

音の取り扱いに関するメモ (C, Java, Mathematica)

桂田 祐史

2008年3月6,7日, 2010年3月, 2013年10月

<http://nalab.mind.meiji.ac.jp/~mk/labo/text/sound/>
私が作ったプログラムは自由に使ってください。無保証ですけど。

2009年度卒研でも「音」を研究テーマに取り上げた学生がいたので、久しぶりにここに手を入れる予定。

1 目標と計画

(準備中)

買物計画 PCMレコーダー (パソコンを持って起動を待つてサウンドレコーダーは非能率的、やはりデンスケでしょうか, PCM-D50, PCM-M10), 手頃なスピーカー&ヘッドホン (Let's note のスピーカーは、時々本当にとんでもない音が出ます¹), 色々な音データ

常微分作用素の固有値問題については、横山 [1](他に桂田 [2] を見よ) があるが、いつかもう一度きちんとチャレンジしたい。

いつか、フレッチャー・ロッシング [3] を真面目に読んでチャレンジする学生が出ないかな、と思っている。

城都 [4], [5] くらいの内容は数学的に整理しておきたいと思っている。

音声データについては、2003年度卒研生の松山周五郎君が録音して集めてくれたデータがあるが(<http://nalab.mind.meiji.ac.jp/~mk/labo/report/open/matsuyama-wave-2014/> で公開中) があるが、もっと色々なものを収集したい。

2 Windows のサウンドレコーダー

(準備中 — と言っているうちに Windows から疎遠になりつつある…多分書くことはないような気がしてきた。)

¹音を合成するには、信号の和でなくて積を求めるという (情けない) 勘違いを学生がしてしまったくらい。

3 Mathematica でサウンドを扱う

実際の使用例としては、2003 年度卒研生の松山周五郎君の卒業研究レポート [6] が参考になる。

3.1 注意事項

Mathematica では「振幅」という言葉を通常の物理の本で採用されている意味 (もっとも大きく振れた幅、変位の絶対値の最大値、ということで、 $f(t) = A \sin(2\pi f(t - t_0))$ の場合、 $|A|$ が振幅である) とは異なる意味で用いている (そのことをはっきりと断った上でのことである)。

Mathematica のマニュアルに曰く

文献によっては、振幅は音のピーク信号強度と定義するものもあるが、本書では、常に、時間の関数である瞬間的な信号強度とする。

3.2 Play []

時刻 t (単位は秒) における音圧が $\sin 2\pi ft$ である音は、周波数が f [Hz] の音である。Play [] を使うと、この音が鳴らせる。使い方は Plot [] に似ている。

440 Hz の音を鳴らす

```
Play[Sin[2 Pi 440 t], {t, 0, 1}]
```

なお、デフォルトのサンプリング周波数 (SampleRate) は 8[kHz] である。音楽 CD 品質のサンプリング周波数 (44.1[kHz]) にするには、

音楽 CD 音質で 440Hz の音を鳴らす

```
Play[Sin[2 Pi 440 t], {t, 0, 1}, SampleRate->44100]
```

音を混ぜるのは、単に和を作るだけである。

440Hz, 880Hz, 1320Hz の音を混ぜる

```
ra[a_, b_, c_] := Play[a * Sin[2 Pi 440 t] + b * Sin[2 Pi 440 2 t] + c * Sin[2 Pi 440 3 t], {t, 0, 1}, SampleRate->44100]
```

ステレオも出来るようである。

```
Play[{Sin[2 Pi 440 t], Sin[2 Pi 880 t]}, {t, 0, 1}, SampleRate->44100]
```

うなりの例

```
Play[Sin[2 Pi 440 t] + Sin[2 Pi (440 - 0.1 t) t], {t, 0, 100}]
```

Mathematica のマニュアルの例 (何だこれは)

```
Play[Sin[700 t + 25 t Sin[350 t]], {t, 0, 4}]
```

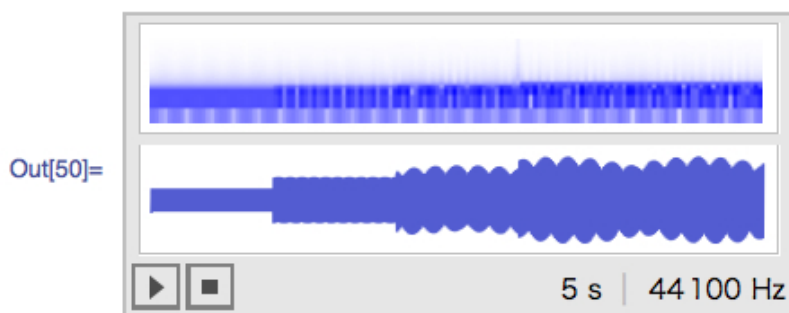
和音, ついでに WAVE ファイル domisodo.wav²を作成。

```
c = 2.0^(1/12);  
do = 261*2*Pi; mi = c^4 do; so = c^7 do; do2 = 2*do;  
snd=Play[Sin[do t]+Boole[t>1]Sin[mi t]+Boole[t>2]Sin[so t]+Boole[t>3]Sin[do2 t],  
{t,0,5}, SampleRate->44100,PlayRange->{-5,5}]  
Export["domisodo.wav",snd]
```

```
In[48]:= c = 2.0^(1/12);
```

```
In[49]:= do = 261 * 2 * Pi; mi = c^4 do; so = c^7 do;  
do2 = 2 * do;
```

```
In[50]:= snd = Play[Sin[do t] + Boole[t > 1] Sin[mi t] +  
Boole[t > 2] Sin[so t] + Boole[t > 3] Sin[do2 t],  
{t, 0, 5}, SampleRate -> 44 100, PlayRange -> {-5, 5}]
```



```
In[51]:= Export["domisodo.wav", snd]
```

```
Out[51]= domisodo.wav
```

(PlayRange->{} を入れないと音が割れてしまう。この辺は良く分らない。Export[] が気が利かないのかも。)

3.3 ListPlay[]

ListPlay[] を用いると、数値リストを音に変換出来る。

Play[] の例にあげた Play[Sin 2 Pi 440 t],{t,0,1}] と同様のことを、自分でサンプリングして行うには、

440Hz の音を鳴らす

```
s=Table[Sin[2 Pi 440 t],{t,0,1.0,1.0/8000}];  
ListPlay[s]
```

(ListPlay[] のデフォルトのサンプリング・レートが 8 kHz なので、1/8000 という間隔でサンプリングしているわけである。)

音楽 CD 音質で 440Hz の音を鳴らす

```
mysamplerate = 44100  
s=Table[Sin[2 Pi 440 t],{t,0,1.0,1.0/mysamplerate}];  
ListPlay[s, SampleRate->mysamplerate]
```

3.4 Import[] によるサウンド・ファイルの読み込み

Import[] で画像ファイルやサウンド・ファイルの入力が可能である。

例えば WAVE ファイルを読み込んで、サウンド・データとして変数に格納できる。

```
SetDirectory["C:/Document and Settings/mk/デスクトップ"]  
snd = Import["piano.wav"]
```

サウンドである snd は、Show[snd] などで再生できる (単に snd を評価しても良い)。

- snd はリストであるが、Length[snd] は 1 である (snd[[2]] は存在しない)。
- snd[[1]] はリストであるが、Length[snd[[1]]] は 2 である。
- snd[[1,2]] はサンプリング周波数 (SampleRate) である。
- Length[snd[[1,1]]] はチャンネル数に等しい。1 ならばモノラルで、2 ならばステレオである。
- snd[[1,1,1]] は数値のリストである。snd[[1,1,2]] も存在するならば (ステレオならば) 数値のリスト。

図示すると

```
snd = {  
  {  
    {l1,l2,...,lN},      <- snd[[1,1,1]]  
    {r1,r2,...,rN}      <- snd[[1,1,2]]  
  },  
  SampleRate           <- snd[[1,2]]  
}
```

