計算機科学実験及演習 4「音楽情報処理」

担当：糸山 克寿，吉井 和佳　TA：津島 啓晃，和田 雄介

2017 年度版（最終更新 2017 年 10 月 11 日）

1　はじめに

最新版の資料・ライブラリ・ドキュメントは演習ページから参照できる．演習期間内も更新することがあるので，こまめに最新版をチェックすること．

■演習 xxx に沿って演習を進めることを想定しているが，これらそのものは成績評価には含めない．演習を無視して課題のみを解いても構わない．**■演習** xxx† は内容をより深く理解するための高度な演習である．余力があれば取り組んでもよいだろう．

本演習では **Java SE Development Kit (JDK) 8**（もしくは 8 以降）を用いる．バージョン 7 以前の JDK では動作しないコードがある．さらに以下のライブラリを用いる．

- 数学・統計ライブラリ Commons Math
- コマンドライン処理ライブラリ Commons CLI
- 本演習用ライブラリ[*1]

ライブラリファイルをダウンロード・展開し，環境変数 `CLASSPATH` にライブラリの JAR ファイルを追加する．例えば `~/.bash_profile` に以下を追加し，シェルを再起動する．`/path/to/library/`の部分，およびバージョン番号は環境に合わせて適切に書き換えること．

```
CLASSPATH=.:\
/path/to/library/commons-math3-3.6.1.jar:\
/path/to/library/commons-cli-1.4.jar:\
/path/to/library/le4music.jar
export CLASSPATH
```

[*1] 本ライブラリは本演習の受講者のみが，演習遂行の目的でのみ利用可能である．

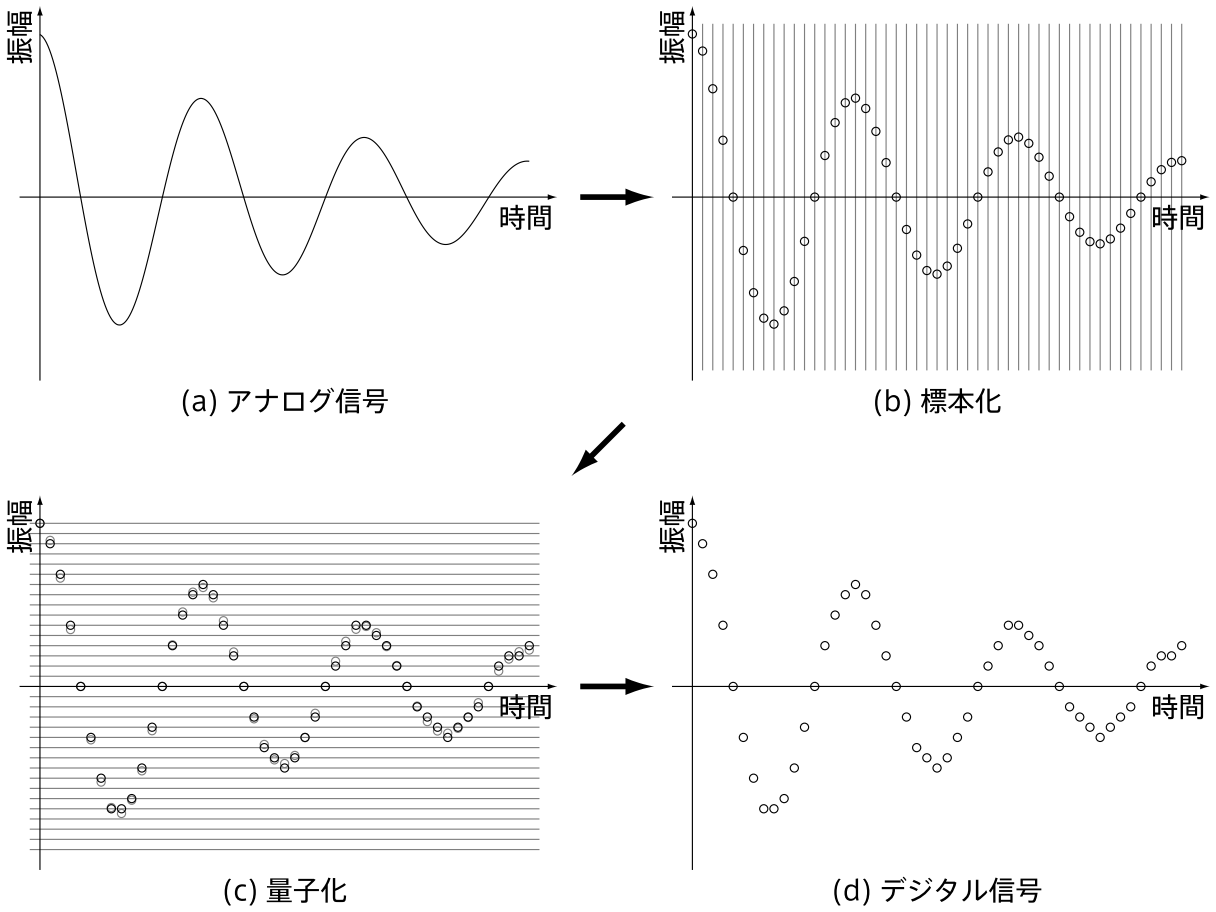


図 1　アナログ信号のデジタイズ

2　音響信号処理の基礎（理論編）

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．

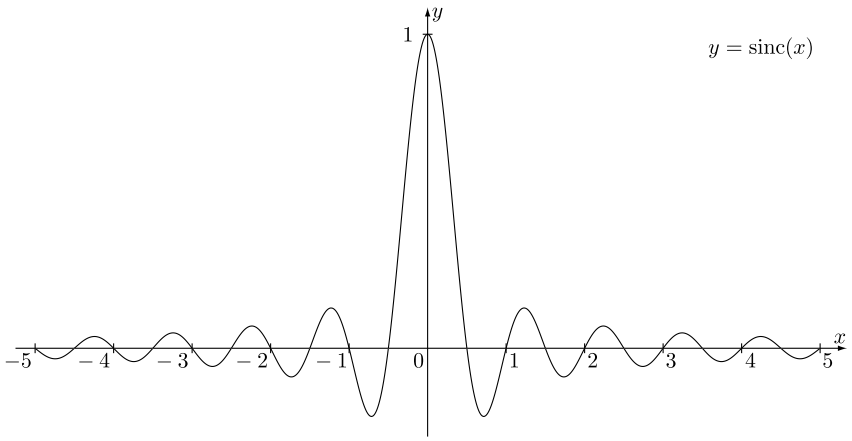


図 2　sinc 関数

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. 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] 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．

