計算機科学実験及演習 4「音楽情報処理」

担当：糸山 克寿　　TA：池宮 由楽，矢澤 一樹

2013 年度版 Ver.20131115

1　はじめに

各回の演習を開始する前に，必ず `http://winnie.kuis.kyoto-u.ac.jp/~itoyama/le4-music/` **から最新版の資料をダウンロードすること．**

課題 xxx は必修課題を，**課題** xxx† は任意課題[*1]を表す．

2　音響信号処理の基礎

2.1　デジタル音響信号

音響信号は空気などの媒質の粗密波として伝達され，ビニルレコードやカセットテープなどのアナログな保存媒体はその波形をそのまま保存している．一方，計算機はアナログ信号をそのまま取り扱うことはできないため，アナログ信号を一定の時間で区切り（**標本化** *sampling*）さらに振幅を離散化する（**量子化** *quantizing*）ことで得られる，離散的な表現の信号を取り扱う（図 1）．この周期の逆数を**サンプリング周波数** *sampling rate* と，振幅を何ビットの数値に離散化するかを**量子化ビット数**と呼ぶ．例えば音楽 CD では，サンプリング周波数は 44100Hz = 44.1kHz で量子化ビット数は 16（−32768 から 32767）である．デジタル信号から元のアナログ信号を復元するためには sinc 関数（図 2）を畳み込む．sinc 関数は以下で定義される．

$$\mathrm{sinc}(x) = \frac{\sin \pi x}{\pi x} \tag{1}$$

デジタル信号処理における最も重要な定理の一つが**標本化定理**である．

標本化定理

あるアナログ信号が正弦波の重ね合わせで表現されており，それらの正弦波のうち最も大きい周波数が $F_s/2$ であるとき，F_s 以上のサンプリング周波数で標本化すると，元の信号に含まれる全ての周波数成分をアナログ信号に復元できる．

サンプリング周波数 F_s の半分の周波数を**ナイキスト周波数** *Nyquist frequency* と呼ぶ．標本化定理から，「デジタル信号ではナイキスト周波数を超える周波数成分は表現できない」ことが分かる．ナイキスト周波数を超える周波数成分を含むアナログ信号を標本化すると，**折り返し** *aliasing* と呼ばれる現象を起こし，本来は含まれなかった周波数成分が現れる（図 3）．折り返しを防ぐためには以下のような対策がとられる．

[*1] 成果によっては加点の対象とする

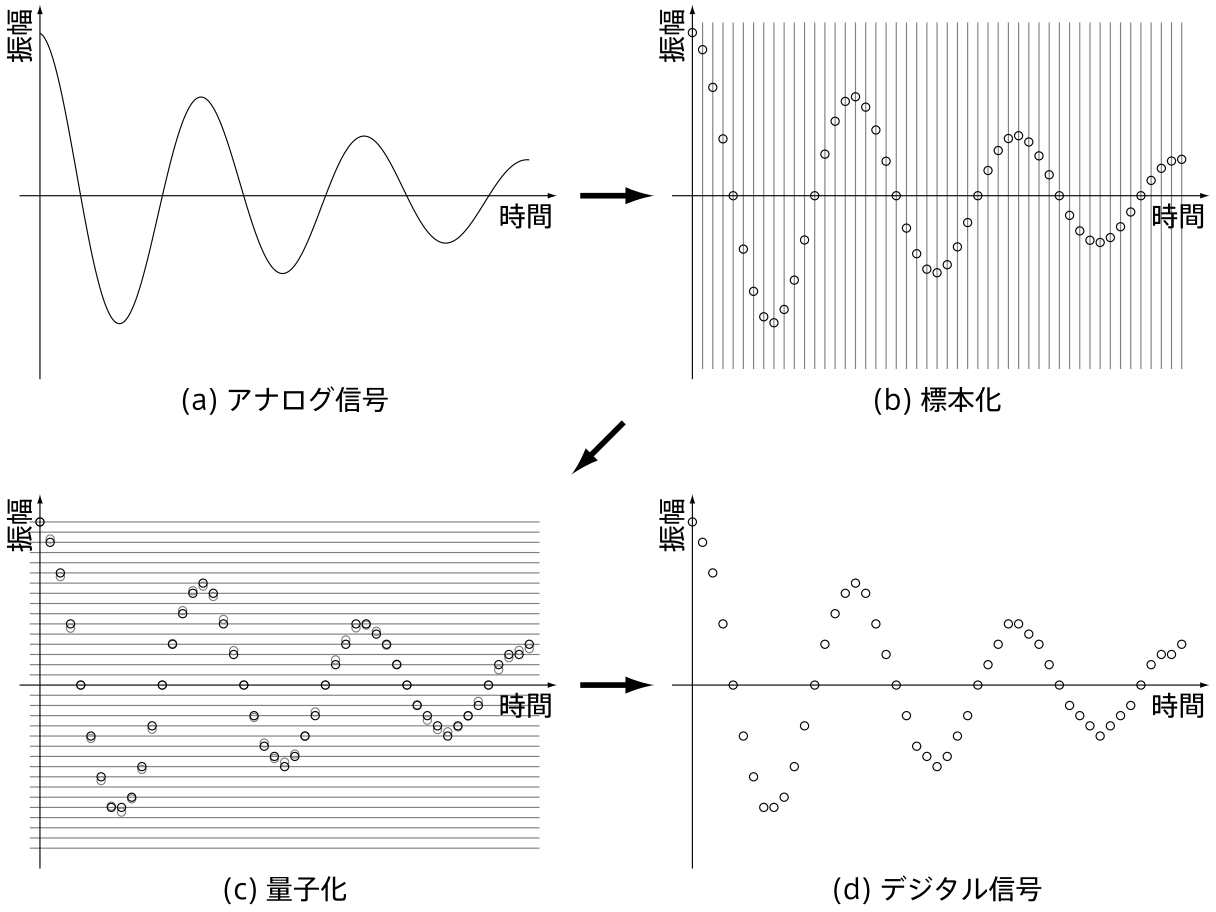


図 1　アナログ信号のデジタイズ

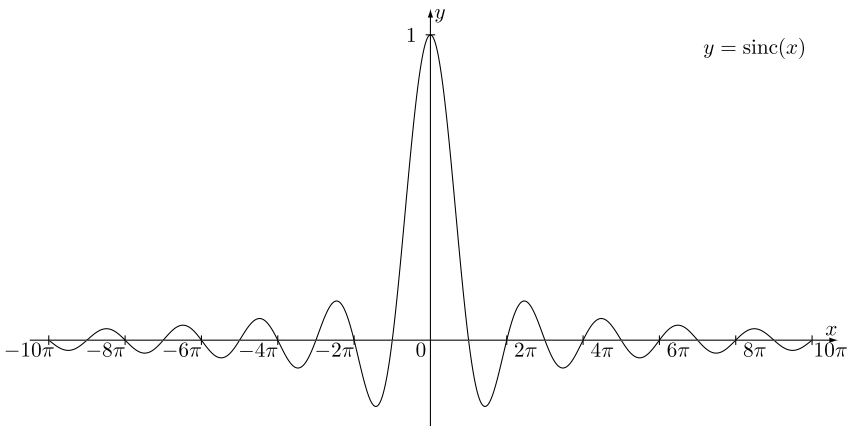


図 2　sinc 関数

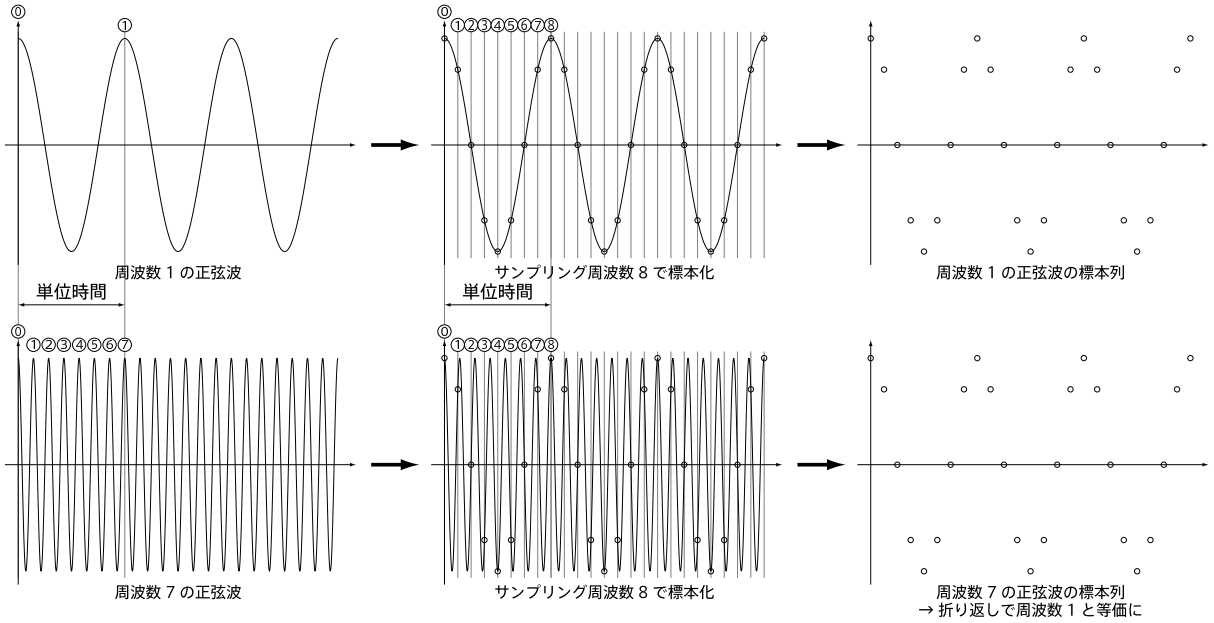


図 3　折り返し雑音の例．周波数 1 と周波数 7 の正弦波をサンプリング周波数 8（ナイキスト周波数 4）で標本化すると，周波数 7 の正弦波は折り返しにより周波数 1 の標本列と等価になる．

1. 十分に大きいサンプリング周波数で標本化する．
2. ナイキスト周波数より少し小さいカットオフ周波数を持つ低域通過フィルタを用いてナイキスト周波数より大きい周波数成分を十分に減衰させる．

デジタル化されたモノラル音響信号[*2]は，標本を時系列に並べた 1 次元の配列で表現できる．多くの音響信号フォーマットでは各標本を符号なし 8 ビットや符号あり 16 ビットなどの整数で表現するが，本演習では各標本の値は −1 から 1 の間の実数で表現されているものとする．例えば各標本が符号あり 16 ビットの整数で表現されている音響信号ファイルを読み込んだ場合，各標本の値を 1/32768 倍することで −1 から 1 の範囲に変換する．

コンピュータで扱う音響信号の形式は多岐にわたる．本演習では，最も広く用いられている形式の一つである WAVE フォーマットの音響信号を用いる．WAVE フォーマットは主に，チャネル数・サンプリング周波数・量子化ビット数などを含む fmt チャンクと音響信号を含む data チャンクからなる．詳細は `https://ccrma.stanford.edu/courses/422/projects/WaveFormat/` などを参照すること．

2.2　音の 3 属性

定常な音の特徴は，以下の 3 属性に分類できる．

1. 音量 loudness
2. 音高 pitch
3. 音色 timbre

[*2] ステレオ（2 チャネル）や 5.1ch サラウンド（6 チャネル）の場合は チャネル数 × 標本数 の 2 次元配列となる．

2.3 フーリエ変換

フーリエ変換 *Fourier transform* は，複数正弦波の加算からなる信号をその構成要素である個々の正弦波へと分解する変換であり，以下で定義される[*3].

$$X(f) = \mathcal{F}[x(t)] \equiv \int_{-\infty}^{\infty} x(t)\, e^{-2\pi i f t}\, dt \tag{2}$$

$X(f)$ から元の信号 $x(t)$ を復元することができ，この変換**逆フーリエ変換** *inverse Fourier transform* と呼ぶ.

$$x(t) = \mathcal{F}^{-1}[X(f)] \equiv \int_{-\infty}^{\infty} X(f)\, e^{2\pi i f t}\, df \tag{3}$$

フーリエ変換（および逆フーリエ変換）がもつ重要な性質のいくつかを紹介する.