残念ながら単なるリストでないので、

これは出来ない

```
{{leftChannel,rightChannel},samplingRate} = snd;
```

のような代入はさせてもらえないが、例えば

```
leftChannel=snd[[1,1,1]]; rightChannel=snd[[1,1,2]]; samplingRate=snd[[1,2]]
```

あるいは

```
{leftChannel,rightChannel},samplingRate}={snd[[1,1]],snd[[1,2]]}
```

とすると、左チャンネルの数値データ leftChannel, 右チャンネルの数値データ rightChannel, サンプル周波数 samplingRate が得られる。

ListPlay[] で再生してみる

```
snd=Import["WAVE ファイル名"];  
ListPlay[snd[[1,1,1]],SampleRate->snd[[1,2]]]  
あるいは  
ListPlay[snd[[1,1]],SampleRate->snd[[1,2]]]
```

最近の Mathematica では、項目 (要素名) を指定した読み込みが可能である。

どういう項目があるか

```
Import["piano.wav","Elements"]
```

とすると

```
{"AudioChannels", "AudioEncoding", "Data", "SampledSoundList", "SampleRate", "Sound"}
```

という結果を返す。

チャンネルの数

```
numofchannels=Import["piano.wav","AudioChannels"]
```

これは多分 numofchannels=Length[Import["piano.wav"][[1,1]]] と同じ。

1 とか 2 とか、チャンネル数を返す。つまりステレオならば 2, モノラルならば 1 である。

エンコードの方法

```
Import["piano.wav","AudioEncoding"]
```

UnsignedInteger8 のような形式を返す。まあ、これは Mathematica が抽象化してくれていて、あまり気にならないところ (C でプログラムを書く時は面倒だ)。

サンプリング周波数

```
samplingrate=Import["piano.wav","SampleRate"]
```

これは多分 `samplingrate=Import["piano.wav"][[1,2]]` と同じ。

44010 のようなサンプリング周波数を返す。

```
snd = Import["piano.wav","Sound"]
```

これは多分 `snd = Import["piano.wav"]` と同じ。

```
data=Import["piano.wav","Data"];
```

これは多分 `data=Import["piano.wav"][[1,1]]`; と同じ。

```
s=Import["piano.wav","SampledSoundList"];
```

これは多分 `data=Import["piano.wav"][[1]]`; と同じ。

「同じ」と言ったものは、`Equal[]` で確認できる。

あまり使用経験はないが、ほぼ裏返しの `Export[]` が使える。

3.5 Fourier[] による離散 Fourier 変換

(考えてみたら、周期を一般の T とした式を書くべきだな…)

Mathematica には、離散フーリエ変換をするための `Fourier[]` がある。

周期 2π の周期関数 $u: \mathbf{R} \rightarrow \mathbf{C}$ があるとき、区間 $[0, 2\pi]$ の N 等分点

$$(1) \quad t_j = \frac{2\pi(j-1)}{N} \quad (j = 1, 2, \dots, N)$$

における u の値

$$(2) \quad u_j := u(t_j) \quad (j = 1, 2, \dots, N)$$

が得られたとする。このとき、 u の複素 Fourier 係数

$$(3) \quad c_k := \frac{1}{2\pi} \int_0^{2\pi} u(t) e^{-ikt} dt \quad (k \in \mathbf{Z})$$

を台形則で近似したものを C_k とすると、

$$(4) \quad C_k = \frac{1}{2\pi} \sum_{j=1}^N u(t_j) e^{-ikt_j} \cdot \frac{2\pi}{N} = \frac{1}{N} \sum_{j=1}^N u_j \exp\left(-\frac{2\pi i(j-1)k}{N}\right) \quad (k \in \mathbf{Z}).$$

`InverseFourier[]` という逆変換が用意されている。

簡単に証明できる良く知られた事実

- 普通の Fourier 係数

$$(5) \quad a_k := \frac{1}{2\pi} \int_0^{2\pi} u(t) \cos kt \, dt, \quad b_k := \frac{1}{2\pi} \int_0^{2\pi} u(t) \sin kt \, dt \quad (k \in \mathbf{Z}, k \geq 0)$$

と複素 Fourier 係数との間に、

$$(6) \quad c_0 = \frac{a_0}{2}, \quad c_k = \begin{cases} \frac{a_k - ib_k}{2} & (k \geq 1) \\ \frac{a_{-k} + ib_{-k}}{2} & (k \leq -1) \end{cases}$$

という関係がある ($\{a_k\}, \{b_k\}$ から $\{c_k\}$ を求めるのに使える)。言い替えると (こちらは $\{c_k\}$ から $\{a_j\}, \{b_k\}$ を求めるのに使える)

$$(7) \quad a_0 = 2c_0, \quad a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}) \quad (k \in \mathbf{N}).$$

特に u が実数値関数の場合、 $\{a_k\}$ と $\{b_k\}$ は実数列であるから、 $c_{-k} = \overline{c_k}$ が成り立ち (特に c_0 は実数)、

$$(8) \quad a_0 = 2c_0 = 2 \operatorname{Re} c_0, \quad a_k = 2 \operatorname{Re} c_k, \quad b_k = -2 \operatorname{Im} c_k \quad (k \in \mathbf{N}).$$

- $\{C_k\}$ は周期 N の周期数列である: $C_{k+N} = C_k$ ($k \in \mathbf{Z}$).
- $|k| \ll N/2$ のとき、 C_k は c_k の“良い近似”である。 $0 \leq k \ll N/2$ のとき

$$(9) \quad c_k \simeq C_k, \quad c_{-k} \simeq C_{-k} = C_{N-k}.$$

(a_k, b_k の近似が必要な場合) $1 \leq k \ll N/2$ のとき、

$$(10) \quad a_0 \simeq 2C_0, \quad a_k \simeq C_k + C_{-k} = C_k + C_{N-k}, \quad b_k \simeq i(C_k - C_{-k}) = i(C_k - C_{N-k}).$$

Mathematica では、長さ N のリスト $u = \{u_1, \dots, u_N\}$ があるとき、

```
v=Fourier[u,FourierParameters->{a,b}]
```

とすると、

$$v_s := \frac{1}{N^{(1-a)/2}} \sum_{r=1}^N u_r \exp\left(\frac{2\pi i b(r-1)(s-1)}{N}\right) \quad (s = 1, 2, \dots, N)$$

で定まる v_1, \dots, v_N を並べたリスト v が得られる。`,FourierParameters->{a,b}` を省略して単に `v=Fourier[u]` とすると、 $(a,b) = (0,1)$ と指定したのと同じになる (いわゆる “default”)。

特に

```
v=Fourier[u,FourierParameters->{-1,-1}]
```

とすると、

$$v_s := \frac{1}{N} \sum_{r=1}^N u_r \exp\left(-\frac{2\pi i(r-1)(s-1)}{N}\right) = C_{s-1} \quad (s = 1, 2, \dots, N).$$

すなわち

$$v = \{C_0, \dots, C_{N-1}\}.$$

三角級数の係数を再生

```
a0 = 1
```

```
a = {1, 2, 3, 4, 5}
```

```
b = {6, 7, 8, 9, 10}
```

```
u[t_] := a0/2 + a . Table[Cos[n t], {n, Length[a]}] + b . Table[Sin[n t], {n, Length[b]}]
```

```
n = 12
```

```
ts = Table[2 Pi (j - 1)/n, {j, n}];
```

```
us = u[ts];
```

```
cs = Fourier[us, FourierParameters -> {-1, -1}]
```

```
A0 = 2 cs[[1]]
```

```
A = Table[cs[[k + 1]] + cs[[n - k + 1]], {k, 5}]
```

```
B = I Table[cs[[k + 1]] - cs[[n - k + 1]], {k, 5}]
```