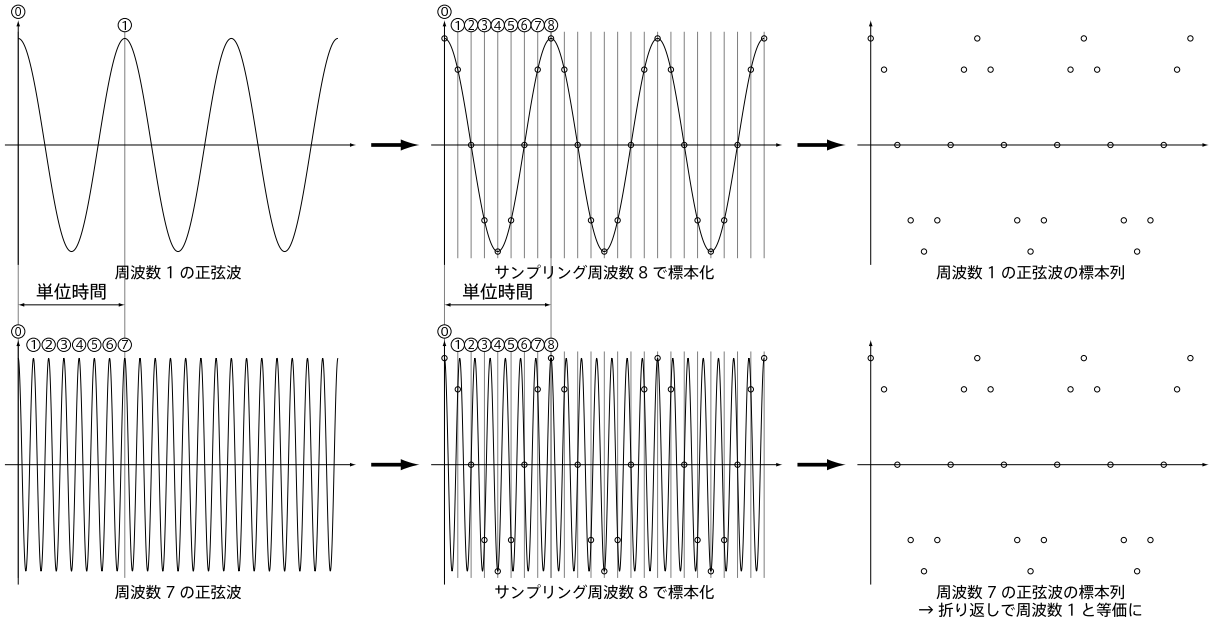


図 3　折り返し雑音の例．周波数 1 と周波数 7 の正弦波をサンプリング周波数 8（ナイキスト周波数 4）で標本化すると，周波数 7 の正弦波は折り返しにより周波数 1 の標本列と等価になる．

2.2　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 $x(t)$ 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]．

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

$X(f)$ から元の信号 $x(t)$ を復元でき，この変換を**逆フーリエ変換 inverse Fourier transform** と呼ぶ．

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

フーリエ変換（および逆フーリエ変換）は 2 つの重要な性質をもつ．

線型性　a, b は適当な定数．

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

畳み込みと乗算の相互変換　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．

[*3] 定義にはいくつかの流儀がある．

2.2.1 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 $(x_0, \ldots, x_{N-1})$ 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．

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

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．

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

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．

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分解可能な場合に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．

2.2.2 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

[*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．

[*5] short-term とする場合もある．

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

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

   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）**に対する様々な処理によって信号を分析することは音響信号処理では頻繁に用いられ，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　音響信号処理の基礎（実践編）

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．ここでは音響信号プログラム群 SoX[*7] が提供するコマンドの一つである `rec` を用いる．例えば以下のコマンドで，サンプリング周波数 16 kHz，符号あり 16 ビット標本化，モノラル（1 チャネル），10 秒間の音響信号を `output.wav` に保存する．

[*6] `http://julius.sourceforge.jp/`
[*7] `http://sox.sourceforge.net/`

```
$ rec -r 16k -b 16 -c 1 output.wav trim 0 10
```

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．ここでは録音と同様に SoX が提供するコマンドの一つである `play` を用いる．以下のコマンドで `input.wav` に格納された音響信号を再生する．

```
$ play input.wav
```

2.1 秒から 1.5 秒間（3.6 秒まで）を切り出して再生する．

```
$ play input.wav trim 2.1 1.5
```

切り出された区間を再生し，さらにそれを 9 回反復する（合計 10 回再生する）．

```
$ play input.wav trim 2.1 1.5 repeat 9
```

■演習 1　数秒程度で「あいうえお」と発声し，これをサンプリング周波数 16kHz，量子化ビット数 16，モノラルで録音せよ．「あーいーうーえーおー」と区切らない連続的な発音と「あ・い・う・え・お」と一音ずつ区切った発音との 2 種類を録音すること．保存した音声は必ず聴取し，以下の項目を確認すること．

- 音量は必要十分で，小さすぎたり大きすぎたりしない．音量が小さすぎる場合はファンのノイズや他者の発話が聞こえる，音量が大きすぎる場合は振幅が ±1 の範囲を超えて大きなノイズが混ざる（いわゆるクリッピング，音割れ）などが起こる．
- 音が歪んでいない．
- 発話の先頭や末尾が途切れていない．
- 必要以上の無音区間がない．発話前後の無音区間の長さは 1 秒未満を目安とする．無音区間が長すぎる場合は SoX の `trim` エフェクトを用いるとよい．

なお，今後この資料で特に断りなく「あいうえお」と記したときは，ここで録音した『「あーいーうーえーおー」と区切らない連続的な発音』を指しているものとする．

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．付録 B の PlotWaveformCLI.java も参考にするとよい．

ソースコード 1　WAVE ファイル読み込みと波形プロット PlotWaveformSimple.java

```
1 import java.io.File;
2 import java.util.stream.Collectors;
3 import java.util.stream.IntStream;
4 import javax.sound.sampled.AudioSystem;
5 import javax.sound.sampled.AudioFormat;
6 import javax.sound.sampled.AudioInputStream;
```

```
7
8  import javafx.application.Application;
9  import javafx.application.Platform;
10 import javafx.stage.Stage;
11 import javafx.scene.Scene;
12 import javafx.scene.chart.XYChart;
13 import javafx.scene.chart.LineChart;
14 import javafx.scene.chart.NumberAxis;
15 import javafx.collections.ObservableList;
16 import javafx.collections.FXCollections;
17
18 import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
19
20 import java.io.IOException;
21 import javax.sound.sampled.UnsupportedAudioFileException;
22
23 public final class PlotWaveformSimple extends Application {
24
25   @Override public final void start(final Stage primaryStage)
26     throws IOException,
27            UnsupportedAudioFileException {
28     /* コマンドライン引数処理 */
29     final String[] args = getParameters().getRaw().toArray(new String[0]);
30     if (args.length < 1) {
31       System.out.println("WAVFILE is not given.");
32       Platform.exit();
33       return;
34     }
35     final File wavFile = new File(args[0]);
36
37     /* WAVファイル読み込み */
38     final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
39     final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
40     final AudioFormat format = stream.getFormat();
41     final double sampleRate = format.getSampleRate();
42     stream.close();
43
44     /* データ系列を作成 */
45     final ObservableList<XYChart.Data<Number, Number>> data =
46       IntStream.range(0, waveform.length)
47         .mapToObj(i -> new XYChart.Data<Number, Number>(i / sampleRate, waveform[i]))
48         .collect(Collectors.toCollection(FXCollections::observableArrayList));
49
50     /* データ系列に名前をつける */
51     final XYChart.Series<Number, Number> series = new XYChart.Series<>();
52     series.setName("Waveform");
53     series.setData(data);
54
55     /* 軸を作成 */
56     final NumberAxis xAxis = new NumberAxis();
57     xAxis.setLabel("Time (seconds)");
58     final NumberAxis yAxis = new NumberAxis();
59     yAxis.setLabel("Amplitude");
60
61     /* チャートを作成 */
62     final LineChart<Number, Number> chart = new LineChart<>(xAxis, yAxis);
63     chart.setTitle("Waveform");
```

```
64      chart.setCreateSymbols(false);
65      chart.getData().add(series);
66
67      /* グラフ描画 */
68      final Scene scene = new Scene(chart, 800, 600);
69
70      /* ウインドウ表示 */
71      primaryStage.setScene(scene);
72      primaryStage.setTitle(getClass().getName());
73      primaryStage.show();
74    }
75
76  }
```

報告書には適切な図を配置すること．下記コードで javafx.scene.Scene に描画されたチャート等を画像ファイルに保存することができる．

ソースコード 2　チャートを画像ファイルに保存するサンプルコード

```
1  import java.io.File;
2  import javax.imageio.ImageIO;
3  import javafx.scene.image.WritableImage;
4  import javafx.embed.swing.SwingFXUtils;
5
6  Platform.runLater(() -> {
7    WritableImage image = scene.snapshot(null);
8    String name = "waveform";
9    String ext = "png";
10   try {
11     ImageIO.write(SwingFXUtils.fromFXImage(image, null), ext, new File(name + "." + ext));
12   } catch (IOException e) { throw new RuntimeException(e); }
13 });
```