線型性 a, b は適当な定数.

$$\mathcal{F}[ax(t) + by(t)] = a\mathcal{F}[x(t)] + b\mathcal{F}[y(t)] \tag{4}$$

畳み込みと乗算 関数の**畳み込み** *convolution* に対するフーリエ変換は，それぞれの関数のフーリエ変換の乗算と等価.

$$\mathcal{F}[(x * y)(t)] = \mathcal{F}[x(t)]\mathcal{F}[y(t)] \tag{5}$$

畳み込み $(x * y)(t)$ は以下で定義される.

$$(x * y)(t) = \int_{-\infty}^{\infty} x(\tau) y(t - \tau)\, d\tau \tag{6}$$

これらの性質の多くは離散フーリエ変換でも同様に成り立つ.

2.3.1 離散フーリエ変換

デジタル信号の長さは必ず有限なので，上記の $(-\infty, \infty)$ の範囲でフーリエ変換を行うことはできない．長さ N のデジタル信号に対する**離散フーリエ変換** *discrete Fourier transform (DFT)* は以下で定義される.

$$X_f = \mathcal{F}[x_0, \ldots, x_{N-1}] \equiv \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} x_t e^{-\frac{2\pi}{n} i f t} \tag{7}$$

フーリエ変換と同様に，離散フーリエ変換にも逆変換が定義される．これを**逆離散フーリエ変換** *(inverse DFT; IDFT)* と呼ぶ.

$$x_t = \mathcal{F}^{-1}[X_0, \ldots, X_{N-1}] \equiv \frac{1}{\sqrt{N}} \sum_{f=0}^{N-1} X_f e^{\frac{2\pi}{n} i f t} \tag{8}$$

離散フーリエ変換そのものは長さ N の複素ベクトルから長さ N の複素ベクトルへの写像であるが，実用上は長さ N の実ベクトルを入力とすることが多い．このときの離散フーリエ変換の結果は，ベクトルの先頭を除いて複素共役な偶対称 $(w_1 = \overline{w_{N-1}}, w_2 = \overline{w_{N-2}}, \ldots)$ となる[*4]ため，得られたベクトルの前半部分 $(w_0, \ldots, w_{N/2})$ のみを用いればよい．後半部分 $(w_{N/2+1}, \ldots, w_{N-1})$ は冗長なので捨ててしまっても差し支えない.

[*3] 定義にはいくつかの流儀がある.

[*4] どの成分がベクトルのどの位置に配置されるかは離散フーリエ変換の実装に依存する.

離散フーリエ変換を定義通りに計算すると，計算量は長さ N のデータに対して $O(N^2)$ となる．**高速フーリエ変換** *fast Fourier transform (FFT)* は，ある条件の下で離散フーリエ変換を $O(N \log N)$ の計算量で行うアルゴリズムである．よく知られた高速フーリエ変換アルゴリズムは N が 2 の累乗のときに適用可能である．これは，データを半分に分割しながら再帰的に変換を行うためである．いくつかの高速フーリエ変換のライブラリでは，N がある程度小さい素数の積に素因数分解可能である場合でも高速に計算できるようにアルゴリズムを拡張している．

2.3.2 短時間フーリエ変換

フーリエ変換には局所性がない．例えば「あいうえお」という音声をフーリエ変換すると「あいうえお」全体の周波数分析がなされるため，『どの周波数成分が「あ」に対応するか』といったことは分からない．**短時間フーリエ変換** *short-time*[*5] *Fourier transform* は，適当に定めた**窓関数**をずらしながら信号を切り出し，切り出された信号のそれぞれをフーリエ変換することでスペクトルの時間変化を求める手法である．

$$\begin{aligned} X(k, f) &= \mathcal{F}(x(t) * w(t+k)) \\ &= \int_{-\infty}^{\infty} x(t)\, w(t+k)\, e^{-2\pi i f t}\, dt \end{aligned} \tag{9}$$

$$X_{k,f} = \mathcal{F}[x_{k\tau} w_0, x_{k\tau+1} w_1, x_{k\tau+2} w_2, \ldots, x_{k\tau+N-1} w_{N-1}] \tag{10}$$

短時間フーリエ変換の性質は以下の 3 要素で決まる．

1. 窓関数の長さ N：時間分解能と周波数分解能のバランスを決める．長い窓関数を使うと周波数分解能が向上し時間分解能が低下する．短い窓関数を使うと時間分解能が向上し周波数分解能が低下する．これらはトレードオフの関係にある．音声信号や音楽音響信号の分析では十ミリ秒から数百ミリ秒の間とすることが多い．
2. 窓関数の種類 $w_0, \ldots, w_{N-1}$：信号を切り出し，両端を 0 に近づけることで端点の不連続さをなくす．様々な窓関数が提案されており，目的によって使い分けられる．代表的な窓関数を以下に挙げる．

   ハン窓 hann window　ハミング窓との類似性から，ハニング窓 hanning window とも呼ばれる．

   $$w_n = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right) \tag{11}$$

   ハミング窓 hamming window

   $$w_n = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right) \tag{12}$$

   ガウス窓 Gaussian window

   $$w_n = \exp\left(-\frac{(n - N/2)^2}{2\sigma^2 N^2}\right) \tag{13}$$

   ブラックマン窓 Blackman window

   $$w_n = 0.42 - 0.5 \cos\left(\frac{2\pi n}{N}\right) + 0.08 \cos\left(\frac{4\pi n}{N}\right) \tag{14}$$

   三角窓 triangular window

   $$w_n = 1 - 2\left|\frac{2n - N}{2N}\right| \tag{15}$$

[*5] short-term とする場合もある．

矩形窓 rectangular window

$$w_n = 1 \tag{16}$$

3. 窓関数をずらす長さ τ（シフト長，ホップサイズ）：10ms から 100ms の間の適当な長さ，もしくは窓関数の長さの半分，4 分の 1，8 分の 1 などとすることが多い．

短時間フーリエ変換に限らず，**フレーム（窓関数を用いて切り出された信号）**に対する様々な信号処理は音響信号処理では頻繁に持ちいられ，これをフレーム分析と呼ぶ．例えば大語彙連続音声認識エンジン Julius[*6]では，長さ 25ms の窓関数を 10ms ずつシフトさせながら Mel-frequency Cepstrum Coefficients (MFCC) と呼ばれる音響特徴量を抽出しそれを用いて音響モデルの構築を行う．

3 音声の分析

本節では，音響信号処理の基礎技術の習得を目的として，音声を対象とした信号処理を扱う．

3.1 音声の録音と再生

音声を録音し WAVE ファイルへ保存する．

```
1  import pyaudio
2  import wave
3
4  record_duration = 2 # seconds
5  output_filename = 'out.wav'
6
7  channels = 1 # monoral
8  sampling_rate = 48000 # Hz
9  buffer_size = 4096
10 format = pyaudio.paInt16 # 16-bit integer
11 sample_width = 2 # 16bits = 2bytes
12
13 p = pyaudio.PyAudio()
14 stream = p.open(
15     format = pyaudio.paInt16,
16     channels = channels,
17     rate = sampling_rate,
18     frames_per_buffer = buffer_size,
19     input = True
20 )
21 wav_file = wave.open(output_filename, 'wb')
22 wav_file.setnchannels(channels)
23 wav_file.setsampwidth(sample_width)
24 wav_file.setframerate(sampling_rate)
25
26 remain = sampling_rate * record_duration
27 while remain > 0:
28     buf = stream.read(min(buffer_size, remain))
29     wav_file.writeframes(''.join(buf))
30     remain -= buffer_size
31
```

[*6] http://julius.sourceforge.jp/

```
stream.close()
p.terminate()
wav_file.close()
```

WAVE ファイルを読み込み音声を再生する．音声を耳で聞くことは音響信号分析のいちばんの基礎である．

```
import pyaudio
import wave

input_filename = 'in.wav'
buffer_size = 4096

wav_file = wave.open(input_filename, 'rb')
p = pyaudio.PyAudio()
stream = p.open(
    format = p.get_format_from_width(wav_file.getsampwidth()),
    channels = wav_file.getnchannels(),
    rate = wav_file.getframerate(),
    output = True
)

remain = wav_file.getnframes()
while remain > 0:
    buf = wav_file.readframes(min(buffer_size, remain))
    stream.write(buf)
    remain -= buffer_size

stream.close()
p.terminate()
wav_file.close()
```

■課題 1　2 秒以内で「あいうえお」と発声し，これをサンプリング周波数 48kHz，量子化ビット数 16，モノラルで録音せよ．「あ・い・う・え・お」と一音ずつ区切った発音と「あーいーうーえーおー」と区切らない連続的な発音の 2 種類を録音すること．保存した音声は必ず聴取し，音量は必要十分か，先頭や末尾が途切れていないか，他者の音声やノイズが混入していないか，などを確認する．

3.2　波形の図示

視覚とは異なり，聴覚には一覧性がない．言い換えると，ある程度の長さの音声をその長さよりも短い時間で把握することや複数の音声を同時に聴いて理解することはできない（もしくは極めて困難である）．したがって，音声を図示して視覚的に確認することは極めて重要である．

WAVE ファイルから音声を読み込み，波形を図示する．図ウインドウ上部のメニューから図を各種画像形式で保存できるので，適切な図をレポートに配置すること．

ソースコード 1　plot_waveform.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import scipy.io.wavfile
import numpy
import pylab
```

