S1916 - Analyse et Compression du Signal Audionumérique - Examen

Sébastien Boisgérault

Jeudi 20 Mars 2014

Table des Matières

Modalités de l'Examen

- **Autorisés:** documents (papier, électronique) et calculette (ou équivalent),
- **Interdits:** interpréteur Python et accès Internet.

Streaming Audio

Je souhaite configurer l'ordinateur de mon domicile en serveur audio afin d'écouter de la musique en temps-réel (*streaming audio*) quand je suis en déplacement. Des mesures faites à mon domicile montrent que le débit montant de données que permet mon fournisseur d'accès à Internet est de l'ordre de 100 kB/s (kilo-octet par seconde).

Puis-je diffuser du son audio "qualité CD" sans compression ? Est-ce que la diffusion de données comprimées sans perte résout le problème ? Et si je souhaite que des amis utilisent mon serveur audio en même temps que moi ?

Entiers Signés et Non Signés

Donner la valeur produite par `BitStream(-1, int8).read(uint8)`.

Non Mais Allo Quoi !

Quand elle parle, Nabilla utilise le vocabulaire de la télé-réalité soit 600 mots, (cf. article "La télé-réalité fait chuter les notes des ados" du Nouvel Observateur), contre 5000 mots en moyenne nationale. En tenant compte d'un débit de 200 mots/minute, et avec les informations disponibles, estimer le débit d'information produit par Nabilla en kb/s. Pourquoi est-ce que cette estimation grossière du débit d'information est encore trop grande ?

Entiers Python

Si la classe `BitStream` sait écrire en binaire les types entiers de Numpy (`int8`, `uint8`, `int16`, etc.) en utilisant des codes de taille fixe, elle ne sait que faire des entiers standards de Python, qui sont non bornés. Pourquoi est-ce que `BitStream` n'utilise pas tout simplement la représentation binaire des entiers (précédée par un bit de signe) ?

Les entiers standards de Python sont en fait représentés par deux types distincts: `int` qui représente les entiers signés sur 32 bits et `long` qui est utilisé lorsque l'entier est plus petit que -2^{31} ou plus grand que $2^{31}-1$. Si l'on suppose que les entiers de type `int` sont beaucoup plus fréquents que les entiers de type `long`, quelle méthode cela suggère-t'il pour écrire les entiers standards sous forme binaire ?

Bytecode Java

Les fichiers Java d'extension ".class" débutent par les bits

```
11001010 11111110 10111010 10111110
```

Quelle est la représentation en hexadécimal de cette séquence ?

Représentation en Virgule Flottante des Réels.

Le standard IEEE 754 pour l'arithmétique binaire à virgule flottante détermine la manière classique de représenter des réels sur 64 bits. Une telle séquence de bits est interprétée comme trois entiers non signés:

- le bit de signe $s \in \{0, 1\}$,
- l'exposant biaisé $e \in \{0, ..., 2^{11} - 1\}$,
- la fraction $f \in \{0, ..., 2^{52} - 1\}$.

puis la valeur réelle correspondante x est donnée par la formule:

$$x = (-1)^s \times 2^{e-1023} \times (1 + f/2^{52})$$

Lorsqu'un réel x ne peut être représenté exactement par une telle valeur, on lui associe par défaut la valeur la plus proche qui le soit, notée $[x]$.

- Est-ce que les réels 1.0, 0.5, π et 0.1 sont représentables exactement dans ce schéma ? (autrement dit, est-ce que $[x] = x$ pour $x \in \{1.0, 0.5, \pi, 0.1\}$?)
- Compléter le code suivant en remplaçant les "`?`" par des valeurs numériques.

  ```
  def fraction(x):
      input = BitStream(x, float64)
      input.read(bool, ?)
      output = BitStream(? * [False], bool)
      output.write(input.read(bool, ?), bool)
      return output.read(uint64)
  ```

  On rappelle que si `n` est un entier, l'expression `n * [False]` désigne une liste répétant `n` fois l'élément `False`.

- Pour $x = 0.1$, on a $s = 0$, $e = 1019$ et $f = 2702159776422298$.
  Quel est le pas $\Delta(x)$ de la quantification $[\,\cdot\,]$ en $x = 0.1$?