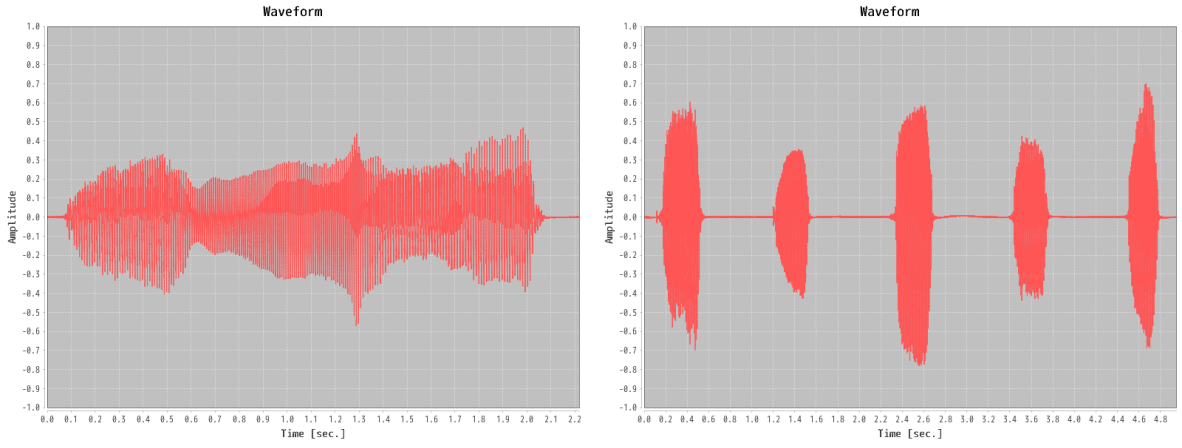


図 4　「あいうえお」の区切らない発音（左）と区切った発音（右）の波形．

3.3 スペクトル

波形は音声の特徴の一部を示すが，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．

より多くの音声の特徴を捉えるためには，音声信号をフーリエ変換して得られるスペクトルを用いる．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（実数列）をフーリエ変換すると，その結果である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[*8]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．

ソースコード 3　PlotSpectrumSimple.java

```
1  import java.io.File;
2  import java.util.Arrays;
3  import java.util.stream.Collectors;
4  import java.util.stream.IntStream;
5  import javax.sound.sampled.AudioSystem;
6  import javax.sound.sampled.AudioFormat;
7  import javax.sound.sampled.AudioInputStream;
8
9  import javafx.application.Application;
10 import javafx.application.Platform;
11 import javafx.stage.Stage;
12 import javafx.scene.Scene;
13 import javafx.scene.chart.XYChart;
14 import javafx.scene.chart.LineChart;
15 import javafx.scene.chart.NumberAxis;
16 import javafx.collections.ObservableList;
17 import javafx.collections.FXCollections;
18
19 import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
20 import org.apache.commons.math3.complex.Complex;
21
22 import java.io.IOException;
23 import javax.sound.sampled.UnsupportedAudioFileException;
24
25 public final class PlotSpectrumSimple extends Application {
26
27   @Override public final void start(final Stage primaryStage)
28     throws IOException,
29            UnsupportedAudioFileException {
30     /* コマンドライン引数処理 */
31     final String[] args = getParameters().getRaw().toArray(new String[0]);
32     if (args.length < 1) {
33       System.out.println("WAVFILE is not given.");
34       Platform.exit();
35       return;
36     }
```

[*8] 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．

```
37      final File wavFile = new File(args[0]);
38
39      /* WAVファイル読み込み */
40      final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
41      final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
42      final AudioFormat format = stream.getFormat();
43      final double sampleRate = format.getSampleRate();
44      stream.close();
45
46      /* fftSize = 2^p >= waveform.length を満たす fftSize を求める
47       * 2^p はシフト演算で求める */
48      final int fftSize = 1 << Le4MusicUtils.nextPow2(waveform.length);
49      final int fftSize2 = (fftSize >> 1) + 1;
50      /* 信号の長さを fftSize に伸ばし，長さが足りない部分は0で埋める．
51       * 振幅を信号長で正規化する． */
52      final double[] src =
53        Arrays.stream(Arrays.copyOf(waveform, fftSize))
54              .map(w -> w / waveform.length)
55              .toArray();
56      /* 高速フーリエ変換を行う */
57      final Complex[] spectrum = Le4MusicUtils.rfft(src);
58
59      /* 対数振幅スペクトルを求める */
60      final double[] specLog =
61        Arrays.stream(spectrum)
62              .mapToDouble(c -> 20.0 * Math.log10(c.abs()))
63              .toArray();
64
65      /* データ系列を作成 */
66      final ObservableList<XYChart.Data<Number, Number>> data =
67        IntStream.range(0, fftSize2)
68          .mapToObj(i -> new XYChart.Data<Number, Number>(i * sampleRate / fftSize, specLog[i]))
69          .collect(Collectors.toCollection(FXCollections::observableArrayList));
70
71      /* データ系列に名前をつける */
72      final XYChart.Series<Number, Number> series = new XYChart.Series<>("Spectrum", data);
73
74      /* 軸を作成 */
75      final NumberAxis xAxis = new NumberAxis();
76      xAxis.setLabel("Frequency (Hz)");
77      final NumberAxis yAxis = new NumberAxis();
78      yAxis.setLabel("Amplitude (dB)");
79
80      /* チャートを作成 */
81      final LineChart<Number, Number> chart = new LineChart<>(xAxis, yAxis);
82      chart.setTitle("Spectrum");
83      chart.setCreateSymbols(false);
84      chart.getData().add(series);
85
86      /* グラフ描画 */
87      final Scene scene = new Scene(chart, 800, 600);
88
89      /* ウインドウ表示 */
90      primaryStage.setScene(scene);
91      primaryStage.setTitle(getClass().getName());
92      primaryStage.show();
93    }
```

```
94
95  }
```

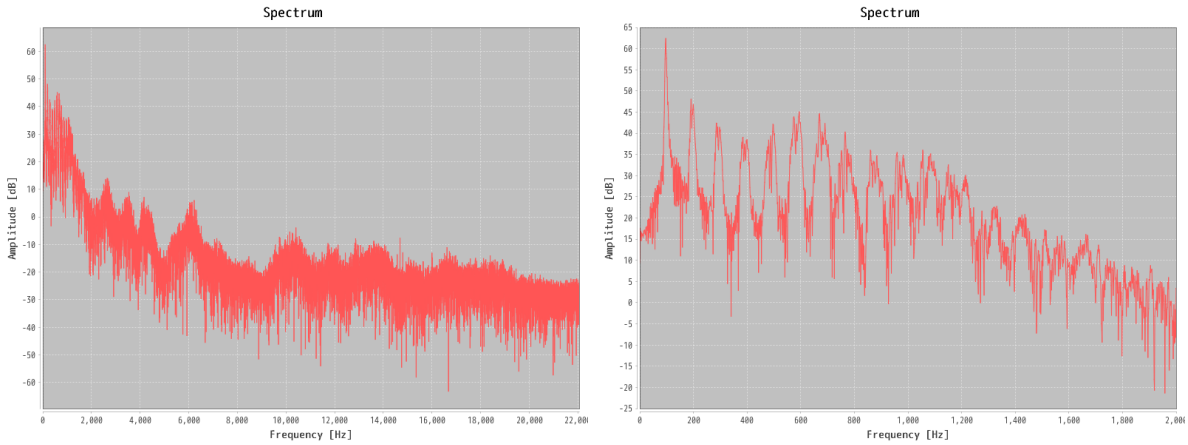


図 5　「あ」のスペクトル（左）とその 0 Hz から 2000 Hz の範囲の拡大図（右）.