```
8
9  def plot_waveform(waveform, sampling_rate):
10     sampling_interval = 1.0 / sampling_rate
11     times = numpy.arange(len(waveform)) * sampling_interval
12     pylab.plot(times, waveform) # pair of two x- and y-coordinate lists/arrays
13     pylab.title('Waveform')
14     pylab.xlabel('Time [sec]')
15     pylab.ylabel('Amplitude')
16     pylab.xlim([0, len(waveform) * sampling_interval])
17     pylab.ylim([-1, 1])
18     pylab.show()
19
20 if __name__ == '__main__': # main function of Python programs
21     argv = sys.argv
22     if len(argv) == 1:
23         print 'no input files.'
24         exit
25
26     filename = argv[1]
27     sampling_rate, waveform = scipy.io.wavfile.read(filename)
28     waveform = waveform / 32768.0 # assume 16-bit integer
29     plot_waveform(waveform, sampling_rate)
```

■課題 2　2 種類の「あいうえお」の波形を図示し，「あ」「い」「う」「え」「お」それぞれの区間をグラフ上で示せ．横軸の単位はサンプルではなく，秒（もしくはミリ秒）単位とせよ．

3.3　スペクトル

波形は音声の情報の一部を図示するが，多くの情報は波形からは把握できない．例えば「あ」「い」「う」「え」「お」を独立に録音した信号の波形からそれぞれがどの音に対応するかを判断することはほぼ不可能である．

そこで，音声信号をフーリエ変換し，そのスペクトルによって音声の特徴を捉えることが一般的である．信号は様々な周波数の正弦波の重ね合わせで表現できるため，ある信号にどのような周波数の正弦波が含まれているかをフーリエ変換で得る．信号をフーリエ変換すると，その結果（フーリエ変換係数）は複素数となる．音声の分析においては，重ね合わされている正弦波の周波数と振幅が重要であり，正弦波の位相はさほど重要ではない[*7]ため，スペクトルを図示する際にはフーリエ変換係数の絶対値をとって位相を捨てることが多い．さらに，人間は振幅の対数に比例した音の大きさを知覚するため，スペクトルの図示においても同様に振幅の対数を図示することが多い．

Python では `numpy.fft` モジュール中の各種関数により離散フーリエ変換を行える．これらの関数は引数配列の長さをチェックし，高速フーリエ変換が適用可能な場合（配列長が 2 のべき乗）とそうでない場合とでアルゴリズムを自動で切り替えるように構成されている．

■課題 3　2 種類の「あいうえお」をそれぞれフーリエ変換し，その振幅スペクトルを図示せよ．横軸が周波数（0Hz からナイキスト周波数まで），縦軸が振幅の対数となるようにすること．

■課題 4　2 種類の「あいうえお」の「あ」「い」「う」「え」「お」それぞれの区間をフーリエ変換し，その振幅スペクトルを図示せよ．

[*7] 逆フーリエ変換によって信号を復元する際には位相が必要である．

■**課題** 5† 離散フーリエ変換を実装せよ．実装が正しいかどうかは，適当な信号を `numpy.fft` および自作フーリエ変換で変換し，それぞれの結果が一致することで確認すること．

■**課題** 6† 高速フーリエ変換を実装せよ．

3.4 短時間フーリエ変換

音声の局所的な特徴を捉えるためには短時間フーリエ変換を用いる．短時間フーリエ変換で得られる，時間変化するスペクトルの画像表現を**スペクトログラム** *spectrogram* と呼ぶ．一般に，横軸が時間，縦軸が周波数，色相や濃淡が各時刻における各周波数成分の強度を表す．Python では `matplotlib.pyplot` モジュールの `specgram()` 関数で短時間フーリエ変換を行える．

ソースコード 2 plot_spectrogram.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import wave
import scipy.io.wavfile
import numpy
import scipy
import pylab
import matplotlib

def plot_spectrogram(waveform, sampling_rate):
    window_duration = 40.0 * 1.0e-3 # seconds
    window_shift = 5.0 * 1.0e-3 # seconds
    window_size = int(window_duration * sampling_rate) # frames
    window_overlap = int((window_duration - window_shift) * sampling_rate)
    window = scipy.hanning(window_size)

    sp, freqs, times, ax = matplotlib.pyplot.specgram(
        waveform,
        NFFT = window_size,
        Fs = sampling_rate,
        window = window,
        noverlap = window_overlap
    )
    matplotlib.pyplot.xlim([0, float(len(waveform)) / sampling_rate])
    matplotlib.pyplot.ylim([0, sampling_rate / 2])
    matplotlib.pyplot.show()


if __name__ == '__main__':
    argv = sys.argv
    if len(argv) == 1:
        print 'no input files.'
        exit

    filename = argv[1]
    sampling_rate, waveform = scipy.io.wavfile.read(filename)
    waveform = waveform / 32768.0
    plot_spectrogram(waveform, sampling_rate)
```

■課題 7　2 種類の「あいうえお」のスペクトログラムを図示せよ．横軸が時間，縦軸が周波数，各ピクセルの色（もしくは濃淡）が振幅を表すようにすること．用いた窓関数の種類，長さ，シフト長も示すこと．それぞれにどのような特徴があるか述べよ．

■課題 8†　短時間フーリエ変換を実装せよ．

■課題 9†　逆短時間フーリエ変換を実装せよ．

3.5　音量の抽出

音量 *loudness* は「大きい」「小さい」と形容される音の属性であり，音の 3 属性ではもっとも抽出が容易な属性である．正弦波においては，音量はその振幅の対数におおむね比例する[*8]．音声波形などの多数の正弦波の重ね合わせでは，各周波数のパワー（振幅の 2 乗）の平均[*9]の平方根が音量となる．さらに，パーセバルの定理 Parseval's theorem により，以下の両辺は等価である．

$$\sqrt{\frac{1}{N}\sum_{f=0}^{N-1} X_f^2} = \sqrt{\frac{1}{N}\sum_{t=0}^{N-1} x_t^2} \tag{17}$$

これで計算される音量を RMS (root-mean-square) と呼ぶ．

音量の知覚は振幅の対数に比例するため，これの図示においても RMS の対数を図示するとより直感的である．音量は一般的にデシベル dB という単位で表される．ある音 A がある音 B よりも 20dB 大きな音量をもつとき，A は B よりも 10 倍大きい RMS をもつ．すなわち，1dB の違いは $\sqrt[20]{10} \simeq 1.122$ 倍の違いとなる．デシベル単位の音量は以下で計算できる．

$$\mathrm{Vol}_{\mathrm{dB}} = 20 \log_{10} \mathrm{RMS} \tag{18}$$

■課題 10　2 種類の「あいうえお」を短時間フレームに分割し，各フレームの音量を図示せよ．横軸を時間，縦軸を音量（単位は dB）とすること．

3.6　音高の抽出

音高 *pitch* は「高い」「低い」と形容される音の属性である．身の回りにある様々な音は，明確な音の高さを持つ音である楽音 pitched sound とそれ以外の噪音 unpitched sound に大きく分けられる．ここでは，楽音に対して音高を抽出する手法を説明する．

3.6.1　調波構造

1. 数秒間の様々な周波数の純音（単一の正弦波からなる音）を生成・聴取する．周波数が小さいと「低い音」が，周波数が大きいと「高い音」が聞こえるはずである．

[*8] 人間の聴覚においては，物理的に同じ振幅の正弦波でも周波数によって異なる音量が知覚される．一般的には 3kHz 付近がもっとも敏感である．また，可聴域外（20Hz 以下，20kHz 以上）は音として感じることができず，可聴域の上限は加齢などによって低下する．詳しくは等ラウドネス曲線などで調査すること．

[*9] 平均をとるのは信号長 N で正規化するためである．

2. 異なる周波数をもつ 2 つの正弦波を生成し，その混合音（正弦波の和）を生成・聴取する．一方の正弦波の周波数 ω_1 を 440Hz に固定し，他方の正弦波の周波数 ω_2 を 220Hz から 880Hz まで変化させる．ω_1 と ω_2 の関係に応じて，
   (a) ω_1 の正弦波と ω_2 の正弦波が独立して聞こえる
   (b) ω_1 の正弦波と ω_2 の正弦波が調和して聞こえる
   ということが起こる．
3. 正弦波の数を 3, 4, ... と増やす．周波数がどのような関係にあるときにそれぞれの正弦波が調和して聞こえるだろうか．

ソースコード 3　generate_sinusoid.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import math
import numpy
import scipy.io.wavfile

def generate_sinusoid(sampling_rate, frequency, duration):
    sampling_interval = 1.0 / sampling_rate
    t = numpy.arange(sampling_rate * duration) * sampling_interval
    waveform = numpy.sin(2.0 * math.pi * frequency * t)
    return waveform

if __name__ == '__main__':
    argv = sys.argv
    if len(argv) == 1:
        print 'no output file.'
        exit

    sampling_rate = 48000.0 # Hz
    frequency = 440.0 # Hz
    duration = 2.0 # seconds
    waveform = generate_sinusoid(sampling_rate, frequency, duration)
    waveform = waveform * 0.9 # avoid clipping noises
    waveform = (waveform * 32768.0).astype('int16')

    filename = argv[1]
    scipy.io.wavfile.write(filename, int(sampling_rate), waveform)
```

上記の実験を通じて，**正弦波の周波数が整数倍の関係にあるときにそれぞれの正弦波が調和して聞こえる**ということが分かる．身の回りにあるほぼ全ての楽音は，等しくこのような特徴を持っている．例えば，ピアノの中央のラ，およびそれと同じ高さを持つ音は，$\omega_1 = 440$Hz, $\omega_2 = 880$Hz, $\omega_3 = 1320$Hz, $\omega_4 = 1760$Hz, $\omega_5 = 2200$Hz, ... という正弦波の和からなり[*10]，440Hz の純音と同じ高さが知覚される．このような音の構造を**調波構造** *harmonic structure* と呼ぶ．調波構造のうち，もっとも低い周波数（すなわち，知覚される音の高さ[*11]）を**基本周波数** *fundamental frequency*[*12] と呼び，その正弦波を基音と呼ぶ．また，基本周波数の

[*10] ピアノに限っては，周波数の比は厳密な整数ではなく，高次倍音の周波数は若干大きくなる．弦に強い張力がかかり堅くなることで生じる現象で，インハーモニシティと呼ばれる．
[*11] 聴覚的には若干の例外がある．ミッシングファンダメンタル現象などが知られている．
[*12] 基本周波数のことを F0 と呼ぶことがある．

整数倍の周波数をもつ正弦波を倍音と呼ぶ．特に，2 倍，3 倍，4 倍，... の周波数を持つ倍音のそれぞれを指して第 2 倍音，第 3 倍音，第 4 倍音，... と呼ぶ[*13]．調波構造を持つ音の振幅スペクトルには，周波数方向に等間隔な強いピークが確認できる．スペクトログラムでは周波数方向の縞模様が現れる．もっとも低い周波数のピークが基本周波数である．

ほぼ全ての楽音は調波構造を持ち，その音の高さは基本周波数に一致する．楽音から音高を抽出することは調波構造の基本周波数を推定することにほかならない．

■課題 11　2 種類の「あいうえお」のスペクトログラムに対して，その基本周波数をスペクトログラム上で図示せよ．必要に応じてスペクトログラムを周波数方向に拡大すること．

3.6.2　自己相関

自己相関 *autocorrelation* は，ある信号とその信号自身を時間方向にシフトした信号とがどの程度類似しているかを表す尺度である．

$$\forall \tau = 0, \ldots, N-1 : \mathrm{AC}_\tau = \sum_{t=0}^{N-\tau-1} x_t x_{t-\tau} \tag{19}$$

周波数 ω の正弦波に対してあらゆる τ のもとでの自己相関を計算すると，$\tau = 0$ のときに自己相関は最大のピークをもち，$\tau = 1/\omega$ のときに 2 番目に大きいピークをもつ．調波構造を持つ音の多くは，基音が倍音よりもある程度大きい振幅を持つものが多いため，自己相関においても基音のそれが倍音よりも支配的になり，やはり 2 番目に大きいピークが基本周波数の逆数に対応する．

ソースコード 4　numpy.correlate() 関数による自己相関の計算と基本周波数推定

