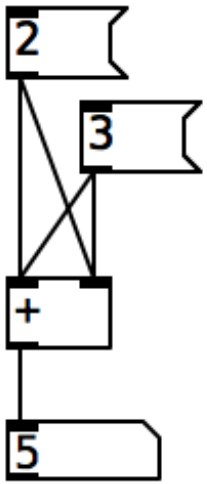
Ordonnancement des messages

Problème

On ne choisit pas l'ordre d'arrivée des messages

Exemple

Comment peut-on arriver à la situation ci-contre ?



Solution

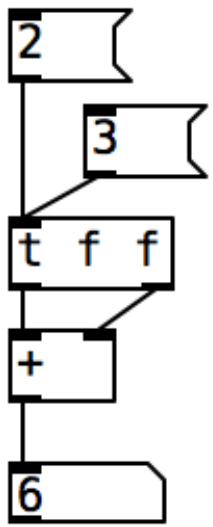
Forcer l'ordre d'arrivée des message

`trigger` ou `t` émet son entrée n de droite à gauche

arguments : une liste de n symboles (`f` : nombre ou `b` : bang ou `l` : liste)

entrée : le message à ordonnancer

sorties : les n messages, émis de droite à gauche



Synthétiseur polyphonique 19.poly.pd