■演習 2　「あいうえお」の全区間と「あ」「い」「う」「え」「お」の各区間をフーリエ変換し，その振幅スペクトルを図示せよ．横軸が周波数（0 Hz 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．

■演習 3†　離散フーリエ変換と高速フーリエ変換を実装し，計算時間を比較せよ．

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．

ソースコード 4　PlotSpectrogramSimple.java

```
1  import java.io.File;
2  import java.util.Arrays;
3  import java.util.stream.Collectors;
4  import java.util.stream.Stream;
5  import java.util.stream.IntStream;
6  import javax.sound.sampled.AudioSystem;
7  import javax.sound.sampled.AudioFormat;
8  import javax.sound.sampled.AudioInputStream;
9
10 import javafx.application.Application;
11 import javafx.application.Platform;
12 import javafx.stage.Stage;
13 import javafx.scene.Scene;
14 import javafx.scene.chart.XYChart;
15 import javafx.scene.chart.LineChart;
16 import javafx.scene.chart.NumberAxis;
17 import javafx.collections.ObservableList;
18 import javafx.collections.FXCollections;
19
```

```java
import org.apache.commons.math3.complex.Complex;
import org.apache.commons.math3.util.MathArrays;

import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
import jp.ac.kyoto_u.kuis.le4music.LineChartWithSpectrogram;

import java.io.IOException;
import javax.sound.sampled.UnsupportedAudioFileException;

public final class PlotSpectrogramSimple extends Application {

  @Override public final void start(final Stage primaryStage)
    throws IOException,
           UnsupportedAudioFileException {
    /* コマンドライン引数処理 */
    final String[] args = getParameters().getRaw().toArray(new String[0]);
    if (args.length < 1) {
      System.out.println("WAVFILE is not given.");
      Platform.exit();
      return;
    }
    final File wavFile = new File(args[0]);

    final double frameDuration = Le4MusicUtils.frameDuration;
    final double shiftDuration = frameDuration / 8.0;

    /* WAVファイル読み込み */
    final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
    final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
    final AudioFormat format = stream.getFormat();
    final double sampleRate = format.getSampleRate();
    stream.close();

    /* 窓関数とFFTのサンプル数 */
    final int frameSize = (int)Math.round(frameDuration * sampleRate);
    final int fftSize = 1 << Le4MusicUtils.nextPow2(frameSize);
    final int fftSize2 = (fftSize >> 1) + 1;

    /* シフトのサンプル数 */
    final int shiftSize = (int)Math.round(shiftDuration * sampleRate);

    /* 窓関数を求め正規化する */
    final double[] window = MathArrays.normalizeArray(
      Arrays.copyOf(Le4MusicUtils.hanning(frameSize), fftSize), 1.0
    );

    /* 短時間フーリエ変換本体 */
    final Stream<Complex[]> spectrogram =
      Le4MusicUtils.sliding(waveform, window, shiftSize)
                   .map(frame -> Le4MusicUtils.rfft(frame));

    /* 複素スペクトログラムを対数振幅スペクトログラムに */
    final double[][] specLog =
      spectrogram.map(sp -> Arrays.stream(sp)
                                  .mapToDouble(c -> 20.0 * Math.log10(c.abs()))
                                  .toArray())
                 .toArray(n -> new double[n][]);
```

```
77
78      /* X軸を作成 */
79      final NumberAxis xAxis = new NumberAxis();
80      xAxis.setLabel("Time (Seconds)");
81      xAxis.setLowerBound(0.0);
82      xAxis.setUpperBound(specLog.length * shiftDuration);
83
84      /* Y軸を作成 */
85      final NumberAxis yAxis = new NumberAxis();
86      yAxis.setLabel("Frequency (Hz)");
87      yAxis.setLowerBound(0.0);
88      yAxis.setUpperBound(sampleRate * 0.5);
89
90      /* chチャートを作成 */
91      final LineChartWithSpectrogram<Number, Number> chart =
92        new LineChartWithSpectrogram<>(xAxis, yAxis);
93      chart.setParameters(specLog.length, fftSize2, sampleRate * 0.5);
94      chart.setTitle("Spectrogram");
95      Arrays.stream(specLog).forEach(chart::addSpecLog);
96      chart.setCreateSymbols(false);
97      chart.setLegendVisible(false);
98
99      /* グラフ描画 */
100     final Scene scene = new Scene(chart, 800, 600);
101
102     /* ウインドウ表示 */
103     primaryStage.setScene(scene);
104     primaryStage.setTitle(getClass().getName());
105     primaryStage.show();
106   }
107
108 }
```

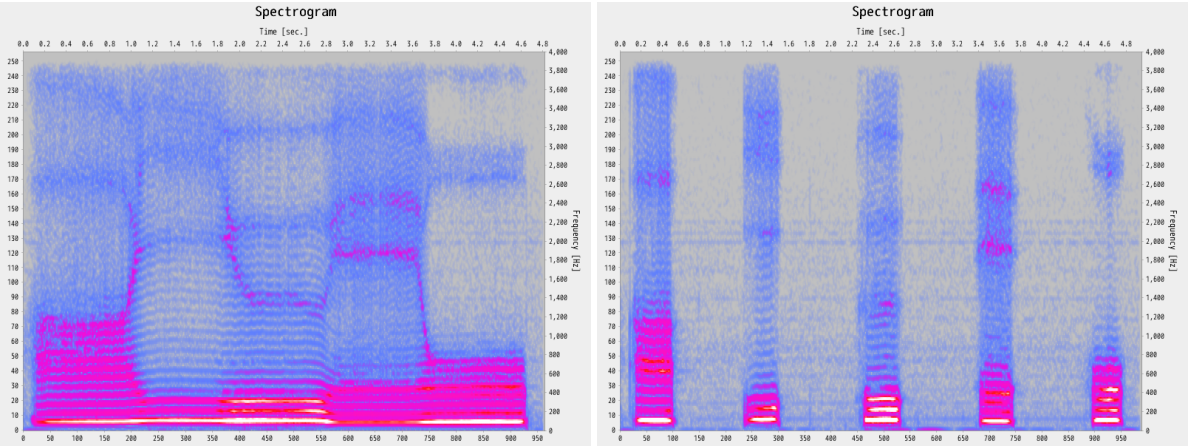


図 6　連続的な「あいうえお」（左）と区切った「あいうえお」（右）のスペクトログラム．

■演習 4　「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．

■演習 5† 逆短時間フーリエ変換を実装せよ．

4 音声の分析

4.1 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[*9]．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[*10]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．

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

■演習 6 2 種類の「あいうえお」をフレームに分割し，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．

4.2 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．

4.2.1 調波構造

以下の実験を行ってみよ．

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．
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，

[*9] 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．可聴域外（20Hz 以下，20kHz 以上）は音として感じることができない．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を参照せよ．

[*10] 平均をとるのは信号長 N で正規化するためである．

（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．

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，等しくこのような性質を持っている．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，$\omega_1 = 440\text{Hz}, \omega_2 = 880\text{Hz}, \omega_3 = 1320\text{Hz}, \omega_4 = 1760\text{Hz}, \omega_5 = 2200\text{Hz}, ...$ という正弦波の和からなり[*11]，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[*12]）を**基本周波数 fundamental frequency**[*13] と呼び，その正弦波を基音と呼ぶ．また，基本周波数の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[*14]．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．

■演習 7　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．

■演習 8†　自分の声域を求めよ．

4.2.2　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．

■演習 9　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．

[*11] 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．
[*12] 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．
[*13] 基本周波数のことを F0 と呼ぶことがある．
[*14] 基音のことを第 1 倍音とも呼ぶ．

■演習 10† 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．

■演習 11† 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．

4.3 音色の抽出

音色[*15] **timbre** について単一の軸や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．

4.3.1 ケプストラム

ケプストラム cepstrum[*16] は，対数振幅スペクトルのフーリエ変換で定義される．

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

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** の振動に対応する．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．

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．

[*15] 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．

[*16] **spec**trum の先頭 4 文字の逆転．

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**．

■演習 12　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．

■演習 13　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．

4.3.2　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[*17]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．

■演習 14　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．

4.4　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．

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

[*17] 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．

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，

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

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

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．

■演習 15　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．認識された音素の推移をスペクトログラムに並べて（もしくは重ねて）示せ．例えば「あ」=0,「い」=1,「う」=2,「え」=3,「お」=4 とした時系列のグラフにするなど．

5　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．

5.1　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[*18]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．