3.6 解析例

2009年度卒業研究(山田祐二君)の例。アコースティック・ギターの5弦3フレットを抑えて弾いた音を録音したWAVEファイル(サンプリング周波数44.1kHz, 量子化ビット数16, ステレオという形式)の、62800個目の数値(ファイルの最初から62800/44100≒1.42秒後)から1秒分を離散Fourier変換した。129Hz, 258Hz, 388Hz, 776Hz, 646Hzにピークがある。


```

SetDirectory["c:/Documents and Settings/ユーザー名/デスクトップ/どこか"]

wavFourier[snd_,start_] :=
  Abs[Fourier[Take[snd[[1,1,1]],{start,start+snd[[1,2]]-1}]]]

plotSpectrum[spec_] :=
  ListPlot[spec,PlotJoined->True,PlotRange->{{1,1600},All}]

peakPositions[spec_,n_] :=
  Module[{pp,pps,halfspec},
    pps={}; halfspec=Take[spec,Floor[Length[spec]/2]];
    For[i=1,i<=n,i++,
      pp=Ordering[halfspec,-1][[1]];
      pps = Append[pps,pp];
      For[j=-6,j<=6,j++,halfspec[[pp+j]]=0]
    ];
    pps
  ]

peakBar[spec_,pps_] :=BarChart[spec[[Sort[pps]]],ChartLabels->Sort[pps]-1]

sounddata=Import["guitar1.wav"];
aft=wavFourier[sounddata,62800];
g1=plotSpectrum[aft]
pps=peakPositions[aft,5]
  結果は {130, 259, 389, 777, 647}

g2=peakBar[aft,pps]

  解析位置
For[s = 40000, s <= 80000, s += 2000,
  Print[peakPositions[wavFourier[Import["guitar1.wav"], s], 5]]]

```

プログラムは色々改良の余地がある。どこから解析するか指示する start の単位は秒にすべきかも。

3.7 その他 (新しめの機能?)

- Speak["Hello, how are you?"]

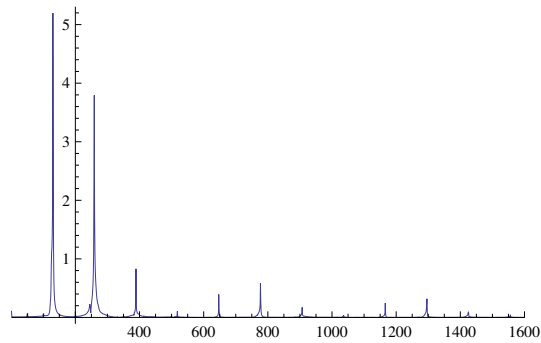


図 1: guitar1.wav のスペクトル, 129,258,388,776,646 にピーク

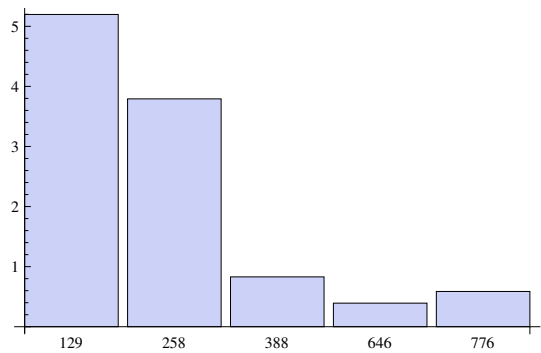


図 2: guitar1.wav のスペクトル 5つのピーク

- Sound []
- SampledSoundList []
- SoundNote [] — 楽音を出す (色々な楽器の音が出せる)
- EmitSound [] — 音を出す (最近の Mathematica は、再生ボタンを押すと音を発するよ
うなウィンドウを作るが、そうしないですぐに音を発する)
- SystemDialogInput["RecordSound"] — 図3を見れば一目瞭然

ピアノでドミソド, フルートで和音

```
EmitSound[
  Sound[{SoundNote["C4"], SoundNote["E4"], SoundNote["G4"], SoundNote["C5"],
    SoundNote[{"C4", "E4", "G4", "C5"}, 1, "Flute"]}]]
```



図 3: Mathematica の Sound Recorder

4 C で WAVE ファイルを扱う

4.1 参考にした情報

<http://homepage3.nifty.com/ryuz/programing/wavfmt.html> に書いてある情報を参考にして、WAVE ファイルの読み書きをするプログラムを C で書いた。

この時は気が付かなかったが、『WAV ファイルフォーマット』³ という WWW ページがあって、これが良さそう。

なお、C で FFT をするには、『Ooura's Mathematical Software Packages』⁴ にある大浦 FFT を使うのがお勧め。

4.2 拙作 readwave.c (WAVE →テキスト・ファイル)

<http://nalab.mind.meiji.ac.jp/~mk/program/sound/readwave.c>

やっつけの、リバースエンジニアリングの、あまり人に見せたくない…

ピアノの音を録音した piano.wav⁵ という WAVE ファイルの内容をテキスト・ファイルに変換してみる。

```
oyabun% gcc -o readwave readwave.c
oyabun% ./readwave piano.wav piano.txt
oyabun% ./readwave piano.wav > piano2.txt
```

³<http://www.kk.iij4u.or.jp/~kondo/wave/index.html>

⁴<http://www.kurims.kyoto-u.ac.jp/~ooura/index-j.html>

⁵<http://nalab.mind.meiji.ac.jp/~mk/labo/text/sound-files/piano.wav>

```

oyabun% ls -l piano.wav piano.txt piano2.txt
-rw-r--r--  1 mk      lab00   2814838  3月  6日  13:44 piano.txt
-rw-r--r--  1 mk      lab00   2828151  3月  6日  13:49 piano2.txt
-rw-r--r--  1 mk      lab00   1234592  1月 23日  20:39 piano.wav
oyabun% head piano.txt
#original file: piano.wav
#number of channels: 2
#sampling rate: 44100
#number of bits (per sample): 16
#number of samples: 307017
64 64
62 62
65 62
65 63
65 60

```

piano2.txt (少し冗長)

```

# RIFF データのサイズ=1234584
# fmt データのサイズ=18
# 01 00 02 00 44 ac 00 00 10 b1 02 00 04 00 10 00
# 00 00
# 非圧縮PCMです。
# ステレオです。
# サンプリング・レート (標本化周波数)=44100
# 1秒当りのバイト数=176400
# ブロック境界=4
# ビット数/サンプル=16
# 拡張情報サイズ=0
# PAD チャンクがあります。# PAD チャンクのサイズ=4042
# 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
(中略)
# 00 00 00 00 00 00 00 00 00 00 00 00
# fact チャンクがあります。# fact データのサイズ=4
# 49 af 04 00
# data データのサイズ=1228068
#original file: piano.wav
#number of channels: 2
#sampling rate: 44100
#number of bits (per sample): 16
#number of samples: 307017
64 64
62 62
(中略 --- 結構長い)
-114 -63
-111 -59
-112 -61
-109 -53
-108 -51

```

```

1  /*
2  * readwave.c --- 無圧縮 WAVE ファイルを読んでテキスト・ファイルにする
3  *               動作することはまったく保証しません
4  *
5  * version 2 (2003/12/21)
6  * version 3 (2005/7/4) "PAD " チャンクが存在を知りそれに対応
7  * version 4 (2013/10/26) #include 増やした。文字コードを UTF8 にした。
8  * version 5 (2014/8/12) FLLR チャンクが存在を知りそれに対応
9  *
10 * コンパイル: gcc -o readwave readwave.c
11 *
12 * 使い方:
13 * (1) ./readwave <WAVE ファイル名> <テキスト・ファイル名>
14 * (2) ./readwave <WAVE ファイル名>
15 *     標準出力に出力
16 * (3) ./readwave
17 *     標準入力から入力、標準出力に出力
18 *
19 * 入手:
20 *   http://www.math.meiji.ac.jp/~mk/program/
21 *
22 * 参考にした情報
23 *   URL: http://member.nifty.ne.jp/Ryuz/programing/wavefmt.html
24 *        -> http://homepage3.nifty.com/ryuz/programing/wavefmt.html
25 *
26 * 普段置いてある場所
27 *   ~/Sotsuken/2007/sound/readwave/
28 */
29
30 #include <stdio.h>
31 #include <stdlib.h>
32 #include <string.h>
33
34 const int verbose = 0;
35 const int debug = 0;
36 FILE *out;
37
38 void usage(char *prog)
39 {
40     fprintf(stderr, "usage (1): %s\n", prog);
41     fprintf(stderr, "usage (2): %s <wave-file-name>\n", prog);
42     fprintf(stderr, "usage (3): %s <wave-file-name> <text-file-name>\n", prog);
43     exit(0);
44 }
45
46 void setstring(unsigned char *buf, FILE *in, int n)
47 {
48     int i;
49     for (i = 0; i < n; i++)
50         buf[i] = getc(in);
51     buf[n] = 0;
52 }
53
54 int readint(FILE *in)
55 {
56     int c1, c2, c3, c4;