```
import numpy

# 配列 a の index 順目の要素がピーク（両隣よりも大きい）であれば True を返す
def is_peak(a, index):
    ...

# 自己相関が格納された，長さが len(waveform) * 2 - 1 の対称な配列を得る
autocorr = numpy.correlate(waveform, waveform, 'full')
# 不要な前半を捨てる
autocorr = autocorr[len(autocorr) / 2 : ]
# ピークのインデックスを抽出する
peakindices = [i for i in range(len(autocorr)) if is_peak(autocorr, i)]
# インデックス 0 がピークに含まれていれば捨てる
peakindices = [i for i in peakindices if i != 0]
# 自己相関が最大となるインデックスを得る
max_peak_index = max(peakindices, key=lambda index: autocorr[index])
# インデックスに対応する周波数を得る
f0 = ...
```

■課題 12　2 種類の「あいうえお」を短時間フレームに分割し，各フレームでの基本周波数を自己相関を用いて推定せよ．

■課題 13†　自己相関はパワースペクトル（振幅スペクトルの 2 乗）の逆フーリエ変換で計算できる．これを証明せよ．また，このように自己相関を計算することの利点について述べよ．

[*13] 基音のことを第 1 倍音とも呼ぶ．

■**課題 14**† ミッシングファンダメンタル現象を起こす楽音に対する基本周波数推定法について議論せよ．

3.7 音色の抽出

音色[*14] *timbre* について単一の形容詞対を当てはめることは難しい．多くの場合，「明るい」「暗い」，「柔らかい」「堅い」，「乾いた」「しっとりとした」などの様々な形容詞対の集合として表現される．辞書的には「聴覚に関する音の属性の一つで，物理的に異なる二つの音が，たとえ同じ音の大きさ及び高さであっても異なった感じに聞こえるとき，その相違に対応する属性 (JIS)」と定義されている．例えばピアノとバイオリンが全く同じ音量と音高で演奏されたとしても，我々はその違いを聞き分けることができる．音色とはこのような違いに対応する性質のことである．より具体的には，スペクトルの形状や分布，音高や音圧の時間変化，倍音の相対強度などに関係する．ピアノとバイオリンを聞き分けられるのと同様に，我々は同じ音量と音高で発声された「あ」「い」「う」「え」「お」を聞き分けることができる．本節ではこのような，日本語母音の音色抽出について扱う．

3.7.1 ケプストラム

ケプストラム *cepstrum*[*15] は，対数振幅スペクトルのフーリエ変換で定義される．

$$\mathcal{F}[\log|\mathcal{F}[x_0, x_1, \ldots, x_{N-1}]|] \tag{20}$$

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy

def cepstrum(amplitude_spectrum):
    log_spectrum = numpy.log(amplitude_spectrum)
    cepstrum = numpy.fft.fft(log_spectrum)
    return cepstrum
```

人間の音声生成過程は，**ソース・フィルタモデル**でよく表現できることが知られている．ソースは音源のことであり，人間では**声帯** *vocal cords* に対応する．声帯は近似的に，ある周期（基本周波数の逆数）でパルス列を生成しているとみなすことができる．フィルタは，一般には信号中の特定の周波数成分を強調・抑圧する線型時不変で安定なシステムであり，人間では喉頭，咽頭，口腔，鼻腔をひとまとめにした**声道** *vocal tract* に対応する．ソース・フィルタモデルでは，声帯は一定周期のパルス列を生成するだけなのでそこに個人性や音韻性は含まれず，音高のみに影響する．一方の声道は，個人によって長さや太さが異なっており，さらに発声する母音によって口腔の形状が変化する．すなわち，音声からソースの成分を除去して，フィルタの成分のみを抽出することができれば各母音の音韻性の違いを取り出すことができる．

スペクトルからソース成分を除去したものが**スペクトル包絡** *spectral envelope* である．音源信号は等間隔のパルス列であり，スペクトルでは全ての調波ピークの大きさが等しい調波構造となる．この成分を取り除いた，スペクトルのピークをなめらかに結ぶ曲線がスペクトル包絡となる．

[*14] 一般には「ねいろ」と読むが，学術的には「おんしょく」と読む．音量を「おんりょう」，音高を「おんこう」と読むこととの一貫性を保つためである．

[*15] **spec**trum の先頭 4 文字の逆転．

ここで，振幅スペクトルを信号として見てみると，低い周波数の成分（＝スペクトル包絡）と高い周波数の成分（＝調波構造）の積からなっていると近似的にみなせる．そこから低い周波数の成分のみを取り出せばよいので，

1. 振幅スペクトルの対数をとり，積を和に変換する．
2. 対数振幅スペクトルをフーリエ変換する．フーリエ変換では線形性が成り立つので，複数の信号の和のフーリエ変換はそれぞれの信号のフーリエ変換の和に等しい．
3. フーリエ変換の結果のうち，低い周波数の成分のみを取り出す．
4. 取り出した成分のみを逆フーリエ変換する．

という手順を踏むことにより，スペクトル包絡を抽出することができる．これは低次のケプストラム係数のみを抽出することと等価である．すなわち，**低次のケプストラム係数には音韻性や個人性のみが含まれる**．

■課題 15　2 種類の「あいうえお」に対して 1 から 13 次までのケプストラム係数を抽出し，そこから得られるスペクトル包絡を元の対数振幅スペクトルに重ねて示せ．

■課題 16　2 種類の「あいうえお」の「あ」「い」「う」「え」「お」それぞれの区間の対数振幅スペクトルとスペクトル包絡を重ねて示せ．

3.7.2　ゼロ交差数

有声音と無声音の識別という課題を考える．**有声音** *voiced sound* とは声帯振動を伴う音声であり，**無声音** *unvoiced sound* とはその逆の声帯振動を伴わない音声である．典型的には /a/, /i/, /u/, /e/, /o/ などの母音，/g/, /d/, /b/ などの子音，/y/, /w/ などの半母音が有声音，/k/, /s/, /t/ などの子音，いわゆる「ひそひそ声」が無声音に対応する．原理的に有声音にのみ基本周波数が存在するため，無声音に対する基本周波数推定は無意味な結果となる．有声音と無声音の両方を含む一般的な音声に対する基本周波数推定においては有声音と無声音を識別し，有声音に対してのみ推定することが必要である．

信号の周期性や白色性を計る指標として**ゼロ交差数** *zero crossing (rate)* が用いられる．ゼロ交差数は信号波形が単位時間あたりに振幅 0 の軸を交差する回数で定義される．調波構造を持つ楽音に対しては，特にその基音の振幅がその他の倍音よりもある程度以上に大きい場合は，その基本周波数の 2 倍[*16]に近いゼロ交差数が得られる．一方，高周波成分がある程度の振幅をもつホワイトノイズなどの雑音に対しては，ゼロ交差数は相応に大きい値をとる．

ゼロ交差数を定義どおりに計算するプログラムは以下である．

```
zc = 0
for i in range(len(waveform) - 1):
    if ((waveform[i] > 0.0 and waveform[i+1] < 0.0) or
        (waveform[i] < 0.0 and waveform[i+1] > 0.0)):
        zc += 1
```

以下でも計算できる（なぜこれでゼロ交差数が計算できるのか考えてみるとよい）．

```
sum([1 if x < 0.0 else 0 in x for waveform[1:] * waveform[:-1]])
```

[*16] 正弦波は 1 周期にゼロ軸を上向きと下向きに 1 回ずつ交差する．

■課題 17　母音と子音の両方を含む適当な文を録音し，そのスペクトログラムと基本周波数を並べて図示せよ．非発音区間および無声区間に対しては基本周波数の表示を抑制すること．あらかじめ基本周波数を自己相関などで推定しておき，各フレームに対してゼロ交差数による有声・無声の判定を行い，無声部の基本周波数を 0 とする，もしくは非表示とする．

3.8　音声認識

ここでは，音響特徴量としてケプストラム係数（の実部）を，確率的生成モデルとして正規分布を用いた音声認識システムを構築する．認識対象は日本語音声の 5 母音「あ」「い」「う」「え」「お」とする．

多次元正規分布 $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ の確率密度関数 $f_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$ は以下で定義される．

$$f_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}{2}\right) \tag{21}$$

ここでは簡単のため，共分散行列 Σ は対角行列

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_D^2 \end{bmatrix} \tag{22}$$

であるとする．これを用いると，多次元正規分布の確率密度関数は

$$f_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_d} \exp\left(-\sum_{d=1}^{D} \frac{(x_d - \mu_d)^2}{2\sigma_d^2}\right) \tag{23}$$

と書き直せる．

まず，認識対象とする各音素に対して，音響特徴量の確率的生成モデルを学習しよう．あらかじめ音声を「あ」「い」「う」「え」「お」の各区間に分割しておき，各区間から短時間フレームを切り出し，各フレームに対してケプストラム（の実部）を計算する．ここまでで，ケプストラムの集合が母音ごとに得られることになる．

各母音のケプストラム集合から，生成モデルを学習する．モデルは正規分布であることを仮定しているので，実際には正規分布のパラメータを最尤推定すればよい．ある母音のケプストラムの集合を $X = \boldsymbol{x}_1, \ldots, \boldsymbol{X}_N$ とすると，ケプストラム集合に対する対数尤度 L は以下で表される．

$$\begin{aligned} L = \log p(X|\boldsymbol{\mu}, \Sigma) &= \log \prod_{n=1}^{N} f_{\mathcal{N}}(\boldsymbol{x}_n; \boldsymbol{\mu}, \Sigma) \\ &= -\sum_{n=1}^{N} \left(\frac{D}{2} \log 2\pi + \sum_{d=1}^{D} \log \sigma_d + \sum_{d=1}^{D} \frac{(x_{n,d} - \mu_d)^2}{2\sigma_d^2} \right) \\ &\propto -\sum_{n=1}^{N} \sum_{d=1}^{D} \left(\log \sigma_d + \frac{(x_{n,d} - \mu_d)^2}{2\sigma_d^2} \right) \end{aligned} \tag{24}$$

この対数尤度を最大化するパラメータ μ と Σ を求める．対数尤度は上に凸な関数なので，導関数が 0 になるとき対数尤度は最大となる．つまり，最尤パラメータ $\hat{\boldsymbol{\mu}}$ と $\hat{\Sigma}$ を求めるには，それぞれの方程式

$$\forall d = 1, \ldots, D : \frac{\partial L}{\partial \hat{\mu_d}} = 0 \tag{25}$$

$$\forall d = 1, \ldots, D : \frac{\partial L}{\partial \hat{\sigma_d}} = 0 \tag{26}$$

を解けばよい．これらを解くと，

$$\forall d = 1, \ldots, D : \hat{\mu_d} = \frac{1}{N} \sum_{n=1}^{N} x_{n,d} \tag{27}$$

$$\forall d = 1, \ldots, D : \hat{\sigma_d} = \frac{1}{N} \sum_{n=1}^{N} (x_{n,d} - \mu_d)^2 \tag{28}$$