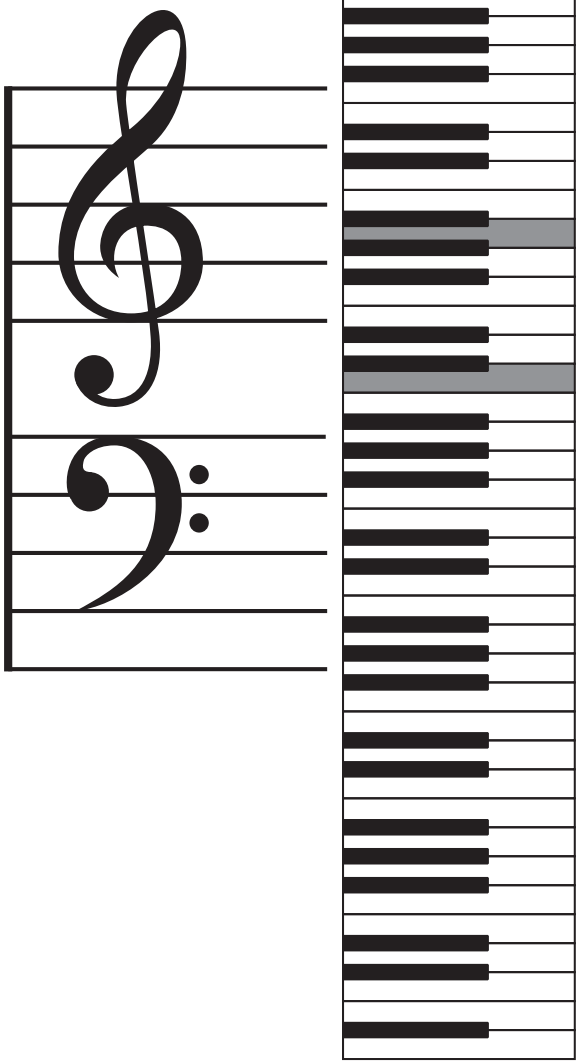
コンピュータ音楽の分野では **Musical Instrument Digital Interface (MIDI)** と呼ばれる規格に沿った電子楽器や楽譜フォーマットが幅広く用いられている．MIDI ではそれぞれの音高に 0 から 127 の**ノートナンバー note number** を与え，この番号を用いて音高を表現する．オクターブ番号と同様に，小さいノートナンバーは低い音高を，大きいノートナンバーは高い音高を表す．ノートナンバー，音名，基本周波数などの関係を図 7 に示す．基準となるのがノートナンバー 69 であり，以下のような音高である．

- 周波数 440Hz
- 音名 A4[*19]
- ピアノの中央のラ
- テレビ放送[*20]などの時報における予告音[*21]

ノートナンバー n と周波数 f (Hz) は以下で相互変換できる．

$$
\begin{aligned}
f &= 440 \times 2^{(n-69)/12} \\
n &= 12 \log_2 \frac{f}{440} + 69
\end{aligned}
\tag{29}
$$

[*18] 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．
[*19] 440Hz を A3 と定める流儀もある．
[*20] 電話の時報は「シ」の音と決められている．
[*21] 「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．



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 7　MIDI ノートナンバー，音名，基本周波数の関係

5.2 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[*22]，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[*23]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．

$$\forall r \in \{\mathrm{C}, \mathrm{C}\#, \ldots, \mathrm{B}\} : \mathrm{CV}(r) = \sum_{\mathrm{nn}(f)=c} X_f \tag{30}$$

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[*24]．

$$\forall c \in \{\mathrm{C}, \mathrm{C}\#, \ldots, \mathrm{B}\} \times \{\mathrm{Major}, \mathrm{Minor}\} : \mathcal{L}(c) = a_{\mathrm{root}}\mathrm{CV}(c_{\mathrm{root}}) + a_{3\mathrm{rd}}\mathrm{CV}(c_{3\mathrm{rd}}) + a_{5\mathrm{th}}\mathrm{CV}(c_{5\mathrm{th}}) \tag{31}$$

$c_{\mathrm{root}}, c_{3\mathrm{rd}}, c_{5\mathrm{th}}$ 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，$c_{\mathrm{root}} = \mathrm{C}$, $c_{3\mathrm{rd}} = \mathrm{E}$, $c_{5\mathrm{th}} = \mathrm{G}$ である．$a_{\mathrm{root}}, a_{3\mathrm{rd}}, a_{5\mathrm{th}}$ は適当な定数で，例えば $(a_{\mathrm{root}}, a_{3\mathrm{rd}}, a_{5\mathrm{th}}) = (1.0, 0.5, 0.8)$ とするとよい．

■演習 16　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．

5.3 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, . . . , 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

[*22] 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．

[*23] 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．

[*24] 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．

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)

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[*25]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．

■演習 17　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．

■演習 18　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．

5.4　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．

[*25] 統計学的な意味とは異なるが便宜上尤度と呼ぶ．

課題 1 音響信号可視化 GUI 作成

音響信号ファイルを読み込み，音響信号のさまざまな情報を表示するグラフィカルユーザーインタフェースを作成せよ．少なくとも以下の 3 つを同時に表示させること．

1. 音響信号のスペクトログラム
2. 音響信号の基本周波数
3. 母音などの何らかの識別を行った結果

余力があれば，このインタフェースがより便利なものになるように改良せよ．例えば以下のような改良が考えられるが，もちろんこれ以外の改良でも構わない．創意工夫すること．

- 音響信号の区間を選択し，その区間のスペクトルを表示する．
- 音響信号を再生し，その再生位置をアニメーションで示す．
- 音楽音響信号のコードとその区間を認識し表示する．

6　リアルタイム音響信号処理プログラミング

第 3 節ではバッチでの音響信号処理プログラミングを行った．バッチ処理は長時間の音響信号の分析や短時間の音響信号の詳細な分析に向くが，原則として録音された音響信号に対してしか適用できない．一方で，音響信号を録音しながら，もしくは再生しながらの分析ができれば，「今何が起こっているのか」をその場で分析でき，音響信号の聴覚的な特徴とスペクトルなどの視覚的な特徴の対応付けが容易となる．本節ではこのような**リアルタイムでの音響信号処理**を実現するためのプログラミングを行う．

Java でのリアルタイム処理実現には複数の手段があるが，ここでは `ScheduledExecutorService` インタフェースを実装したクラスを用いる．このインタフェースにはタスクを繰り返し実行するための 2 つの手続き `scheduleAtFixedRate` と `scheduleWithFixedDelay` が定義されており，これらを用いて波形のプロット，フーリエ変換，特徴量抽出などのタスクを一定時間（例えば 100 ミリ秒）ごとに実行する．演習用ライブラリの `Recorder` および `Player` はそれぞれ音響信号の録音と再生を行うクラスである．両クラスにはメソッド `currentFrame` が定義されており，このメソッドの返り値は録音／再生した音響信号から切り出した最新のフレームである．録音／再生が進むにつれてフレームの内容も変化するため，このメソッドから得たフレームに対する処理をタスクとして `scheduleAtFixedRate` や `scheduleWithFixedDelay` で定期実行させることでリアルタイム処理を実現する．

課題 2 簡易カラオケシステムの制作

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．

オプション課題

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．
- 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　Java ライブラリ

仮想マシンから利用可能なオーディオデバイスを調べるクラス CheckAudioSystem の使用例．

```
% java jp.ac.kyoto_u.kuis.le4music.CheckAudioSystem
13 Mixer(s) are available.
Mixer 0: default [default], version 3.10.0-229.11.1.el7.x86_64
    2 SourceLine(s) are available.
    SourceLine 0: interface SourceDataLine supporting 512 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
    SourceLine 1: interface Clip supporting 512 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
    1 TargetLine(s) are available.
    TargetLine 0: interface TargetDataLine supporting 512 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
Mixer 1: PCH [plughw:0,0], version 3.10.0-229.11.1.el7.x86_64
    2 SourceLine(s) are available.
    SourceLine 0: interface SourceDataLine supporting 24 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
    SourceLine 1: interface Clip supporting 24 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
    1 TargetLine(s) are available.
    TargetLine 0: interface TargetDataLine supporting 24 audio formats,
    and buffers of at least 32 bytes
```

```
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
Mixer 2: PCH [plughw:0,2], version 3.10.0-229.11.1.el7.x86_64
    1 TargetLine(s) are available.
    TargetLine 0: interface TargetDataLine supporting 24 audio formats,
    and buffers of at least 32 bytes
        BufferSize: 88200
        Format: PCM_SIGNED 44100.0 Hz, 16 bit, stereo, 4 bytes/frame, little-endian
...
```