```

```

57     c1 = getc(in); c2 = getc(in); c3 = getc(in); c4 = getc(in);
58     return c1 + 256 * (c2 + 256 * (c3 + 256 * c4));
59 }
60
61 int checkstring(unsigned char *buf, char *key)
62 {
63     int i, n = strlen(key);
64     for (i = 0; i < n; i++)
65         if (buf[i] != key[i]) {
66             fprintf(stderr, "%s は \"%s\" と一致しない\n", buf, key);
67             exit(1);
68         }
69     return 0;
70 }
71
72 void dump(unsigned char *buf, int n)
73 {
74     int i;
75     printf("# ");
76     for (i = 0; i < n; i++) {
77         printf(" %02x", buf[i]);
78         if ((i + 1) % 16 == 0)
79             printf("\n# ");
80     }
81     printf("\n");
82 }
83
84 int read2byte(unsigned char *buf)
85 {
86     return buf[0] + 256 * buf[1];
87 }
88
89 int read1(FILE *in)
90 {
91     int c = getc(in);
92     if (debug) {
93         printf("%02x ", c);
94         fprintf(out, "%02x ", c);
95     }
96     return c;
97 }
98
99 int read2(FILE *in)
100 {
101     int c1, c2, c;
102     c1 = getc(in); c2 = getc(in);
103     if (debug) {
104         printf("%02x %02x ", c1, c2);
105         fprintf(out, "%02x %02x ", c1, c2);
106     }
107     c = c1 + (c2 << 8);
108     if (c >= 32768)
109         c -= 65536;
110     return c;
111 }
112

```

```

113 int read4byte(unsigned char *buf)
114 {
115     return buf[0] + 256 * (buf[1] + 256 * (buf[2] + 256 * buf[3]));
116 }
117
118 void analyze(unsigned char *buf,
119             int *fmt_tag, int *num_channel, int *sampling_rate,
120             int *bytes_per_seconds, int *block, int *num_bits,
121             int *extended_information_size)
122 {
123     *fmt_tag = read2byte(buf);
124     *num_channel = read2byte(buf + 2);
125     *sampling_rate = read4byte(buf + 4);
126     *bytes_per_seconds = read4byte(buf + 8);
127     *block = read2byte(buf + 12);
128     *num_bits = read2byte(buf + 14);
129     *extended_information_size = read2byte(buf + 16);
130 }
131
132 void readpcm(FILE *in, int size, int bits, int channel)
133 {
134     int i, c, right;
135     if (bits == 16) {
136         if (channel == 2) {
137             for (i = 0; i < size / 4; i++) {
138                 c = read2(in); right = read2(in);
139                 if (verbose) printf("%d %d\n", c, right);
140                 fprintf(out, "%d %d\n", c, right);
141             }
142         }
143         else {
144             for (i = 0; i < size / 2; i++) {
145                 c = read2(in);
146                 if (verbose) printf("%d\n", c);
147                 fprintf(out, "%d\n", c);
148             }
149         }
150     }
151     else {
152         if (channel == 2) {
153             for (i = 0; i < size / 2; i++) {
154                 c = read1(in); right = read1(in);
155                 if (verbose) printf("%d %d\n", c, right);
156                 fprintf(out, "%d %d\n", c, right);
157             }
158         }
159         else {
160             for (i = 0; i < size; i++) {
161                 c = read1(in);
162                 if (verbose) printf("%d\n", c);
163                 fprintf(out, "%d\n", c);
164             }
165         }
166     }
167 }
168

```

```

169 /* 取り扱うファイルの名前 */
170 char *input_fname, *output_fname, *prog_name;
171
172 int main(int argc, char **argv)
173 {
174     int size;
175     FILE *in;
176     unsigned char buf[5120];
177     int fmt_tag, num_channel, sampling_rate, bytes_per_seconds;
178     int block, num_bits, extended_information_size;
179
180     prog_name = argv[0];
181     if (argc > 3)
182         usage(prog_name);
183
184     if (argc == 3) {
185         output_fname = argv[2];
186         if ((out = fopen(output_fname, "w")) == NULL) {
187             fprintf(stderr, "%s がオープンできません。 \n", output_fname);
188             exit(1);
189         }
190     }
191     else {
192         output_fname = "stdout";
193         out = stdout;
194     }
195     if (argc >= 2) {
196         input_fname = argv[1];
197         if ((in = fopen(input_fname, "r")) == NULL) {
198             fprintf(stderr, "%s がオープンできません。 \n", input_fname);
199             exit(1);
200         }
201     }
202     else {
203         input_fname = "input"; in = stdin;
204     }
205
206     /* ヘッダーを読む */
207     setstring(buf, in, 4); checkstring(buf, "RIFF");
208     size = readint(in); printf("# RIFF データのサイズ=%d\n", size);
209     setstring(buf, in, 4); checkstring(buf, "WAVE");
210     setstring(buf, in, 4); checkstring(buf, "fmt ");
211     size = readint(in); printf("# fmt データのサイズ=%d\n", size);
212     setstring(buf, in, size); dump(buf, size);
213     analyze(buf, &fmt_tag, &num_channel, &sampling_rate,
214             &bytes_per_seconds, &block, &num_bits,
215             &extended_information_size);
216     printf("# 非圧縮PCM%s。 \n", (fmt_tag == 1) ? "です" : "ではありません");
217     printf("# %s です。 \n", (num_channel == 1) ? "モノラル" : "ステレオ");
218     printf("# サンプリング・レート (標準化周波数)=%d\n", sampling_rate);
219     printf("# 1秒当りのバイト数=%d\n", bytes_per_seconds);
220     printf("# ブロック境界=%d\n", block);
221     printf("# ビット数/サンプル=%d\n", num_bits);
222     printf("# 拡張情報サイズ=%d\n", extended_information_size);
223     setstring(buf, in, 4);
224     /* 以下の if を書き加えた (2005/7/4) */

```



```

225     if (strcmp(buf, "PAD ") == 0) {
226         printf("# PAD チャンクがあります。");
227         size = readint(in); printf("# PAD チャンクのサイズ=%d\n", size);
228         setstring(buf, in, size);
229         dump(buf, size);
230         setstring(buf, in, 4);
231     }
232     if (strcmp(buf, "fact") == 0) {
233         printf("# fact チャンクがあります。");
234         size = readint(in); printf("# fact データのサイズ=%d\n", size);
235 #ifdef OLD
236         setstring(buf, in, 4); /* 多分 BUG */
237 #else
238         setstring(buf, in, size);
239 #endif
240         dump(buf, size);
241         setstring(buf, in, 4);
242     }
243     /* */
244     if (strcmp(buf, "FLLR") == 0) {
245         printf("# FLLR チャンクがあります。");
246         size = readint(in); printf("# FLLR データのサイズ=%d\n", size);
247         setstring(buf, in, size);
248         dump(buf, size);
249         setstring(buf, in, 4);
250     }
251     /* */
252     checkstring(buf, "data");
253     size = readint(in); printf("# data データのサイズ=%d\n", size);
254
255     /* 書き出す */
256     fprintf(out, "#original file: %s\n", input_fname);
257     fprintf(out, "#number of channels: %d\n", num_channel);
258     fprintf(out, "#sampling rate: %d\n", sampling_rate);
259     fprintf(out, "#number of bits (per sample): %d\n", num_bits);
260     fprintf(out, "#number of samples: %d\n",
261             size / num_channel / (num_bits / 8));
262     readpcm(in, size, num_bits, num_channel);
263     fclose(out);
264
265     return 0;
266 }