となる．

モデルの学習が完了すると，いよいよ音素の認識である．認識対象の音声では，どの区間がどの音素に対応するのかは未知である．音声から短時間フレームを切り出し，各フレームからケプストラムを計算し，各フレームに対して，学習されたモデルを用いて各音素の音素らしさを計算する．音素らしさを表す尺度にはモデルの対数尤度を用いる．対数尤度が最大となる音素をそのフレームの音素（認識結果）として出力する．

■課題 18　2 種類の「あいうえお」の一方を用いてモデルを学習し，他方を用いて母音を認識せよ．「あ」=0, 「い」=1, 「う」=2, 「え」=3, 「お」=4 として，認識された音素の推移をスペクトログラムに並べて示せ．

4　音楽音響信号の分析

本節では，音響信号処理の発展的な技術の習得を目的として，音楽音響信号処理技術について扱う．音声信号と音楽音響信号の最も大きな違いの一つが複数音の混合である．音声信号に対する信号処理技術の多くは対象が単独音であることを前提にしており，このような技術を混合音である音楽音響信号にそのまま適用しても望ましい結果が得られない．混合音を扱うためには，対象が混合音であることを前提とした手法や，**音源分離** *sound source separation* で混合音を個々の音源へと分離することが必要不可欠である．

4.1　楽器音の音高

我々の身の回りには様々な種類の楽曲があり，中でも西洋音楽に端を発するものは特に幅広く普及している．西洋音楽では，1 オクターブを 12 の音高に分割[*17]した音高を楽譜上での音高の最小単位として扱う．さらに，各音高を C, C#, D, D#, E, F, F#, G, G#, A, A#, B のピッチクラスに分類する．同じピッチクラスに属する音高は周波数が 2^n 倍の関係にあり，音楽的に同じ役割を与えることが多い．ピッチクラス内の特定の音高を参照するには，例えば A4 や F#1 のように，ピッチクラスに続けてオクターブ番号を付与する．オクターブ番号が小さいほど低い音高を，大きいほど高い音高を表す．

コンピュータ音楽の分野では *Musical Instrument Digital Interface (MIDI)* と呼ばれる規格に沿った電子楽器や楽譜フォーマットが幅広く用いられている．MIDI ではそれぞれの音高に 0 から 127 の**ノートナンバー** *note number* を与え，この番号を用いて音高を表現する．オクターブ番号と同様に，小さいノートナンバーは低い音高を，大きいノートナンバーは高い音高を表す．ノートナンバー，音名，基本周波数などの関係を図 4 に示す．基準となるのがノートナンバー 69 であり，以下のような音高である．

- 周波数 440Hz

[*17] ポピュラー音楽では隣り合う半音の周波数比が等しく $\sqrt[12]{2}$ となる平均律が広く用いられる．周波数比が均等でない音律には純正律やピタゴラス音律などがある．

- 音名 A4[*18]
- ピアノの中央のラ
- テレビ放送[*19]などの時報における予告音[*20]

ノートナンバー n から周波数 f を以下で得る．

$$f = 440 \times 2^{(n-69)/12} \tag{29}$$

逆変換は以下で与えられる．

$$n = 12 \log_2 \frac{f}{440} + 69 \tag{30}$$

Python では以下でこれらの変換を得られる．

```
import math

def nn2hz(notenum):
    return 440.0 * (2.0 ** (notenum - 69) / 12.0)

def hz2nn(frequency):
    int(round(12.0 * (math.log(frequency / 440.0) / math.log(2.0)))) + 69
```

4.2 和音の認識

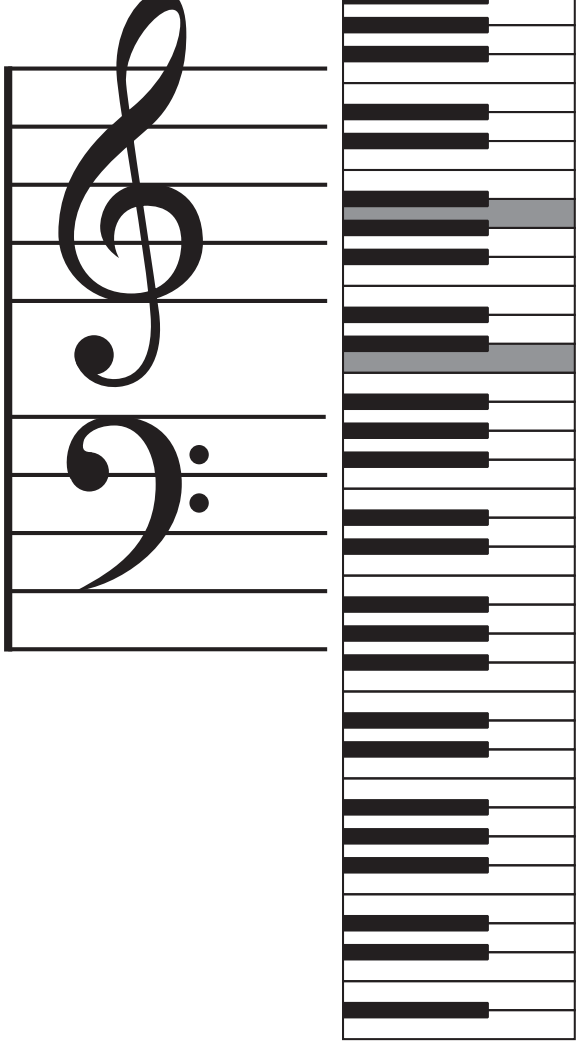
メロディ・ハーモニー・リズムは音楽の 3 要素と呼ばれ，楽曲の特徴を大きく左右する．ここではハーモニーの主要な構成要素である**和音** *chord* の認識に取り組む．

和音とは，異なるピッチクラスの複数の音をそれらが調和するように同時に演奏したものである．和音は一般的に C Major や G# Minor などの，[ピッチクラス]+[和音の種類] の形をもつ和音名で表される．和音は基本的には 3 つのピッチクラスからなり[*21]，基準となる根音に，その 3 度上のピッチクラスをもつ音（第 3 音）と 5 度上のピッチクラスをもつ音（第 5 音）を重ねることで構成される．根音のピッチクラスが和音のピッチクラスを，根音と第 3 音・第 5 音の音程[*22]が和音の種類を決定する．ここでは，和音の種類としてメジャーコードとマイナーコードの 2 つを扱う．メジャーコードは根音に対する第 3 音の音程が長 3 度（4 半音上），第 5 音の音程が完全 5 度（8 半音上）であり，マイナーコードは第 3 音の音程が短 3 度（3 半音上），第 5 音の音程が完全 5 度である．

音響信号に対する和音認識に広く用いられる手がかりが**クロマベクトル** *chroma vector* である．クロマベクトルは，ある混合音の中でどのピッチクラスが主要なパワーをもつかを表す．

$$\forall r \in \{\mathrm{C}, \mathrm{C\#}, \ldots, \mathrm{B}\} : \mathrm{CV}(r) = \sum_{\mathrm{nn}(f)=c} X_f \tag{31}$$

[*18] 440Hz を A3 と定める流儀もある．
[*19] 電話の時報は「シ」の音と決められている．
[*20] 「ピッ・ピッ・ピッ・ポーン」の「ピッ」の音．「ポーン」の音は本信号音と呼ぶ．
[*21] 3 つのピッチクラスからなる和音を三和音もしくはトライアド triad と呼び，西洋音楽ではこれを基本的なものとして考える．4 つ以上のピッチクラスからなる和音は，基準となるトライアドに新たなピッチクラスを加えて拡張したものと考える．
[*22] 口語的には音高と音程はほぼ同じ意味で使われることが多いが，学術的用語としてはこの 2 つは明確に区別される．音高は絶対的な周波数そのものを表すのに対して，音程は 2 つの音高の差を表す相対的な概念である．



MIDI Note Num	Note Name	Frequency [Hz]	Period [ms]
108	C8	4186.0	0.2389
107	B7	3951.1	0.2531
106	A#7	3729.3	0.2682
105	A7	3520.0	0.2841
104	G#7	3322.4	0.3010
103	G7	3136.0	0.3189
102	F#7	2960.0	0.3378
101	F7	2793.8	0.3579
100	E7	2637.0	0.3792
99	D#7	2489.0	0.4018
98	D7	2349.3	0.4257
97	C#7	2217.5	0.4510
96	C7	2093.0	0.4778
95	B6	1975.5	0.5062
94	A#6	1864.7	0.5363
93	A6	1760.0	0.5682
92	G#6	1661.2	0.6020
91	G6	1568.0	0.6378
90	F#6	1480.0	0.6757
89	F6	1396.9	0.7159
88	E6	1318.5	0.7584
87	D#6	1244.5	0.8035
86	D6	1174.7	0.8513
85	C#6	1108.7	0.9019
84	C6	1046.5	0.9556
83	B5	987.77	1.0124
82	A#5	932.33	1.0726
81	A5	880.00	1.1364
80	G#5	830.61	1.2039
79	G5	783.99	1.2755
78	F#5	739.99	1.3514
77	F5	698.46	1.4317
76	E5	659.26	1.5169
75	D#5	622.25	1.6071
74	D5	587.33	1.7026
73	C#5	554.37	1.8039
72	C5	523.25	1.9111
71	B4	493.88	2.0248
70	A#4	466.16	2.1452
69	**A4**	**440.00**	**2.2727**
68	G#4	415.30	2.4079
67	G4	392.00	2.5511
66	F#4	369.99	2.7027
65	F4	349.23	2.8635
64	E4	329.63	3.0337
63	D#4	311.13	3.2141
62	D4	293.66	3.4052
61	C#4	277.18	3.6077
60	**C4**	**261.63**	**3.8223**
59	B3	246.94	4.0495
58	A#3	233.08	4.2903
57	A3	220.00	4.5455
56	G#3	207.65	4.8157
55	G3	196.00	5.1021
54	F#3	185.00	5.4055
53	F3	174.61	5.7269
52	E3	164.81	6.0675
51	D#3	155.56	6.4282
50	D3	146.83	6.8105
49	C#3	138.59	7.2155
48	C3	130.81	7.6445
47	B2	123.47	8.0991
46	A#2	116.54	8.5807
45	A2	110.00	9.0909
44	G#2	103.83	9.6315
43	G2	97.999	10.204
42	F#2	92.499	10.811
41	F2	87.307	11.454
40	E2	82.407	12.135
39	D#2	77.782	12.856
38	D2	73.416	13.621
37	C#2	69.296	14.431
36	C2	65.406	15.289
35	B1	61.735	16.198
34	A#1	58.270	17.161
33	A1	55.000	18.182
32	G#1	51.913	19.263
31	G1	48.999	20.408
30	F#1	46.249	21.622
29	F1	43.654	22.908
28	E1	41.203	24.270
27	D#1	38.891	25.713
26	D1	36.708	27.242
25	C#1	34.648	28.862
24	C1	32.703	30.578
23	B0	30.868	32.396
22	A#0	29.135	34.323
21	A0	27.500	36.364