各オーディオデバイスは，仮想マシンからはミキサー (Mixer) オブジェクトとして参照される．基本的には 1 つのデバイスに 1 つのミキサーオブジェクトが対応付けられるが，1 つのデバイスに複数のミキサーオブジェクトが対応付けられることもある．ミキサーの中には音声再生用ライン (SourceDataLine) や音声録音用ライン (TargetDataLine) を開くことができるものがある．音声の再生や録音にはこれらのラインを持つミキサーを明示的に指定することが望ましい．

付録 B　サンプルコード集

ソースコード 5　波形のプロット PlotWaveformCLI.java

```
1  import java.lang.invoke.MethodHandles;
2  import java.io.File;
3  import java.util.Arrays;
4  import java.util.Optional;
5  import java.util.stream.IntStream;
6  import java.util.stream.Collectors;
7  import javax.sound.sampled.AudioSystem;
8  import javax.sound.sampled.AudioFormat;
9  import javax.sound.sampled.AudioInputStream;
10 import javax.imageio.ImageIO;
11
12 import javafx.application.Application;
13 import javafx.application.Platform;
14 import javafx.stage.Stage;
15 import javafx.scene.Scene;
16 import javafx.scene.chart.XYChart;
17 import javafx.scene.chart.LineChart;
18 import javafx.scene.chart.NumberAxis;
19 import javafx.scene.image.WritableImage;
20 import javafx.collections.ObservableList;
21 import javafx.collections.FXCollections;
22 import javafx.embed.swing.SwingFXUtils;
23
24 import org.apache.commons.cli.CommandLine;
25 import org.apache.commons.cli.DefaultParser;
26 import org.apache.commons.cli.Options;
27 import org.apache.commons.cli.HelpFormatter;
28
29 import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
30
31 import java.io.IOException;
32 import javax.sound.sampled.UnsupportedAudioFileException;
33 import org.apache.commons.cli.ParseException;
```

```
34
35  public final class PlotWaveformCLI extends Application {
36
37    private static final Options options = new Options();
38    private static final String helpMessage =
39      MethodHandles.lookup().lookupClass().getName() + " [OPTIONS] <WAVFILE>";
40
41    static {
42      /* コマンドラインオプション定義 */
43      options.addOption("h", "help", false, "Display this help and exit");
44      options.addOption("o", "outfile", true,
45                        "Output image file (Default: " +
46                        MethodHandles.lookup().lookupClass().getSimpleName() +
47                        "." + Le4MusicUtils.outputImageExt + ")");
48      options.addOption("a", "amp-bounds", true,
49                        "Upper(+) and lower(-) bounds in the amplitude direction " +
50                        "(Default: " + Le4MusicUtils.waveformAmplitudeBounds + ")");
51    }
52
53    @Override
54    public final void start(final Stage primaryStage)
55      throws IOException,
56             UnsupportedAudioFileException,
57             ParseException {
58      /* コマンドライン引数処理 */
59      final String[] args = getParameters().getRaw().toArray(new String[0]);
60      final CommandLine cmd = new DefaultParser().parse(options, args);
61      if (cmd.hasOption("help")) {
62        new HelpFormatter().printHelp(helpMessage, options);
63        Platform.exit();
64        return;
65      }
66      final String[] pargs = cmd.getArgs();
67      if (pargs.length < 1) {
68        System.out.println("WAVFILE is not given.");
69        new HelpFormatter().printHelp(helpMessage, options);
70        Platform.exit();
71        return;
72      }
73      final File wavFile = new File(pargs[0]);
74
75      /* WAVファイル読み込み */
76      final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
77      final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
78      final AudioFormat format = stream.getFormat();
79      final double sampleRate = format.getSampleRate();
80      stream.close();
81
82      /* データ系列を作成 */
83      final ObservableList<XYChart.Data<Number, Number>> data =
84        IntStream.range(0, waveform.length)
85          .mapToObj(i -> new XYChart.Data<Number, Number>(i / sampleRate, waveform[i]))
86          .collect(Collectors.toCollection(FXCollections::observableArrayList));
87
88      /* データ系列に名前をつける */
89      final XYChart.Series<Number, Number> series =
90        new XYChart.Series<>("Waveform", data);
```

```
91
92      /* X軸を作成 */
93      final double duration = (waveform.length - 1) / sampleRate;
94      final NumberAxis xAxis = new NumberAxis(
95        /* axisLabel  = */ "Time (seconds)",
96        /* lowerBound = */ 0.0,
97        /* upperBound = */ duration,
98        /* tickUnit   = */ Le4MusicUtils.autoTickUnit(duration)
99      );
100     xAxis.setAnimated(false);
101
102     /* Y軸を作成 */
103     final double ampBounds =
104       Optional.ofNullable(cmd.getOptionValue("amp-bounds"))
105         .map(Double::parseDouble)
106         .orElse(Le4MusicUtils.waveformAmplitudeBounds);
107     final NumberAxis yAxis = new NumberAxis(
108       /* axisLabel  = */ "Amplitude",
109       /* lowerBound = */ -ampBounds,
110       /* upperBound = */ +ampBounds,
111       /* tickUnit   = */ Le4MusicUtils.autoTickUnit(ampBounds * 2.0)
112     );
113     yAxis.setAnimated(false);
114
115     /* チャートを作成 */
116     final LineChart<Number, Number> chart = new LineChart<>(xAxis, yAxis);
117     chart.setTitle("Waveform");
118     chart.setCreateSymbols(false);
119     chart.setLegendVisible(false);
120     chart.getData().add(series);
121
122     /* グラフ描画 */
123     final Scene scene = new Scene(chart, 800, 600);
124     scene.getStylesheets().add("src/le4music.css");
125
126     /* ウインドウ表示 */
127     primaryStage.setScene(scene);
128     primaryStage.setTitle(getClass().getName());
129     primaryStage.show();
130
131     /* チャートを画像ファイルへ出力 */
132     Platform.runLater(() -> {
133       final String[] name_ext = Le4MusicUtils.getFilenameWithImageExt(
134         Optional.ofNullable(cmd.getOptionValue("outfile")),
135         getClass().getSimpleName()
136       );
137       final WritableImage image = scene.snapshot(null);
138       try {
139         ImageIO.write(SwingFXUtils.fromFXImage(image, null),
140                       name_ext[1], new File(name_ext[0] + "." + name_ext[1]));
141       } catch (IOException e) { e.printStackTrace(); }
142     });
143   }
144
145 }
```