```

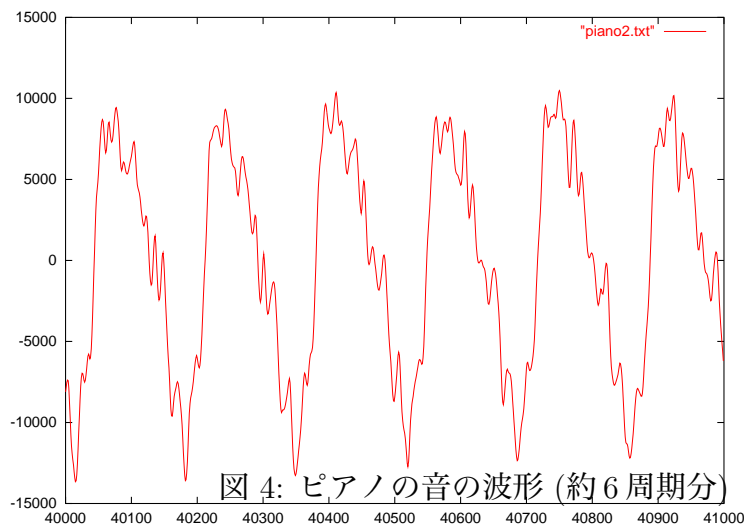
(2014/8/12) data が来るかというところで、FLLR というのが入ることがあり、そういうのは読めないことが判明した。

4.2.1 音の波形 (音圧の時間変化) を見る

```

oyabun% ./readwave piano.wav > piano.txt
oyabun% gawk '(NR>=40000 && NR< 41000) {print NR,$1,$2}' piano.txt > piano2.txt
oyabun% gnuplot
gnuplot> plot "piano2.txt" with lines

```



4.3 拙作 writewave.c (テキスト・ファイル→WAVE)

writewave は、readwave の出力を読んで WAVE ファイルを作成する。(完全な逆変換と言うわけではなく、ヘッダーが軽くなります。)

<http://nalab.mind.meiji.ac.jp/~mk/program/sound/writewave.c>

コンパイルと色々な使い方

```
oyabun% gcc -o writewave writewave.c
oyabun% ./readwave piano.wav piano.txt
oyabun% ./readwave piano.wav > piano2.txt
oyabun% ./writewave piano.txt piano1.wav
oyabun% ./writewave piano2.txt > piano2.wav
oyabun% ./readwave piano.wav | ./writewave > piano3.wav
```

```
1 /*
2 * writewave.c --- 無圧縮 PCM データ (テキスト・ファイル) から WAVE ファイルを作る
3 * version 1 (2003/12/21)
4 * version 2 (2013/10/26)
5 * ラベルの後に ; を入れた (コンパイル・エラー回避)。
6 * 文字コードを UTF8 にした。
7 *
8 * コンパイル: gcc -o writewave writewave.c
9 *
10 * 使い方:
11 * (1): ./writewave <テキスト・ファイル名> <WAVE ファイル名>
12 * (2): ./writewave <テキスト・ファイル名>
13 *      標準出力に出力
14 * (3): ./writewave
15 *      標準入力から入力、標準出力に出力
16 *
17 * 入手:
```

```

18 *   http://www.math.meiji.ac.jp/~mk/program/
19 *
20 *   参考にした情報
21 *   URL: http://member.nifty.ne.jp/Ryuz/programing/wavefmt.html
22 *       -> http://homepage3.nifty.com/ryuz/programing/wavefmt.html
23 *
24 *   普段置いてある場所
25 *       ~/Sotsuken/2007/sound/readwave/
26 */
27
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <strings.h>
31
32 int verbose = 1;
33
34 #define fwrite16(out,n) {fputc((n)&0xff,(out));fputc(((n)>>8)&0xff,(out));}
35
36 #define fwrite32(out,n) {fputc((n)&0xff,(out));fputc(((n)>>8)&0xff,(out));\
37 fputc(((n)>>16)&0xff,(out));fputc(((n)>>24)&0xff,(out));}
38
39 void usage(char *prog_name)
40 {
41     fprintf(stderr, "usage: %s <txt-file> <wave-file>", prog_name);
42     exit(1);
43 }
44
45 int main(int argc, char **argv)
46 {
47     int num_channels, num_bits, num_samples, sampling_rate;
48     FILE *in,*out;
49     char buf[4096], original_fname[1024];
50     char *prog_name, *input_fname, *output_fname;
51     int left, right, c;
52     int ready = 0, len;
53     int fmt_size, fmt_tag, bytes_per_seconds, block, extended_information_size,
54         data_size, riff_size;
55     static char label[5][32] =
56     {"#original file: ",
57      "#number of channels: ",
58      "#number of bits (per sample): ",
59      "#number of samples: ", "#sampling rate: "};
60
61     prog_name = argv[0];
62
63     if (argc > 3) {
64         usage(prog_name);
65     }
66     if (argc == 3) {
67         output_fname = argv[2];
68         if ((out = fopen(output_fname, "w")) == NULL) {
69             fprintf(stderr, "出力ファイル %s がオープンできません。 \n",
70                 output_fname);
71             exit(1);
72         }
73     }

```

```

74  else {
75      output_fname = "stdout";
76      out = stdout;
77  }
78  if (argc >= 2) {
79      input_fname = argv[1];
80      if ((in = fopen(input_fname, "r")) == NULL) {
81          fprintf(stderr, "入力ファイル %s がオープンできません。",
82                  input_fname);
83          exit(1);
84      }
85  }
86  else {
87      input_fname = "stdin";
88      in = stdin;
89  }
90  while (fgets(buf, sizeof(buf), in) != NULL) {
91      if (buf[0] == '#') {
92          switch (buf[1]) {
93              case 'o':
94                  if (strncmp(buf, label[0], len = strlen(label[0])) == 0) {
95                      sscanf(buf + len, "%s", original_fname);
96                      ready |= 0x01;
97                  }
98                  break;
99              case 'n':
100                 if (strncmp(buf, label[1], len = strlen(label[1])) == 0) {
101                     sscanf(buf + len, "%d", &num_channels);
102                     ready |= 0x02;
103                 }
104                 else if (strncmp(buf, label[2], len = strlen(label[2])) == 0) {
105                     sscanf(buf + len, "%d", &num_bits);
106                     ready |= 0x04;
107                 }
108                 else if (strncmp(buf, label[3], len = strlen(label[3])) == 0) {
109                     sscanf(buf + len, "%d", &num_samples);
110                     ready |= 0x08;
111                 }
112                 else {
113                     fprintf(stderr, "unknown: %s\n", buf);
114                 }
115                 break;
116             case 's':
117                 if (strncmp(buf, label[4], len = strlen(label[4])) == 0) {
118                     sscanf(buf + len, "%d", &sampling_rate);
119                     ready |= 0x10;
120                 }
121                 break;
122             default:
123                 ;
124                 /* 何もしない */
125             }
126         if (ready == 0x1f) {
127             if (verbose) {
128                 fprintf(stderr, "original file: %s\n", original_fname);
129                 fprintf(stderr, "number of channels: %d\n", num_channels);