図 4　MIDI ノートナンバー，音名，基本周波数の関係

表 1　和音とピッチクラスの対応．括弧内は C を基準とした半音差．

和音名	根音	3 度	5 度	和音名	根音	3 度	5 度
C Major	C (0)	E (4)	G (7)	C Minor	C (0)	D# (3)	G (7)
C# Major	C# (1)	F (5)	G# (8)	C# Minor	C# (1)	E (4)	G# (8)
D Major	D (2)	F# (6)	A (9)	D Minor	D (2)	F (5)	A (9)
D# Major	D# (3)	G (7)	A# (10)	D# Minor	D# (3)	F# (6)	A# (10)
E Major	E (4)	G# (8)	B (11)	E Minor	E (4)	G (7)	B (11)
F Major	F (5)	A (9)	C (0)	F Minor	F (5)	G# (8)	C (0)
F# Major	F# (6)	A# (10)	C# (1)	F# Minor	F# (6)	A (9)	C# (1)
G Major	G (7)	B (11)	D (2)	G Minor	G (7)	A# (10)	D (2)
G# Major	G# (8)	C (0)	D# (3)	G# Minor	G# (8)	B (11)	D# (3)
A Major	A (9)	C# (1)	E (4)	A Minor	A (9)	C (0)	E (4)
A# Major	A# (10)	D (2)	F (5)	A# Minor	A# (10)	C# (1)	F (5)
B Major	B (11)	D# (3)	F# (6)	B Minor	B (11)	D (2)	F# (6)

```
1 import numpy
2
3 def chroma_vector(spectrum, frequencies):
4     # 0 = C, 1 = C#, 2 = D, ..., 11 = B
5     cv = numpy.zeros(12)
6     for s, f in zip(spectrum, frequencies):
7         nn = hz2nn(f)
8         cv[nn % 12] += s
9     return cv
```

ここでは，クロマベクトルから和音 c の和音らしさ $\mathcal{L}(c)$ を以下で定義する[*23]．

$$\forall c \in \{\mathrm{C}, \mathrm{C\#}, \ldots, \mathrm{B}\} \times \{\mathrm{Major}, \mathrm{Minor}\} : \mathcal{L}(c) = a_{\mathrm{root}}\mathrm{CV}(c_{\mathrm{root}}) + a_{\mathrm{3rd}}\mathrm{CV}(c_{\mathrm{3rd}}) + a_{\mathrm{5th}}\mathrm{CV}(c_{\mathrm{5th}}) \quad (32)$$

c_{root}, c_{3rd}, c_{5th} はそれぞれ和音 c の根音，3 度の音，5 度の音を表す．例えば $c = \mathrm{C}$ Major のとき，$c_{\mathrm{root}} = \mathrm{C}$, $c_{\mathrm{3rd}} = \mathrm{E}$, $c_{\mathrm{5th}} = \mathrm{G}$ である．a_{root}, a_{3rd}, a_{5th} は適当な定数で，例えば $(a_{\mathrm{root}}, a_{\mathrm{3rd}}, a_{\mathrm{5th}}) = (1.0, 0.5, 0.8)$ とするとよい．

■課題 19　適当な楽曲から 10 秒程度の区間を切り出し，スペクトログラムとクロマグラムを図示せよ．クロマグラムに対して，各フレームでの和音を推定しその結果を示せ．例えば C Major = 0, C# Major = 1, . . . , B Minor = 23 として，横軸を時間，縦軸を和音とした折れ線グラフが利用できる．楽曲の出典を明示すること．

[*23] 一般的には音声認識と同様に，あらかじめ用意した和音の音響信号（学習データ）から和音ごとのクロマベクトルの生成モデルを学習し，そのモデルを用いて和音らしさを定義する．

4.3 メロディの認識

メロディ *melody* は，ハーモニー・リズムと並ぶ音楽の 3 要素の 1 つとされ，楽曲の性格を決定づける主要な要素である．ポピュラー楽曲では歌声がメロディとなることが多い．混合音中のメロディを簡単な言葉で定義することは難しいが，ここでは**混合音中で最も優勢なパワーをもつ楽音の列**をメロディと定義する．

混合音中のメロディの音高を推定する手法の 1 つが *sub-harmonic summation (SHS)* である．まず最初に基本周波数の候補集合を決定する．例えばノートナンバーで $\{36, 37, 38, \ldots, 60\}$ とすれば，多くの男性歌唱者の音域をカバーできる．ビブラートなどの検出のためにはノートナンバーを 0.1 刻みにするなどの工夫が必要である．次に候補集合のそれぞれに対して，その基本周波数を持つ楽音が仮に混合音中に含まれたとした場合の全高調波成分の振幅（もしくはパワー）を総和し，これをその候補の基本周波数らしさ[*24]とする．最後に，各候補の尤度を比較し，尤度が最大となる候補をそのスペクトルにおける基本周波数とする．

■課題 20　適当な楽曲のメロディを歌って録音し，その波形，スペクトログラム，音高を示せ．音高は自己相関もしくは SHS で推定し，どちらを推定に用いたのかを示すこと．

■課題 21　歌声と伴奏の両方を含む適当な楽曲から 10 秒程度の区間を切り出し，その波形，スペクトログラム，SHS で推定したメロディの音高を示せ．

4.4 音源分離

非負値行列因子分解 *non-negative matrix factorization (NMF)* を用いた音源分離に取り組む．

NMF は，$N \times M$ の非負値行列（すべての要素が非負である行列）Y を，2 つの非負値行列 H と U（それぞれ $N \times K$, $K \times M$）に分解する手法である．K は $K \ll M$, $K \ll N$ であるような値をあらかじめ決めておく．このとき，H と U の積が Y を近似するように最適な H と U を推定する．振幅スペクトログラムは $(N, M) =$ (フレーム数, 周波数ビン数) の非負値行列であり，音源（特に楽器音）のスペクトログラムは NMF の性質とよく合致するため，NMF は音源分離や楽器音解析によく用いられている．また，音源分離に限らず，情報圧縮の手段として NMF を用いることもできる．

5 簡易カラオケシステムの制作

■課題 22　WAV ファイルに保存された楽曲の再生とマイクからの歌声の録音を同時に行いながら，楽曲のスペクトログラムと音声の音高（基本周波数）を同時に図示するカラオケシステムを制作せよ．

■課題 23†　上記のカラオケシステムを改良せよ．例えば以下のような改良が考えられる．もちろんこれら以外の改良でもよいので，創意工夫をこらすこと．

- 歌声の音量やスペクトルも同時に表示する．どのように表示すると効果的かを考えること．
- 楽曲のメロディや和音を同時に表示する．
- 有声・無声判定を用いて，不自然な音高の表示を抑制する．
- 歌い終わると（もしくは歌いながら）歌唱内容を採点する．

[*24] 統計学的な意味とは異なるが便宜上尤度と呼ぶ．

- 楽譜を読み込んでおき，いわゆるガイドを同時再生する．
- 歌詞を読み込んでおき，楽曲の再生位置に合わせて歌詞を表示する．

付録 A　Python プログラミングの基礎

以下で述べる内容は Python の機能のごく一部である．Python は様々なドキュメントがオンライン・オフラインを問わず大量に存在しているため，それらも参考にしながら課題を解くこと．

課題に対応したコードが Web で閲覧できることがある．課題を解くためにそれらを参考にすること，受講生同士で相談することなどは禁止しないが，**他者のコードを参考にした場合は情報源を必ず明記すること．**

A.1　四則演算など

```
>>> 5 + 3
8
>>> 5 - 3
2
>>> 5 * 3
15
>>> 5 / 3
1
>>> 5.0 / 3.0
1.6666666666666667
>>> 5 % 3
2
>>> 5 ** 3
125
```

Python では複素数が標準的にサポートされている．数値に接尾辞 `j` もしくは `J` を付けると虚数を表す．`complex(x, y)` でも複素数を生成できる．実部，虚部を取り出すには `real`，`imag` を用いる．

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0, 1)
(-1+0j)
>>> 5+3j * 2
(5+6j)
>>> (5+3j) * 2
(10+6j)
>>> (5.0+3.0j).real
5.0
>>> (5.0+3.0j).imag
3.0
```

A.2　変数

多くのスクリプト言語と同様に，変数の型宣言は不要．

```
>>> a = 5
>>> b = 3
>>> a + b
```

```
8
>>> a = b = 0
>>> a + b
0
>>> a, b, c = 1, 2, 3
>>> a * b * c
6
```

A.3 制御構文

Python の大きな特徴の一つは**ブロック（ループ・条件分岐・関数などの本体）をインデントで表す**ことである．インデントには（複数個の）タブや空白を用いる．インデントの字下げ幅は特に決められていないが，同じ階層のブロックは同じ字下げ幅でなければならない[*25]．インタラクティブな環境でブロックを終了させるには空行を入力する．

```
>>> a, b, i = 0, 1, 0
>>> while i < 10:
...     print i, b
...     a, b, i = b, a + b, i + 1
...
0 1
1 1
2 2
3 3
4 5
5 8
6 13
7 21
8 34
9 55
```

```
>>> a = ['kyoto', 'univ', 'engineering']
>>> for b in a:
...     print b, len(b)
...
kyoto 5
univ 4
engineering 11
```

```
>>> a = int(raw_input("Enter an integer: "))
Enter an integer: 5
>>> if a < 0:
...     print 'Negative'
... elif a == 0:
...     print 'Zero'
... else:
...     print 'Positive'
...
Positive
```

`range()` 関数は数列を生成する．

[*25] タブと空白の混在は悪質なバグの温床となる．標準的なスタイルでは，空白 4 つを 1 段階のインデントとし，タブは使用しない．

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(5, 10, 2)
[5, 7, 9]
>>> range(10, 0, -2)
[10, 8, 6, 4, 2]
```

A.4 モジュールのインポート

Python では数多くのモジュールが用意されている．例えば以下のようにすると，数学関数を定義した math モジュールをインポートする．

```
>>> import math
>>> math.log(2.0)
0.69314718055994529
>>> math.sin(0.5 * math.pi)
1.0
```

モジュール中の特定の関数や変数，もしくはサブモジュールをインポートする．

```
>>> from math import sin, cos, tan, pi
>>> sin(pi / 6)
0.49999999999999994
>>> cos(pi / 6)
0.86602540378443871
>>> tan(pi / 6)
0.57735026918962573
```

A.5 関数の定義

```
>>> def square(n):
...     return n * n
...
>>> square(2)
4
```

```
>>> def fact(n):
...     if n <= 1:
...         return 1
...     else:
...         return n * fact(n - 1)
...
>>> fact(10)
3628800
```

```
>>> def fib(n):
...     def fibiter(a, b, n):
...         if n == 0:
...             return a
...         else:
```

```
...             return fibiter(b, a + b, n - 1)
...     return fibiter(0, 1, n)
...
>>> fib(10)
55
```