ソースコード 6　スペクトルのプロット PlotSpectrumCLI.java

```
1  import java.lang.invoke.MethodHandles;
2  import java.io.File;
3  import java.util.Arrays;
4  import java.util.Optional;
5  import java.util.stream.IntStream;
6  import java.util.stream.Collectors;
7  import javax.sound.sampled.AudioSystem;
8  import javax.sound.sampled.AudioFormat;
9  import javax.sound.sampled.AudioInputStream;
10 import javax.imageio.ImageIO;
11
12 import javafx.application.Application;
13 import javafx.application.Platform;
14 import javafx.stage.Stage;
15 import javafx.scene.Scene;
16 import javafx.scene.chart.XYChart;
17 import javafx.scene.chart.LineChart;
18 import javafx.scene.chart.NumberAxis;
19 import javafx.scene.image.WritableImage;
20 import javafx.collections.ObservableList;
21 import javafx.collections.FXCollections;
22 import javafx.embed.swing.SwingFXUtils;
23
24 import org.apache.commons.cli.CommandLine;
25 import org.apache.commons.cli.DefaultParser;
26 import org.apache.commons.cli.Options;
27 import org.apache.commons.cli.HelpFormatter;
28
29 import org.apache.commons.math3.complex.Complex;
30
31 import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
32
33 import java.io.IOException;
34 import javax.sound.sampled.UnsupportedAudioFileException;
35 import org.apache.commons.cli.ParseException;
36
37 public final class PlotSpectrumCLI extends Application {
38
39   private static final Options options = new Options();
40   private static final String helpMessage =
41     MethodHandles.lookup().lookupClass().getName() + " [OPTIONS] <WAVFILE>";
42
43   static {
44     /* コマンドラインオプション定義 */
45     options.addOption("h", "help", false, "Display this help and exit");
46     options.addOption("o", "outfile", true,
47                       "Output image file (Default: " +
48                       MethodHandles.lookup().lookupClass().getSimpleName() +
49                       "." + Le4MusicUtils.outputImageExt + ")");
50     options.addOption(null, "amp-lo", true,
51                       "Lower bound of amplitude [dB] (Default: " +
52                       Le4MusicUtils.spectrumAmplitudeLowerBound + ")");
53     options.addOption(null, "amp-up", true,
54                       "Upper bound of amplitude [dB] (Default: " +
55                       Le4MusicUtils.spectrumAmplitudeUpperBound + ")");
56     options.addOption(null, "freq-lo", true,
```

```
57                      "Lower bound of frequency [Hz] (Default: 0.0)");
58    options.addOption(null, "freq-up", true,
59                      "Upper bound of frequency [Hz] (Default: Nyquist)");
60  }
61
62  @Override public final void start(final Stage primaryStage)
63    throws IOException,
64           UnsupportedAudioFileException,
65           ParseException {
66    /* コマンドライン引数処理 */
67    final String[] args = getParameters().getRaw().toArray(new String[0]);
68    final CommandLine cmd = new DefaultParser().parse(options, args);
69    if (cmd.hasOption("help")) {
70      new HelpFormatter().printHelp(helpMessage, options);
71      Platform.exit();
72      return;
73    }
74    final String[] pargs = cmd.getArgs();
75    if (pargs.length < 1) {
76      System.out.println("WAVFILE is not given.");
77      new HelpFormatter().printHelp(helpMessage, options);
78      Platform.exit();
79      return;
80    }
81    final File wavFile = new File(pargs[0]);
82
83    /* WAVファイル読み込み */
84    final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
85    final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
86    final AudioFormat format = stream.getFormat();
87    final double sampleRate = format.getSampleRate();
88    final double nyquist = sampleRate * 0.5;
89    stream.close();
90
91    /* fftSize = 2^p >= waveform.length を満たす fftSize を求める
92     * 2^p はシフト演算で求める */
93    final int fftSize = 1 << Le4MusicUtils.nextPow2(waveform.length);
94    final int fftSize2 = (fftSize >> 1) + 1;
95    /* 信号の長さを fftSize に伸ばし，長さが足りない部分は0で埋める．
96     * 振幅を信号長で正規化する． */
97    final double[] src =
98      Arrays.stream(Arrays.copyOf(waveform, fftSize))
99            .map(w -> w / waveform.length)
100           .toArray();
101   /* 高速フーリエ変換を行う */
102   final Complex[] spectrum = Le4MusicUtils.rfft(src);
103
104   /* 対数振幅スペクトルを求める */
105   final double[] specLog =
106     Arrays.stream(spectrum)
107           .mapToDouble(c -> 20.0 * Math.log10(c.abs()))
108           .toArray();
109
110   /* スペクトル配列の各要素に対応する周波数を求める．
111    * 以下を満たすように線型に
112    * freqs[0] = 0Hz
113    * freqs[fftSize2 - 1] = sampleRate / 2 (= Nyquist周波数) */
```

```
114     final double[] freqs =
115       IntStream.range(0, fftSize2)
116                .mapToDouble(i -> i * sampleRate / fftSize)
117                .toArray();
118
119     /* データ系列を作成 */
120     final ObservableList<XYChart.Data<Number, Number>> data =
121       IntStream.range(0, fftSize2)
122         .mapToObj(i -> new XYChart.Data<Number, Number>(freqs[i], specLog[i]))
123         .collect(Collectors.toCollection(FXCollections::observableArrayList));
124
125     /* データ系列に名前をつける */
126     final XYChart.Series<Number, Number> series =
127       new XYChart.Series<>("spectrum", data);
128
129     /* X軸を作成 */
130     final double freqLowerBound =
131       Optional.ofNullable(cmd.getOptionValue("freq-lo"))
132         .map(Double::parseDouble)
133         .orElse(0.0);
134     if (freqLowerBound < 0.0)
135       throw new IllegalArgumentException(
136         "freq-lo must be non-negative: " + freqLowerBound
137       );
138     final double freqUpperBound =
139       Optional.ofNullable(cmd.getOptionValue("freq-up"))
140         .map(Double::parseDouble)
141         .orElse(nyquist);
142     if (freqUpperBound <= freqLowerBound)
143       throw new IllegalArgumentException(
144         "freq-up must be larger than freq-lo: " +
145         "freq-lo = " + freqLowerBound + ", freq-up = " + freqUpperBound
146       );
147     final NumberAxis xAxis = new NumberAxis(
148       /* axisLabel  = */ "Frequency (Hz)",
149       /* lowerBound = */ freqLowerBound,
150       /* upperBound = */ freqUpperBound,
151       /* tickUnit   = */ Le4MusicUtils.autoTickUnit(freqUpperBound - freqLowerBound)
152     );
153     xAxis.setAnimated(false);
154
155     /* Y軸を作成 */
156     final double ampLowerBound =
157       Optional.ofNullable(cmd.getOptionValue("amp-lo"))
158         .map(Double::parseDouble)
159         .orElse(Le4MusicUtils.spectrumAmplitudeLowerBound);
160     final double ampUpperBound =
161       Optional.ofNullable(cmd.getOptionValue("amp-up"))
162         .map(Double::parseDouble)
163         .orElse(Le4MusicUtils.spectrumAmplitudeUpperBound);
164     if (ampUpperBound <= ampLowerBound)
165       throw new IllegalArgumentException(
166         "amp-up must be larger than amp-lo: " +
167         "amp-lo = " + ampLowerBound + ", amp-up = " + ampUpperBound
168       );
169     final NumberAxis yAxis = new NumberAxis(
170       /* axisLabel  = */ "Amplitude (dB)",
```

```
171         /* lowerBound = */ ampLowerBound,
172         /* upperBound = */ ampUpperBound,
173         /* tickUnit   = */ Le4MusicUtils.autoTickUnit(ampUpperBound - ampLowerBound)
174       );
175       yAxis.setAnimated(false);
176
177       /* チャートを作成 */
178       final LineChart<Number, Number> chart =
179         new LineChart<>(xAxis, yAxis);
180       chart.setTitle("Spectrum");
181       chart.setCreateSymbols(false);
182       chart.setLegendVisible(false);
183       chart.getData().add(series);
184
185       /* グラフ描画 */
186       final Scene scene = new Scene(chart, 800, 600);
187       scene.getStylesheets().add("src/le4music.css");
188
189       /* ウインドウ表示 */
190       primaryStage.setScene(scene);
191       primaryStage.setTitle(getClass().getName());
192       primaryStage.show();
193
194       /* チャートを画像ファイルへ出力 */
195       Platform.runLater(() -> {
196         final String[] name_ext = Le4MusicUtils.getFilenameWithImageExt(
197           Optional.ofNullable(cmd.getOptionValue("outfile")),
198           getClass().getSimpleName()
199         );
200         final WritableImage image = scene.snapshot(null);
201         try {
202           ImageIO.write(SwingFXUtils.fromFXImage(image, null),
203                         name_ext[1], new File(name_ext[0] + "." + name_ext[1]));
204         } catch (IOException e) { e.printStackTrace(); }
205       });
206     }
207
208 }
```