```

```

130         fprintf(stderr, "number of bits: %d\n", num_bits);
131         fprintf(stderr, "number of samples: %d\n", num_samples);
132         fprintf(stderr, "sampling rate: %d\n", sampling_rate);
133     }
134     break;
135 }
136 }
137 else {
138     /* パラメーターが揃わない (ready == 0x1f にならない) うちに
139     * 生データが来たらエラーだ */
140     printf("ready=%0x\n", ready);
141     fprintf(stderr, "%s\n", buf);
142     exit(1);
143 }
144 }
145
146 /* fmt チャンク */
147 fmt_size = 18; /* 2+2+4+4+2+2+2 */
148 fmt_tag = 1; /* 無圧縮 PCM */
149 bytes_per_seconds = sampling_rate * (num_bits / 8) * num_channels;
150 block = 1; /* ブロック境界 (何のこと?) */
151 extended_information_size = 0;
152 /* fact チャンクつけない */
153 /* data チャンク */
154 data_size = num_samples * (num_bits / 8) * num_channels;
155 /* */
156 riff_size = fmt_size + data_size;
157
158 /* ヘッダーを書く */
159 fprintf(out, "RIFF");
160 fwrite32(out, riff_size);
161 fprintf(out, "WAVE");
162 fprintf(out, "fmt ");
163 fwrite32(out, fmt_size);
164 fwrite16(out, fmt_tag); /* 2bytes */
165 fwrite16(out, num_channels); /* 2bytes */
166 fwrite32(out, sampling_rate); /* 4bytes */
167 fwrite32(out, bytes_per_seconds); /* 4bytes */
168 fwrite16(out, block); /* 2bytes */
169 fwrite16(out, num_bits); /* 2bytes */
170 fwrite16(out, extended_information_size); /* 2bytes */
171 fprintf(out, "data");
172 fwrite32(out, data_size);
173
174 if (num_channels == 2 && num_bits == 16)
175     while (fgets(buf, sizeof(buf), in) != NULL) {
176         sscanf(buf, "%d%d", &left, &right);
177         fputc(left & 0xff, out); fputc((left >> 8) & 0xff, out);
178         fputc(right & 0xff, out); fputc((right >> 8) & 0xff, out);
179     }
180 else if (num_channels == 2 && num_bits == 8)
181     while (fgets(buf, sizeof(buf), in) != NULL) {
182         sscanf(buf, "%d%d", &left, &right);
183         fputc(left & 0xff, out);
184         fputc(right & 0xff, out);
185     }

```

```

186     else if (num_channels == 1 && num_bits == 16)
187         while (fgets(buf, sizeof(buf), in) != NULL) {
188             sscanf(buf, "%d", &c);
189             fputc(c & 0xff, out); fputc((c >> 8) & 0xff, out);
190         }
191     else if (num_channels == 1 && num_bits == 8)
192         while (fgets(buf, sizeof(buf), in) != NULL) {
193             sscanf(buf, "%d", &c);
194             fputc(c & 0xff, out);
195         }
196     fclose(out);
197     return 0;
198 }

```

5 Java でサウンドを扱う

5.1 参考にした情報

1. Sun が公式に出している情報 (ネットでクラスやメソッドの名前で検索すると出て来るもの)
2. <http://forum.java.sun.com/thread.jspa?threadID=5205151&tstart=75>
3. 『Java で HelloWorld サウンド編』⁶

2007 年度卒研が終ってから、本屋で赤間 [7] を見つけた (忙しいとのんびり本屋さんの棚を見る時間がないということで、とてもありがちなこと)。これは便利 (お勧め)。まあ苦勞したのはは無駄にならないと考えることにしよう。

桂田研の学生の Java はぶっつけ本番。戸川 [8] くらいを与えて、プールに突き飛ばす感じ です。

5.2 拙作 DumpWave.java — WAVE ファイルはこうやって読める

<http://nalab.mind.meiji.ac.jp/~mk/program/sound/DumpWave.java>
 要点は

```

AudioInputStream ais = AudioSystem.getAudioInputStream(new File("sound.wav"));
byte [] data = new byte [ais.available()];
ais.read(data);
ais.close();

```

ということ。

(クラス・ライブラリが使えるので、readwave.c のような低レベルのことはする必要がない。)

⁶<http://www.hellohiro.com/sound.htm>

以上はデータを一気に読む場合だが、1バイトずつ読んだり、適当なサイズずつ読んだりする read() メソッドもある(ネットで検索すると簡単に見つかる)。

```
1  /*
2  * DumpWave.java
3  */
4
5  import java.io.IOException;
6  import java.io.File;
7  import javax.sound.sampled.*;
8
9  public class DumpWave {
10     public static void main(String [] args) {
11         try {
12             AudioInputStream ais =
13                 AudioSystem.getAudioInputStream(new File(args[0]));
14             // オーディオ入力ストリームからデータを読む (一気に読むバージョン)
15             byte [] data = new byte [ais.available()];
16             ais.read(data);
17             ais.close();
18             // ファイルのフォーマットを調べる
19             AudioFormat af = ais.getFormat();
20             System.out.println("#channels: "          + af.getChannels());
21             System.out.println("#isBigEndian: "       + af.isBigEndian());
22             System.out.println("#FrameSize: "        + af.getFrameSize());
23             System.out.println("#SampleSizeInBits: " + af.getSampleSizeInBits());
24             System.out.println("#SampleRate: "      + af.getSampleRate());
25             if (af.getEncoding() == AudioFormat.Encoding.PCM_SIGNED) {
26                 System.out.println("#符号付 PCM");
27             }
28             else if (af.getEncoding() == AudioFormat.Encoding.PCM_UNSIGNED) {
29                 System.out.println("#符号無 PCM");
30             }
31             System.out.println("#サイズ: " + data.length + "バイト, "
32                 + data.length / af.getSampleRate() + "秒");
33             // ステレオ,16ビット,符号付PCM,リトルエンディアンのみ
34             if (af.getChannels() == 2 &&
35                 af.getSampleSizeInBits() == 16 &&
36                 af.getEncoding() == AudioFormat.Encoding.PCM_SIGNED &&
37                 (!af.isBigEndian())) {
38                 short left, right;
39                 for (int i = 0; i < data.length; i += 4) {
40                     left = (short)(data[i] & 0xff | (data[i+1] << 8));
41                     right = (short)(data[i+2] & 0xff | (data[i+3] << 8));
42                     System.out.println("" + left + " " + right);
43                 }
44             }
45             else {
46                 System.out.println("#ステレオ,16ビット,符号付PCM,リトルエンディアンのみ対
47                 応");
48             }
49         } catch (Exception e) {
50             e.printStackTrace();
51             System.exit(1);
52         }
53     }
54 }
```

コンパイル&実行

```

mathpc% javac DumpWave.java
mathpc% java DumpWave piano.wav
#channels: 2
#isBigEndian: false
#FrameSize: 4
#SampleSizeInBits: 16
#SampleRate: 44100.0
#符号付 PCM
#サイズ: 1228068 バイト, 27.847347 秒
64 64
62 62
65 62
(後略)

```

5.3 拙作 PlayWave.java — WAVE ファイルを読んで音を再生

<http://nalab.mind.meiji.ac.jp/~mk/program/sound/PlayWave.java>

『Java で HelloWorld サウンド編』⁷ にサンプル・プログラムがある。それを参考にして作ったプログラム。

```

1  /*
2   * PlayWave.java --- WAVE ファイルの音を再生する
3   */
4
5  import java.io.IOException;
6  import java.io.File;
7  import javax.sound.sampled.*;
8
9  public class PlayWave {
10     public static void main(String [] args) {
11         try {
12             AudioInputStream ais =
13                 AudioSystem.getAudioInputStream(new File(args[0]));
14             // オーディオ入力ストリームからデータを読む
15             byte [] data = new byte [ais.available()];
16             ais.read(data);
17             ais.close();
18             // ファイルのフォーマットを調べる
19             AudioFormat af = ais.getFormat();
20             // 再生する
21             DataLine.Info info = new DataLine.Info(SourceDataLine.class, af);
22             SourceDataLine line = (SourceDataLine)AudioSystem.getLine(info);
23             line.open(af);