A.6 グラフ描画

ソースコード 5　正弦波の描画，線種・色・太さの変更

```
>>> import math
>>> import numpy
>>> import pylab
>>> x = numpy.linspace(0, 2.0 * math.pi, 101)
>>> y = numpy.sin(x)
>>> pylab.plot(x, y)
>>> pylab.plot(x + 0.2, y) # next color if no color parameters
>>> pylab.plot(x + 0.4, y, 'c') # explicit color specification
>>> pylab.plot(x + 0.6, y, ':') # line type
>>> pylab.plot(x + 0.8, y, '--')
>>> pylab.plot(x + 1.0, y, '-.')
>>> pylab.plot(x + 1.2, y, linewidth = 4) # line width
>>> pylab.plot(x + 1.4, y, 'k--', lw = 2) # simultaneous specification
>>> pylab.show()
```

ソースコード 6　マークの変更

```
>>> import math
>>> import numpy
>>> import pylab
>>> x = numpy.linspace(0, 10, 21)
>>> y = x
>>> pylab.plot(x + 0, y, '.') # dot
>>> pylab.plot(x + 1, y, ',') # small dot
>>> pylab.plot(x + 2, y, 'o') # circle
>>> pylab.plot(x + 3, y, '*') # star
>>> pylab.plot(x + 4, y, '+') # plus sign
>>> pylab.plot(x + 5, y, 'x') # multiplication sign
>>> pylab.plot(x + 6, y, '^') # upward triangle
>>> pylab.plot(x + 7, y, 'v') # downward triangle
>>> pylab.plot(x + 8, y, '<') # leftward triangle
>>> pylab.plot(x + 9, y, '>') # rightward triangle
>>> pylab.plot(x + 10, y, 's') # square
>>> pylab.plot(x + 11, y, 'D') # diamond
>>> pylab.plot(x + 12, y, 'd') # slim diamond
>>> pylab.plot(x + 13, y, 'p') # pentagon
>>> pylab.plot(x + 14, y, 'H') # hexagon
>>> pylab.plot(x + 15, y, 'h') # small hexagon
>>> pylab.legend(['.', ',', 'o', '*', '+', 'x', '^', 'v', '<', '>',
...                's', 'D', 'd', 'p', 'H', 'h'], "lower right", numpoints = 3)
>>> pylab.show()
```

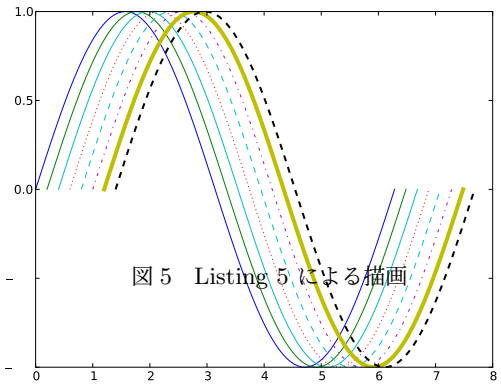


図 5　Listing 5 による描画

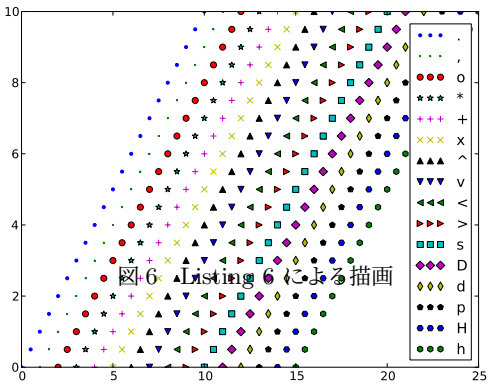


図 6　Listing 6 による描画

A.7　numpy.fft モジュール

ソースコード 7　FFT ルーチン numpy.fft.fft(), numpy.fft.ifft(), numpy.fft.rfft(), numpy.fft.irfft()

```
>>> import math
>>> import numpy
>>> import numpy.fft
>>> t = numpy.arange(8) / 8.0
>>> a = numpy.sin(2.0 * math.pi * t) # 1周期分の正弦波
>>> a
array([  0.00000000e+00,
         7.07106781e-01,
         1.00000000e+00,
         7.07106781e-01,
         1.22464680e-16,
        -7.07106781e-01,
        -1.00000000e+00,
        -7.07106781e-01])
>>> b = numpy.fft.fft(a)
>>> b
array([  1.14423775e-17 +0.00000000e+00j,
        -3.55271772e-15 -4.00000000e+00j,  # ------|
         1.22464680e-16 -1.11022302e-16j,  # ----| |
         3.10186197e-16 -8.88178420e-16j,  # --| | |
         2.33486982e-16 +0.00000000e+00j,  #   | | | 複素共役
        -4.04453250e-21 -8.88178420e-16j,  # --| | |
         1.22464680e-16 +1.11022302e-16j,  # ----| |
         2.75267685e-15 +4.00000000e+00j]) # ------|
>>> br = numpy.fft.rfft(a)  # a[] が実数のみを含む配列のときに適用できる
>>> br
array([  1.14423775e-17 +0.00000000e+00j,  # br[] は b[0 : len(b) / 2 + 1] と等しい
        -1.23268770e-15 -4.00000000e+00j,  # 複素共役な部分を取り除いたもの
         1.22464680e-16 -1.11022302e-16j,
         9.87758345e-16 +0.00000000e+00j,
         2.33486982e-16 +0.00000000e+00j])
>>> numpy.abs(b)
```

```
array([  1.14423775e-17,  # 周波数1に対応するb[1]（とその複素共役であるb[7]）のみが
         4.00000000e+00,  # 大きい値をもつ
         1.65298365e-16,
         9.40784982e-16,
         2.33486982e-16,
         8.88178420e-16,
         1.65298365e-16,
         4.00000000e+00])
>>> c = numpy.fft.ifft(b)
>>> c.real
array([  9.86076132e-32,  # （数値誤差を除いて）変換前と同じ配列
         7.07106781e-01,
         1.00000000e+00,
         7.07106781e-01,
         1.22464680e-16,
        -7.07106781e-01,
        -1.00000000e+00,
        -7.07106781e-01])
>>> cr = numpy.fft.irfft(br)  # irfft() は実数のみを含む配列を返す
>>> cr
array([  9.86076132e-32,  # cr[] は c.real[] と等しい
         7.07106781e-01,
         1.00000000e+00,
         7.07106781e-01,
         1.22464680e-16,
        -7.07106781e-01,
        -1.00000000e+00,
        -7.07106781e-01])
```

A.8 リアルタイム音響信号処理とグラフィックス

ソースコード 8　音響信号再生 player.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import wave
import time
import numpy as np
import alsaaudio
from optparse import OptionParser
from multiprocessing import Queue

BUF_SIZE = 4000 # samples, must be >= 64
DELTA_TIME = 0.001 # 再生時間同期のための最小待ち時間［秒］
WRITE_BEFORE = 0.1 # 再生を途切れさせないための猶予［秒］

FRAME_DURATION = 0.1  # フレームの長さ［秒］
FRAME_RATE = 10.0 # 1秒あたりのフレーム数
FRAME_INTERVAL = 1.0 / FRAME_RATE

# WAVファイル再生用のクラス．
# WAVファイルをコンストラクタで指定し，run()関数で再生を開始する．
# 内部にキューを持っており，再生中の最新フレームが随時格納される．
class player():
```

```
24      def __init__(self, wavfile,
25                   buf_size = BUF_SIZE,
26                   delta_time = DELTA_TIME,
27                   write_before = WRITE_BEFORE,
28                   frame_duration = FRAME_DURATION,
29                   frame_rate = FRAME_RATE,
30                   verbose = False):
31          self.wavfile = wavfile
32          self.buf_size = buf_size
33          self.delta_time = delta_time
34          self.write_before = write_before
35          self.frame_duration = frame_duration
36          self.frame_rate = frame_rate
37          self.frame_interval = 1.0 / self.frame_rate
38          self.verbose = verbose
39
40          self.nchannels = self.wavfile.getnchannels()
41          if self.verbose: print 'player: channels: %d' % (self.nchannels)
42          self.sampling_rate = self.wavfile.getframerate()
43          if self.verbose: print 'player: sampling_rate: %d' % (self.sampling_rate)
44          self.nsamples = self.wavfile.getnframes()
45          if self.verbose: print 'player: number of samples: %d' % (self.nsamples)
46
47          self.out = alsaaudio.PCM(alsaaudio.PCM_PLAYBACK)
48          self.out.setchannels(self.wavfile.getnchannels())
49          self.out.setrate(self.wavfile.getframerate())
50          self.out.setformat(alsaaudio.PCM_FORMAT_S16_LE)
51          self.out.setperiodsize(self.buf_size)
52
53          self.buf = np.ndarray(self.buf_size)
54          self.buf_pos = 0
55
56          self.frame_size = int(self.sampling_rate * self.frame_duration)
57          self.frame_shift = int(self.sampling_rate * self.frame_interval)
58          self.frame = np.zeros(self.frame_size)
59          self.window = np.hanning(self.frame_size)
60          self.frame_replace = min(self.frame_size, self.frame_shift)
61          self.queue = Queue()
62
63      def play(self, size):
64          buf_remain = self.buf_size - self.buf_pos
65          if buf_remain >= size:
66              buf_pos0 = self.buf_pos
67              self.buf_pos += size
68              return self.buf[buf_pos0 : self.buf_pos]
69          else:
70              frame = np.ndarray(size)
71              frame[ : buf_remain] = self.buf[self.buf_pos : ]
72              frame_remain = size - buf_remain
73              while frame_remain > self.buf_size:
74                  d = self.wavfile.readframes(self.buf_size)
75                  self.out.write(d)
76                  w = np.fromstring(d, dtype = 'int16') * (2.0 ** -15)
77                  frame[-frame_remain : -frame_remain + self.buf_size] = w
78                  frame_remain -= self.buf_size
79              if frame_remain > 0:
80                  d = self.wavfile.readframes(self.buf_size)
```

```
                self.out.write(d)
                self.buf = np.fromstring(d, dtype = 'int16') * (2.0 ** -15)
                frame[-frame_remain : ] = self.buf[ : frame_remain]
                self.buf_pos = frame_remain
            return frame

    def run(self):
        t0 = time.time()
        current_sample = 0
        while current_sample < self.nsamples:
            t1 = time.time()
            while (t1 - t0) * self.sampling_rate - self.write_before < current_sample:
                time.sleep(self.delta_time)
                t1 = time.time()
            if self.verbose: print 'player: elapsed: %f, expected: %f' % (
                t1 - t0, float(current_sample) / self.sampling_rate
                )

            frame0 = self.play(self.frame_shift)
            if self.frame_size > self.frame_shift:
                self.frame[ : self.frame_shift] = frame0[-self.frame_shift : ]
                self.frame = np.roll(self.frame, -self.frame_shift)
            else:
                self.frame = frame0
            wframe = self.frame * self.window
            self.queue.put(wframe)
            current_sample += self.frame_shift

def main():
    parser = OptionParser(usage = 'usage: %prog [options] WAVFILE')
    parser.add_option('-v', '--verbose',
                      dest = 'verbose', action = 'store_true', default = False,
                      help = 'verbose output')
    (options, args) = parser.parse_args()
    verbose = options.verbose
    if verbose: print 'verbose output mode.'