Résolution des Quantificateurs

On dit qu'un quantificateur scalaire $[\cdot]_1$ a une résolution inférieure à un quantificateur scalaire $[\cdot]_2$ – ce que l'on note $[\cdot]_1 \leq [\cdot]_2$ – si

$$[\cdot]_2 \circ [\cdot]_1 = [\cdot]_1.$$

- Montrer que si $[\cdot]_1 \leq [\cdot]_2$, les opérateurs $[\cdot]_2 \circ [\cdot]_1$ et $[\cdot]_1 \circ [\cdot]_2$ sont des quantificateurs scalaires.
- Montrer que la relation $\leq$ définie sur l'ensemble des quantificateurs scalaires est réflexive et transitive, c'est-à-dire que $\forall [\cdot],\ [\cdot] \leq [\cdot]$ et $\forall [\cdot]_1, \forall [\cdot]_2, ([\cdot]_1 \leq [\cdot]_2 \ \wedge\ [\cdot]_2 \leq [\cdot]_3) \implies [\cdot]_1 \leq [\cdot]_3$.
- L'arrondi à l'entier inférieur $\lfloor \cdot \rfloor$ a-t'il une résolution inférieure à l'arrondi à l'entier supérieur $\lceil \cdot \rceil$? L'inverse est-il vrai ? La relation $\leq$ est-elle une relation d'ordre ?
- Comparer deux à deux la résolution des quantificateurs suivants:
  - 8-bit uniforme sur $[-1 - 2^{-8}, +1 - 2^{-8}]$,
  - 16-bit uniforme sur $[-1 - 2^{-16}, +1 - 2^{-16}]$,
  - μ-law (G.711).

  Le pré-ordre $\leq$ est-il total ? Autrement dit, pour toute paire de quantificateurs $[\cdot]_1$ et $[\cdot]_2$, a-t'on soit $[\cdot]_1 \leq [\cdot]_2$, soit $[\cdot]_2 \leq [\cdot]_1$?

Quantification Non Linéaire

La fonction `index` est un quantificateur direct qui associe à tout flottant un entier non signé sur 8 bits:

```
def index(x):                   # x: float
    x = min(1.0, max(0.0, x))   # x: float in [0,1]
    if x <= 0.5:
        y = 1.8 * x
    else:
        y = 0.2 * x + 0.8       # y: float in [0,1]
    j = floor(2**8 * y)
    return uint8(j)             #  : uint8
```

- Est-elle plus adaptée au traitement de nombre dont les valeurs sont majoritairement petites (inférieures à 0.5) ou grandes (supérieures) ?
- Implémenter le quantificateur inverse associé à `index`.
- Certaines valeurs de $x \in [0, 1]$ sont très éloignées de $[x]$. Exhiber une de ces valeurs, puis corriger la fonction `index` pour éviter ce phénomène.

Conversion Analogique-Digitale

Le PCM422 de Texas Instruments est un convertisseur analogique-digital (ADC) avec une plage dynamique (*dynamic range*) de 124 dB. Le choix de ce matériel est-il adapté pour de la musique qui sera gravée sur un CD audio ?

Lors de l'acquisition d'un signal dans un format haute-résolution 96kHz/24bit, quel va être l'élément limitant la qualité du résultat ? La profondeur de bits utilisée pour la représentation des signaux ou le choix du PCM422 ?

Protection Auditive

Des protections auditives atténuent plus ou moins les sons selon leur fréquence. L'efficacité de protections en mousse "Disco" de la marque Quies est spécifié comme suit:

Fréquence (Hz)	Atténuation Moyenne (dB)
125	36.0
250	36.7
500	39.0
1000	36.0
2000	37.0
4000	46.7
8000	44.6

Est-ce que ces protections auditives rendent inaudible toute conversation ? On considérera qu'une conversation peut générer un son quelconque dans la bande de fréquence 0-4000 Hz et de niveau sonore maximal 60 dB.

Harmoniques des Signaux Peignes

Considérons un signal $x(t)$ de fréquence d'échantillonnage Δf qui soit nul sauf à des instants régulièrement espacés où il vaut 1. Soit f_1 la fréquence d'occurrence de ces instants et n le nombre d'échantillon par période de $x(t)$, soit $n = \Delta f / f_1$.

Le signal peut être représenté comme la superposition de n tons purs – appelés harmoniques – dont les fréquences f_k sont des multiples de la fréquence f_1. Dans notre cas, chaque harmonique est de même niveau sonore: on a comme transformée de Fourier du signal $x(t)$:

$$x(f) \propto \sum_{k=0}^{n-1} \delta(f - k f_1)$$

Toutes les harmoniques présentes dans cette décomposition spectrale sont-elles nécessairement audibles ? On prend $\Delta f = 16$ kHz et $f_1 = 500$ Hz. Déterminer

quelles harmoniques on peut sans impact supprimer du signal original (on utilisera le modèle psycho-acoustique de Fletcher).