```

⁷<http://www.hellohiro.com/sound.htm>


```

24         line.start();
25         line.write(data, 0, data.length);
26     } catch (Exception e) {
27         e.printStackTrace();
28         System.exit(1);
29     }
30 }
31 }

```

もしかして、`line.drain();` と `line.close();` も必要かな。

コンパイル&実行

```

knoppix$ javac PlayWave.java
knoppix$ java PlayWave piano.wav

```

なお、Clip として再生する方法もある。見比べると良い。

```

1  /*
2  * PlayWave_another.java --- Clip を使って再生する (参考)
3  *   データの中身が見えないので卒研的にはあまり意味がないけれど...
4  */
5
6  import java.io.*;
7  import javax.sound.sampled.*;
8
9  public class PlayWave_another {
10     public static void main(String [] args) throws Exception {
11         AudioInputStream ais =
12             AudioSystem.getAudioInputStream(new File(args[0]));
13         Clip clip = (Clip)AudioSystem.getLine(new Line.Info(Clip.class));
14         try {
15             clip.open(ais);
16             clip.loop(0); // 0+1 回再生...それなら clip.start() でもよい
17             while (clip.isRunning()) {
18                 Thread.sleep(100);
19             }
20         }
21         finally {
22             clip.close();
23             System.exit(0); // 意外にも必要
24         }
25     }
26 }

```

5.4 マイクロホンで録音して WAVE ファイルを作成

『Java で HelloWorld サウンド編』⁸ にサンプル・プログラムがある。

⁸<http://www.hellohiro.com/sound.htm>

5.5 拙作 ReadWave.java

(これは古くてバグ入りだが、卒研の記録として残しておく。既に紹介した『JavaでHelloWorldサウンド編』を参考にしたのだが、卒研が終ってから、赤間 [7] という本を見つけて、「この本があれば楽だったのに」と、ちょっと悔しい思いをした。)

```
http://nalab.mind.meiji.ac.jp/~mk/program/sound/ReadWave.java
```

ReadWave.java は、Wave ファイルを読み込んで、音を鳴らしながら、音声データを数値表示するプログラムである。オーディオ・データの形式は、Java のクラス・ライブラリが調べてくれるので、自前で解析する手間は省けている (Cf: readwave.c)。

コンパイル&実行

```
knoppix$ javac ReadWave.java
knoppix$ java ReadWave piano.wav > piano.txt
```

チェック用に <http://nalab.mind.meiji.ac.jp/~mk/labo/2007/piano.wav> と <http://nalab.mind.meiji.ac.jp/~mk/labo/2007/piano.txt> を公開しておく。

ソース・プログラムは案外長いが、(i) 16ビット・ステレオ、(ii) 8ビット・ステレオ、(iii) 16ビット・モノラル、(iv) 8ビット・モノラル、と4つに場合分けしてあるからである (本当はこれにエンディアンの違いと、符号の有無の違いで、さらに場合分けする必要があるが、それは無視した— と書いたのだけど、これは誤解かも知れない (2013/10)。)。

```
1 //
2 // ReadWave.java
3 // 2008/1/30 初めて作成
4 // 2008/2/13 一木君にバグを指摘される。Byte は符号付きだった。
5 // 2008/2/15 やはりビット演算だけで記述することにした。
6 //
7
8 import java.io.File;
9 import javax.sound.sampled.*;
10
11 public class ReadWave {
12     private static final int EXTERNAL_BUFFER_SIZE = 128000;
13     public static void main(String[] args) {
14         int frameSize; //
15         int sampleSizeInBits; // 音声データの数値のビット数 (16 または 8)
16         int channels; // チャンネルの数 (2 がステレオ, 1 がモノラル)
17         float sampleRate; // サンプリングレート (1 秒間の...)
18         boolean isStereo; // ステレオか
19         boolean isBigEndian; // ビッグエンディアンか (上位バイトが先か)
20         if (args.length == 0) System.exit(0);
21         try {
22             // File クラスのインスタンスを生成する
23             File soundFile = new File(args[0]);
24             // オーディオ入力ストリームを取得する
25             AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(soundFile);
26             // オーディオフォーマットを取得する
27             AudioFormat audioFormat = audioInputStream.getFormat();
28
29             // データラインの情報オブジェクトを生成する
30             DataLine.Info info = new DataLine.Info(SourceDataLine.class, audioFormat);
```

```

31 // 指定されたデータライン情報に一致するラインを取得する
32 SourceDataLine line = (SourceDataLine) AudioSystem.getLine(info);
33 // 指定されたオーディオ形式でラインを開きます
34 line.open(audioFormat);
35 // ラインでのデータ入出力を可能にします
36 line.start();
37
38 // データの形式を読み取る
39 channels = audioFormat.getChannels();
40 isStereo = (channels == 2);
41 isBigEndian = audioFormat.isBigEndian();
42 frameSize = audioFormat.getFrameSize();
43 sampleSizeInBits = audioFormat.getSampleSizeInBits();
44 sampleRate = audioFormat.getSampleRate();
45 System.out.println("#original file: " + args[0]);
46 System.out.println("#number of channels: " + channels);
47 System.out.println("#sampling rate: " + sampleRate);
48 System.out.println("#number of bits per sample: " + sampleSizeInBits);
49 System.out.println("#FrameSize: " + frameSize);
50 System.out.println("#isBigEndian: " + isBigEndian);
51 if (audioFormat.getEncoding() == AudioFormat.Encoding.PCM_SIGNED) {
52     System.out.println("#PCM Signed");
53 }
54 else if (audioFormat.getEncoding() == AudioFormat.Encoding.PCM_UNSIGNED) {
55     System.out.println("#PCM Unsigned!!!");
56     System.exit(0);
57 }
58 else {
59     System.out.println("#NO PCM!!");
60     System.exit(0);
61 }
62
63 // 音声データを読み取り、鳴らして、数値を表示
64 int nBytesRead = 0;
65 byte[] abData = new byte[EXTERNAL_BUFFER_SIZE];
66 if (isStereo) {
67     if (sampleSizeInBits == 16) {
68         // 16ビットステレオ (フツー)
69         while (nBytesRead != -1) {
70             // オーディオストリームからデータを読み込みます
71             nBytesRead = audioInputStream.read(abData, 0, abData.length);
72             if (nBytesRead >= 0) {
73                 // オーディオデータをミキサーに書き込みます
74                 int nBytesWritten = line.write(abData, 0, nBytesRead);
75                 for (int i = 0; i < nBytesRead; i += 4) {
76                     short left, right;
77                     left = (short)(abData[i] & 0xff | (abData[i+1] << 8));
78                     right = (short)(abData[i+2] & 0xff | (abData[i+3] << 8));
79                     System.out.println("" + left + " " + right);
80                 }
81             }
82         }
83     }
84     else { // sampleSizeInBits == 8
85         // 8ビットステレオ (フツー)
86         while (nBytesRead != -1) {