ソースコード 7　スペクトログラムのプロット PlotSpectrogramCLI.java

```
1  import java.lang.invoke.MethodHandles;
2  import java.io.File;
3  import java.util.Arrays;
4  import java.util.Optional;
5  import java.util.stream.Stream;
6  import java.util.stream.IntStream;
7  import javax.sound.sampled.AudioSystem;
8  import javax.sound.sampled.AudioFormat;
9  import javax.sound.sampled.AudioInputStream;
10 import javax.imageio.ImageIO;
11
12 import javafx.application.Application;
13 import javafx.application.Platform;
14 import javafx.stage.Stage;
15 import javafx.scene.Scene;
16 import javafx.scene.chart.NumberAxis;
```

```
17 import javafx.scene.image.WritableImage;
18 import javafx.embed.swing.SwingFXUtils;
19
20 import org.apache.commons.cli.CommandLine;
21 import org.apache.commons.cli.DefaultParser;
22 import org.apache.commons.cli.Options;
23 import org.apache.commons.cli.HelpFormatter;
24
25 import org.apache.commons.math3.complex.Complex;
26 import org.apache.commons.math3.util.MathArrays;
27
28 import jp.ac.kyoto_u.kuis.le4music.Le4MusicUtils;
29 import jp.ac.kyoto_u.kuis.le4music.LineChartWithSpectrogram;
30
31 import java.io.IOException;
32 import javax.sound.sampled.UnsupportedAudioFileException;
33 import org.apache.commons.cli.ParseException;
34
35 public final class PlotSpectrogramCLI extends Application {
36
37   private static final Options options = new Options();
38   private static final String helpMessage =
39     MethodHandles.lookup().lookupClass().getName() + " [OPTIONS] <WAVFILE>";
40
41   static {
42     /* コマンドラインオプション定義 */
43     options.addOption("h", "help", false, "Display this help and exit");
44     options.addOption("o", "outfile", true,
45                       "Output image file (Default: " +
46                       MethodHandles.lookup().lookupClass().getSimpleName() +
47                       "." + Le4MusicUtils.outputImageExt + ")");
48     options.addOption("f", "frame", true,
49                       "Duration of frame [seconds] (Default: " +
50                       Le4MusicUtils.frameDuration + ")");
51     options.addOption("s", "shift", true,
52                       "Duration of shift [seconds] (Default: frame/8)");
53   }
54
55   @Override public final void start(final Stage primaryStage)
56     throws IOException,
57            UnsupportedAudioFileException,
58            ParseException {
59     /* コマンドライン引数処理 */
60     final String[] args = getParameters().getRaw().toArray(new String[0]);
61     final CommandLine cmd = new DefaultParser().parse(options, args);
62     if (cmd.hasOption("help")) {
63       new HelpFormatter().printHelp(helpMessage, options);
64       Platform.exit();
65       return;
66     }
67     final String[] pargs = cmd.getArgs();
68     if (pargs.length < 1) {
69       System.out.println("WAVFILE is not given.");
70       new HelpFormatter().printHelp(helpMessage, options);
71       Platform.exit();
72       return;
73     }
```

```
74      final File wavFile = new File(pargs[0]);
75
76      /* WAVファイル読み込み */
77      final AudioInputStream stream = AudioSystem.getAudioInputStream(wavFile);
78      final double[] waveform = Le4MusicUtils.readWaveformMonaural(stream);
79      final AudioFormat format = stream.getFormat();
80      final double sampleRate = format.getSampleRate();
81      final double nyquist = sampleRate * 0.5;
82      stream.close();
83
84      /* 窓関数とFFTのサンプル数 */
85      final double frameDuration =
86        Optional.ofNullable(cmd.getOptionValue("frame"))
87          .map(Double::parseDouble)
88          .orElse(Le4MusicUtils.frameDuration);
89      final int frameSize = (int)Math.round(frameDuration * sampleRate);
90      final int fftSize = 1 << Le4MusicUtils.nextPow2(frameSize);
91      final int fftSize2 = (fftSize >> 1) + 1;
92
93      /* シフトのサンプル数 */
94      final double shiftDuration =
95        Optional.ofNullable(cmd.getOptionValue("shift"))
96          .map(Double::parseDouble)
97          .orElse(Le4MusicUtils.frameDuration / 8);
98      final int shiftSize = (int)Math.round(shiftDuration * sampleRate);
99
100     /* 窓関数を求め，それを正規化する */
101     final double[] window = MathArrays.normalizeArray(
102       Arrays.copyOf(Le4MusicUtils.hanning(frameSize), fftSize), 1.0
103     );
104
105     /* 短時間フーリエ変換本体 */
106     final Stream<Complex[]> spectrogram =
107       Le4MusicUtils.sliding(waveform, window, shiftSize)
108                    .map(frame -> Le4MusicUtils.rfft(frame));
109
110     /* 複素スペクトログラムを対数振幅スペクトログラムに */
111     final double[][] specLog =
112       spectrogram.map(sp -> Arrays.stream(sp)
113                                   .mapToDouble(c -> 20.0 * Math.log10(c.abs()))
114                                   .toArray())
115                  .toArray(n -> new double[n][]);
116
117     /* 参考：フレーム数と各フレーム先頭位置の時刻 */
118     final double[] times =
119       IntStream.range(0, specLog.length)
120                .mapToDouble(i -> i * shiftDuration)
121                .toArray();
122
123     /* 参考：各フーリエ変換係数に対応する周波数 */
124     final double[] freqs =
125       IntStream.range(0, fftSize2)
126                .mapToDouble(i -> i * sampleRate / fftSize)
127                .toArray();
128
129     /* X軸を作成 */
130     final double duration = (specLog.length - 1) * shiftDuration;
```

```
131     final NumberAxis xAxis = new NumberAxis(
132       /* axisLabel  = */ "Time (seconds)",
133       /* lowerBound = */ 0.0,
134       /* upperBound = */ duration,
135       /* tickUnit   = */ Le4MusicUtils.autoTickUnit(duration)
136     );
137     xAxis.setAnimated(false);
138
139     /* Y軸を作成 */
140     final NumberAxis yAxis = new NumberAxis(
141       /* axisLabel  = */ "Frequency (Hz)",
142       /* lowerBound = */ 0.0,
143       /* upperBound = */ nyquist,
144       /* tickUnit   = */ Le4MusicUtils.autoTickUnit(nyquist)
145     );
146     yAxis.setAnimated(false);
147
148     /* チャートを作成 */
149     final LineChartWithSpectrogram<Number, Number> chart =
150       new LineChartWithSpectrogram<>(xAxis, yAxis);
151     chart.setParameters(specLog.length, fftSize2, nyquist);
152     chart.setTitle("Spectrogram");
153     Arrays.stream(specLog).forEach(chart::addSpecLog);
154     chart.setCreateSymbols(false);
155     chart.setLegendVisible(false);
156
157     /* グラフ描画 */
158     final Scene scene = new Scene(chart, 800, 600);
159     scene.getStylesheets().add("src/le4music.css");
160
161     /* ウインドウ表示 */
162     primaryStage.setScene(scene);
163     primaryStage.setTitle(getClass().getName());
164     primaryStage.show();
165
166     /* チャートを画像ファイルへ出力 */
167     Platform.runLater(() -> {
168       final String[] name_ext = Le4MusicUtils.getFilenameWithImageExt(
169         Optional.ofNullable(cmd.getOptionValue("outfile")),
170         getClass().getSimpleName()
171       );
172       final WritableImage image = scene.snapshot(null);
173       try {
174         ImageIO.write(SwingFXUtils.fromFXImage(image, null),
175                       name_ext[1], new File(name_ext[0] + "." + name_ext[1]));
176       } catch (IOException e) { e.printStackTrace(); }
177     });
178   }
179
180 }
```

付録 C　SoX cookbook

音響信号処理プログラム群 SoX[*26] の代表的な使用方法を解説する.

- 全チャネル混合によるモノラル化

```
$ sox input.wav output.wav channels 1
```

- リサンプリングによるサンプリング周波数変換

```
$ sox input.wav output.wav rate 16000
```

- トリミング（切り出し）. 2.1 秒から 1.5 秒間（3.6 秒まで）

```
$ sox input.wav output.wav trim 2.1 1.5
```

[*26] http://sox.sourceforge.net/