Prédiction Linéaire de Signaux Synthétiques

Le résidu $e(t)$ de la prédiction linéaire d'un signal $x(t)$, $t \in \mathbb{Z}\Delta t$, associé aux coefficients $a_1, \ldots\ a_m$, est donné par:

$$e(t) = x(t) - \sum_{k=1}^{m} a_k x(t - k\Delta t)$$

A quel condition portant sur les a_k la prédiction linéaire est-elle exacte pour tous les tons purs de fréquence f (d'amplitude A et de phase ϕ arbitraires) ? Montrer que satisfaire cette condition revient à imposer deux racines à un polynôme dont les coefficients dépendent des a_k. Quel ordre m faut-il sélectionner si l'on souhaite garantir qu'il existe un jeu de coefficients $a_1, ..., a_m$ réalisant une prédiction exacte pour tout signal composé de n tons purs ?

Bancs de Filtre

Vous êtes chargé d'interpréter le programme de compression audio d'un ingénieur ayant quitté votre entreprise. Ca devrait être simple a priori, vu qu'il y a des commentaires ...

- La première définition du programme est:

```
def analysis(input):
    "Analysis Filter Bank"
    # input is a 1d numpy array.
    subband_1 = 0.5 * (r_[input, 0] + r_[0, input])
    subband_2 = 0.5 * (r_[input, 0] - r_[0, input])
    return [subband_1, subband_2]
```

  Un test rapide dans un interpréteur Python vous rappelle que l'on a:

```
>>> r_[0, [1, 2, 3]]
array([0, 1, 2, 3])
```

  Quelles sont les réponses impulsionnelles et les réponses fréquentielles des deux filtres composant ce banc de filtre ? (La période d'échantillonnage Δt des signaux étant non spécifiée, on fera l'hypothèse que $\Delta t = 1$). Laquelle des sous-bandes `subband_1` ou `subband_2` contient la majeure partie des basses fréquences du signal `input` ?

- La suite du programme contient la définition suivante:

```
def transmogrify(subbands):
    "Transform in a surprising or magical manner"
    # subbands is a list of two 1d numpy arrays of same length.
    output_subbands = []
    for subband in subbands:
        output_subband = zeros(shape(subband))
        output_subband[::2] = subband[::2]
        output_subbands.append(output_subband)
    return output_subbands
```

  Pouvez-vous rédiger une documentation plus explicite ?

- Le programme contient ensuite la définition suivante:

```
def synthesis(subbands):
    "Synthesis Filter Bank"
    # subbands is a list of two 1d numpy arrays of same length.
    output_0 =  r_[subbands[0], 0] + r_[0, subbands[0]]
    output_1 = -r_[subbands[1], 0] + r_[0, subbands[1]]
    return output_0 + output_1
```

  et finalement le code:

```
subbands = analysis(input)
output_subbands = transmogriphy(subbands)
output = synthesis(output_subbands)
```

  Déterminer les 5 premières valeurs du tableau `output` lorsque

```
input = array([1.0, 2.0, 3.0, 4.0, 5.0])
```

  On pourra utiliser le tableau ci-dessous:

x	x[0]	x[1]	x[2]	x[3]	x[4]
input	1.0	2.0	3.0	4.0	5.0
subbands[0]	?	?	?	?	?
subbands[1]	?	?	?	?	?
output_subbands[0]	?	?	?	?	?
output_subbands[1]	?	?	?	?	?
output_0	?	?	?	?	?
output_1	?	?	?	?	?
output	?	?	?	?	?

Si l'on se fie à cet exemple, peut-on parler de reconstruction parfaite ?

Pré-Accentuation

Les signaux audio de voix étant dominés par leur contenu basse-fréquence, il est classique d'égaliser leur contenu fréquentiel (c'est-à-dire d'amplifier l'énergie des hautes fréquences par rapport aux basses fréquences) avant de les comprimer. Cette étape dite de pré-accentuation (*pre-emphasis*) est souvent implémentée en associant au signal original $u(t)$ le signal accentué $y(t)$ donné pour tout $t \in \mathbb{Z}\Delta t$ par la formule

$$y(t + \Delta t) = u(t + \Delta t) - ay(t)$$

où $a \in \mathbb{R}$ et Δt est la période d'échantillonnage commune à $u(t)$ et $y(t)$.

- Le filtre d'entrée $u(t)$ et de sortie $y(t)$ est-il toujours stable ? Parmi les valeurs de a qui garantissent la stabilité, déterminer lesquelles accentuent effectivement le contenu haute fréquence par rapport au contenu basse fréquence. (On pourra prendre comme référence $f = 0$ Hz pour les basses fréquences et $f = \Delta f/2 = 1/(2\Delta t)$ pour les hautes fréquences).
- Un signal est pré-accentué, puis compressé et décompressé. On souhaite comme étape finale du traitement annuler les effets de l'accentuation pour reconstruire le signal original. Compléter le code ci-dessous pour atteindre cet objectif.

```
def de_emphasis(a, u):
    # a: float
    # u: 1d numpy array
    y = ???
    return y
```