    if len(args) == 0:
        print 'no input files.'
        exit

    filename = args[0]
    if verbose: print 'filename: ' + filename
    wavfile = wave.open(filename, 'r')
    p = player(wavfile, verbose = verbose)
    p.run()

if __name__ == '__main__':
    main()
```

ソースコード 9　音響信号録音 recorder.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# import sys
# import wave
```

```
6  import numpy as np
7  import alsaaudio
8  import time
9  from optparse import OptionParser
10 from threading import Thread
11 from multiprocessing import Queue
12
13 BUF_SIZE = 4000 # samples, must be >= 64
14
15 RECORDER_NCHANNELS = 1
16 RECORDER_SAMPLING_RATE = 44100
17 RECORDER_DURATION = 10.0
18 RECORDER_FILENAME = 'out.wav'
19
20 FRAME_DURATION = 0.1 # seconds
21 FRAME_RATE = 10.0
22 FRAME_INTERVAL = 1.0 / FRAME_RATE
23
24 # 音響信号録音用のクラス.
25 # 内部で信号をバッファリングするので
26 # バッファサイズの設定によらず自由な長さの音響信号を取得可能.
27 class recorder(Thread):
28     def __init__(self,
29                  nchannels = RECORDER_NCHANNELS,
30                  sampling_rate = RECORDER_SAMPLING_RATE,
31                  duration = RECORDER_DURATION,
32                  filename = RECORDER_FILENAME,
33                  frame_duration = FRAME_DURATION,
34                  frame_rate = FRAME_RATE,
35                  verbose = False):
36         super(recorder, self).__init__()
37         self.setDaemon(True)
38         self.recorder = recorder
39
40         self.verbose = verbose
41         self.nchannels = nchannels
42         if self.verbose: print 'recorder: channels: %d' % (self.nchannels)
43         self.sampling_rate = sampling_rate
44         if self.verbose: print 'recorder: sampling_rate: %d' % (self.sampling_rate)
45         self.duration = duration
46         if self.verbose: print 'recorder: duration: %f' % (self.duration)
47         self.filename = filename
48         self.frame_duration = frame_duration
49         self.frame_rate = frame_rate
50         self.frame_interval = 1.0 / self.frame_rate
51
52         self.audio_in = alsaaudio.PCM(alsaaudio.PCM_CAPTURE)
53         self.audio_in.setchannels(self.nchannels)
54         self.audio_in.setrate(self.sampling_rate)
55         self.audio_in.setformat(alsaaudio.PCM_FORMAT_S16_LE)
56         self.audio_in.setperiodsize(BUF_SIZE)
57
58         self.frame_size = int(self.sampling_rate * self.frame_duration)
59         self.frame_shift = int(self.sampling_rate * self.frame_interval)
60         self.frame = np.zeros(self.frame_size)
61         self.window = np.hanning(self.frame_size)
62         self.frame_replace = min(self.frame_size, self.frame_shift)
```

```
63
64          self.pcount = 0
65          self.buf = np.ndarray(BUF_SIZE)
66          self.buf_pos = BUF_SIZE
67          self.queue = Queue()
68
69      def record(self, size):
70          buf_remain = len(self.buf) - self.buf_pos
71          if buf_remain >= size:
72              buf_pos0 = self.buf_pos
73              self.buf_pos += size
74              return self.buf[buf_pos0 : self.buf_pos]
75          else:
76              frame = np.ndarray(size)
77              frame[ : buf_remain] = self.buf[self.buf_pos : ]
78              frame_remain = size - buf_remain
79              while frame_remain > BUF_SIZE:
80                  l, d = self.audio_in.read()
81                  self.pcount += 1
82                  w = np.fromstring(d, dtype = 'int16') * (2.0 ** -15)
83                  frame[-frame_remain : -frame_remain + BUF_SIZE] = w
84                  frame_remain -= BUF_SIZE
85              if frame_remain > 0: # 0 < frame_remain < BUF_SIZE
86                  l, d = self.audio_in.read()
87                  self.pcount += 1
88                  w = np.fromstring(d, dtype = 'int16') * (2.0 ** -15)
89                  self.buf[:len(w)] = w[:]
90                  frame[-frame_remain : ] = self.buf[ : frame_remain]
91                  self.buf_pos = frame_remain
92              return frame
93
94      def run(self):
95          while True:
96              frame0 = self.record(self.frame_shift)
97              if (self.frame_size > self.frame_shift):
98                  self.frame[ : self.frame_shift] = frame0[-self.frame_shift : ]
99                  self.frame = np.roll(self.frame, -self.frame_shift)
100             else:
101                 self.frame = frame0
102             wframe = self.frame * self.window
103             if self.verbose:
104                 rms = np.sqrt(np.sum(wframe * wframe) / len(wframe))
105                 print 'recorder: rms: %g' % (rms)
106             self.queue.put(wframe)
107
108 def main():
109     parser = OptionParser()
110     parser.add_option('-v', '--verbose',
111                       dest = 'verbose', action = 'store_true', default = False,
112                       help = 'verbose output')
113     (options, args) = parser.parse_args()
114     verbose = options.verbose
115     if verbose: print 'verbose output mode.'
116
117     r = recorder(verbose = verbose)
118     r.start()
119     time.sleep(10)
```

```
120
121 if __name__ == '__main__':
122     main()
```

ソースコード 10　録音・再生・表示の同時処理

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import math
5  import wave
6  import numpy as np
7  import time
8  import matplotlib.pyplot as plt
9  from optparse import OptionParser
10 from threading import Thread
11 from multiprocessing import Process
12
13 import player
14 import recorder
15
16 # WAVファイルの再生とマイクからの録音を同時に行いながら
17 # それらの短時間フレームのRMSをリアルタイムに表示する.
18
19 DELTA_TIME = 0.001 # seconds
20 PLOT_FRAMES = 50
21 PLOT_FREQ_MAX = 2000.0
22
23 class plotter(object):
24     def __init__(self, player, recorder,
25                  plot_frames = PLOT_FRAMES,
26                  plot_freq_max = PLOT_FREQ_MAX,
27                  delta_time = DELTA_TIME):
28         self.p = player
29         self.r = recorder
30         self.plot_frames = plot_frames
31         self.plot_freq_max = plot_freq_max
32         self.delta_time = delta_time
33
34         self.current_frame = -1
35         self.p_rms = np.zeros(self.plot_frames)
36         self.r_rms = np.zeros(self.plot_frames)
37         self.p_spectrogram = np.ndarray([self.plot_frames, self.p.frame_size / 2 + 1])
38         self.p_spectrogram.fill(20.0 * math.log10(1.0e-1))
39         self.r_spectrogram = np.ndarray([self.plot_frames, self.r.frame_size / 2 + 1])
40         self.r_spectrogram.fill(20.0 * math.log10(1.0e-1))
41         self.p_nyquist = self.p.sampling_rate * 0.5
42         self.r_nyquist = self.r.sampling_rate * 0.5
43         self.p_freqs = np.linspace(0.0, self.p_nyquist, self.p.frame_size / 2 + 1)
44         self.r_freqs = np.linspace(0.0, self.r_nyquist, self.r.frame_size / 2 + 1)
45
46         self.p_plot_freq_max_index = next(
47             x[1] for x in zip(self.p_freqs, range(len(self.p_freqs)))
48             if x[0] > self.plot_freq_max
49             )
50         self.r_plot_freq_max_index = next(
51             x[1] for x in zip(self.r_freqs, range(len(self.r_freqs)))
```

```
            if x[0] > self.plot_freq_max
            )

        self.updated = True

    def __call__(self):
        plt.ion()
        loop = plotterloop(self)
        loop.start()
        while True:
            if not self.updated:
                time.sleep(self.delta_time)
                continue
            plt.clf()
            self.plot_player_rms()
            self.show_player_spec()
            self.plot_recorder_rms()
            self.show_recorder_spec()
            plt.draw()
            self.updated = False

    def plot_player_rms(self):
        plt.subplot(221)
        times = (
            np.arange(self.plot_frames) - self.plot_frames + self.current_frame + 1
            ) * self.p.frame_interval
        plt.plot(times, self.p_rms, 'o-b')

    def show_player_spec(self):
        plt.subplot(222)
        extent = [(self.current_frame - self.plot_frames) * self.p.frame_interval,
                  self.current_frame * self.p.frame_interval,
                  0.0, self.plot_freq_max]
        plt.imshow(np.transpose(self.p_spectrogram[:, : self.p_plot_freq_max_index]),
                   extent = extent, origin = 'lower', aspect = 'auto')

    def plot_recorder_rms(self):
        plt.subplot(223)
        times = (
            np.arange(self.plot_frames) - self.plot_frames + self.current_frame + 1
            ) * self.r.frame_interval
        plt.plot(times, self.r_rms, 'o-b')

    def show_recorder_spec(self):
        plt.subplot(224)
        extent = [(self.current_frame - self.plot_frames) * self.r.frame_interval,
                  self.current_frame * self.r.frame_interval,
                  0.0, self.plot_freq_max]
        plt.imshow(np.transpose(self.r_spectrogram[:, : self.r_plot_freq_max_index]),
                   extent = extent, origin = 'lower', aspect = 'auto')

    def update(self):
        if not self.p.queue.empty():
            wframe = self.p.queue.get()
            self.p_rms[0] = self.rms(wframe)
            self.p_rms = np.roll(self.p_rms, -1)
            self.p_spectrogram[0] = np.log(np.abs(np.fft.rfft(wframe)))
```

```
109             self.p_spectrogram = np.roll(self.p_spectrogram, -1, axis = 0)
110 
111         if not self.r.queue.empty():
112             wframe = self.r.queue.get()
113             self.r_rms[0] = self.rms(wframe)
114             self.r_rms = np.roll(self.r_rms, -1)
115             self.r_spectrogram[0] = np.log(np.abs(np.fft.rfft(wframe)))
116             self.r_spectrogram = np.roll(self.r_spectrogram, -1, axis = 0)
117 
118         self.updated = True
119 
120     def rms(self, a):
121         return math.sqrt(np.dot(a, a) / len(a))
122 
123 class plotterloop(Thread):
124     def __init__(self, plotter):
125         super(plotterloop, self).__init__()
126         self.setDaemon(True)
127         self.plotter = plotter
128 
129     def run(self):
130         while True:
131             if self.plotter.p.queue.empty() and self.plotter.r.queue.empty():
132                 time.sleep(DELTA_TIME)
133                 continue
134             self.plotter.update()
135 
136 
137 def main():
138     parser = OptionParser()
139     parser.add_option('-v', '--verbose',
140                       dest = 'verbose', action = 'store_true', default = False,
141                       help = 'verbose output')
142     (options, args) = parser.parse_args()
143     verbose = options.verbose
144     if verbose: print 'verbose output mode.'
145 
146     if len(args) == 0:
147         print 'no input files.'
148         exit
149 
150     filename = args[0]
151     if verbose: print 'filename: ' + filename
152     wavfile = wave.open(filename, 'r')
153     p = player.player(wavfile)
154 
155     r = recorder.recorder()
156     r.start()
157 
158     pl = plotter(p, r)
159     proc = Process(target = pl)
160     proc.daemon = True
161     proc.start()
162 
163     p.run()
164 
165 if __name__ == '__main__':
```

```
166     main()
```