```

```

87         // オーディオストリームからデータを読み込みます
88         nBytesRead = audioInputStream.read(abData, 0, abData.length);
89         if (nBytesRead >= 0) {
90             // オーディオデータをミキサーに書き込みます
91             int nBytesWritten = line.write(abData, 0, nBytesRead);
92             for (int i = 0; i < nBytesRead; i += 2) {
93                 short left, right;
94                 left = (short)(abData[i] & 0xff);
95                 right = (short)(abData[i+1] & 0xff);
96                 System.out.println("" + left + " " + right);
97             }
98         }
99     }
100 }
101 }
102 else {
103     // モノラル
104     if (sampleSizeInBits == 16) {
105         // 16ビットモノラル
106         while (nBytesRead != -1) {
107             // オーディオストリームからデータを読み込みます
108             nBytesRead = audioInputStream.read(abData, 0, abData.length);
109             if (nBytesRead >= 0) {
110                 // オーディオデータをミキサーに書き込みます
111                 int nBytesWritten = line.write(abData, 0, nBytesRead);
112                 for (int i = 0; i < nBytesRead; i += 2) {
113                     short c;
114                     c = (short)(abData[i] & 0xff | (abData[i+1] << 8));
115                     System.out.println("" + c);
116                 }
117             }
118         }
119     }
120     else { // sampleSizeInBits == 8
121         // 8ビットモノラル
122         while (nBytesRead != -1) {
123             // オーディオストリームからデータを読み込みます
124             nBytesRead = audioInputStream.read(abData, 0, abData.length);
125             if (nBytesRead >= 0) {
126                 // オーディオデータをミキサーに書き込みます
127                 int nBytesWritten = line.write(abData, 0, nBytesRead);
128                 for (int i = 0; i < nBytesRead; i++) {
129                     short c;
130                     c = (short)(abData[i] & 0xff);
131                     System.out.println("" + c);
132                 }
133             }
134         }
135     }
136 }
137 // ラインからキューに入っているデータを排出します
138 line.drain();
139 // ラインを閉じます
140 line.close();
141
142 System.exit(0);

```

```

143     } catch (Exception e) {
144         e.printStackTrace();
145         System.exit(1);
146     }
147 }
148 }

```

5.6 拙作 MakeWaveFile.java — 数値データから Wave ファイルを作る

<http://nalab.mind.meiji.ac.jp/~mk/program/sound/MakeWaveFile.java>

```

1  /*
2  * MakeWaveFile.java
3  *   version 1 (2008/3/5) by mk
4  *   参考: http://okwave.jp/qa1942478.html
5  http://java.sun.com/j2se/1.5.0/ja/docs/ja/api/javax/sound/sampled/AudioSystem.html
6  */
7
8  import java.io.IOException;
9  import java.io.File;
10 import java.io.InputStream;
11 import java.io.ByteArrayInputStream;
12 import javax.sound.sampled.*;
13
14 public class MakeWaveFile {
15     public static void main(String[] args) throws IOException {
16         // 音の PCM データ (44.1kHz, 量子化ビット数 16, ステレオ, 3 秒)
17         byte[] data = new byte[44100 * 2 * 2 * 3];
18         // 適当なデータ (良い子は自分が考えるデータに変えてみよう)
19         for (int i = 0; i < data.length; i++) {
20             data[i] = (byte) (((i & (1 << 6)) == 0) ? -18 : 18);
21         }
22         // オーディオフォーマットを決める (44.1kHz, 16bit, stereo)
23         AudioFormat audioFormat =
24             new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
25                             44100.0F, 16, 2, 4, 44100.0F, false);
26         // ByteArrayInputStream を AudioInputStream にする
27         InputStream in = new ByteArrayInputStream(data);
28         AudioInputStream ais = new AudioInputStream(in,
29                                                     audioFormat,
30                                                     44100 * 2 * 2);
31         // オーディオファイルの種類を指定
32         AudioFileFormat.Type targetType = AudioFileFormat.Type.WAVE;
33         // 出力ファイル名
34         File outputFile = new File("test.wav");
35         // えいつ、と書く
36         AudioSystem.write(ais, targetType, outputFile);
37     }
38 }

```

5.7 FFT

5.7.1 FFTPACK

FFTW とかにも興味があるのだけれど、Java からどう使うかイマイチ分からなかったので、まあまあ分かっているものを叩き台にすることにした。

「FFT についてのメモ — FFTPACK の利用に向けて —」⁹

5.7.2 大浦 FFT

C で書かれた汎用 FFT (高速 フーリエ/コサイン/サイン 変換) パッケージ by 大浦拓哉¹⁰ は、比較的簡単に Java に書き直せる。

コンパイル&テスト

```
oyabun% tar xzf fft.tgz
oyabun% cd fft
oyabun% cd sample1
oyabun% make
oyabun% ./test4g
data length n=? (must be 2^m)
1024
cdft err= 5.55112e-16
rdft err= 4.44089e-16
ddct err= 7.77156e-16
ddst err= 6.66134e-16
dfct err= 4.44089e-16
dfst err= 7.77156e-16
  test4g_h, test8g, test8g_h, testsg, testsg_h についても同様
oyabun% cd ../sample2
oyabun% make
oyabun% ./pi_fft4g
... (略) ...
```

2007 年度卒研で一木君は `fft4g.c` を Java に書き換えて使用した。

5.7.3 jfftpack

Paul N. Swarztrauber の **FFTPACK**¹¹ は古くからあるライブラリだが、Java への移植 **jfftpack** (by Baoshe Zhang `baoshe.zhang@uleth.ca`)¹² がある。

⁹<http://nalab.mind.meiji.ac.jp/~mk/labo/text/fft-lecture.pdf>

¹⁰<http://www.kurims.kyoto-u.ac.jp/~ooura/fft-j.html>

¹¹<http://www.netlib.org/fftpack/>

¹²<http://www.netlib.org/fftpack/jfftpack.tgz>

FFTPACK については、マニュアルの邦訳「FFTPACK」¹³ を用意してある。
卒研では、サンプル・プログラム testjffftpack-unix.tar.gz¹⁴, testjffftpack-win.lzh¹⁵ を用意した。

```
knoppix$ tar xzf testjffftpack-unix.tar.gz
knoppix$ cd testjffftpack-unix
knoppix$ javac -cp jffftpack.jar testfft.java
knoppix$ java -cp jffftpack.jar:. testfft
```

```
1  /*
2  * testfft.java
3  *   FFTPACk の Java 版 jffftpack (http://www.netlib.org/fftpack/ で入手可)
4  *   http://www.math.meiji.ac.jp/~mk/labo/2007/jffftpack.jar に載せる
5  *
6  *   jffftpack.jar の位置を環境変数 CLASSPATH で指定する。
7  *   (1) Windows では set CLASSPATH=%CLASSPATH%;どこか\jffftpack.jar
8  *   (2) Linux では export CLASSPATH=$CLASSPATH:どこか/jffftpack.jar
9  *
10 *   あるいは java コマンドの -classpath オプションを使う。
11 *   java -classpath どこか/jffftpack.jar:. testfft
12 *
13 */
14
15 import ca.uol.aig.fftpack.*;
16
17 public class testfft {
18     public static void main(String args[]) {
19         int i,n;
20         double dx,x;
21         double [] a;
22         n = 1024;
23         a = new double [n];
24         RealDoubleFFT myfft = new RealDoubleFFT(n);
25         // n 等分点上の関数値を求める
26         dx = 2 * Math.PI / n;
27         for (i = 0; i < n; i++) {
28             x = i * dx;
29             a[i] = 1+2*Math.cos(x)+3*Math.sin(x)+4*Math.cos(2*x);
30         }
31         // 離散 Fourier 変換
32         myfft.ft(a);
33         // Fourier 係数を求める
34         a[0] /= n;
35         for (i = 1; i < n; i++)
36             a[i] = a[i] / (n / 2.0);
37         // 1,2,3,4 が現われますやら…
38         for (i = 0; i < 10; i++)
39             System.out.printf("i=%d: %f\n", i, Math.abs(a[i]));
40         //           System.out.println("i=" + i + ":" + Math.abs(a[i]));
41     }
42 }
```

¹³<http://nalab.mind.meiji.ac.jp/~mk/labo/text/fftpack-index.pdf>

¹⁴<http://nalab.mind.meiji.ac.jp/~mk/labo/2007/testjffftpack-unix.tar.gz>

¹⁵<http://nalab.mind.meiji.ac.jp/~mk/labo/2007/testjffftpack-win.lzh>

```
[chronos:sound/sound/jfftpack] mk% java testfft
i=0: 1.000000
i=1: 2.000000
i=2: 3.000000
i=3: 4.000000
i=4: 0.000000
i=5: 0.000000
i=6: 0.000000
i=7: 0.000000
i=8: 0.000000
i=9: 0.000000
[chronos:sound/sound/jfftpack] mk%
```

無事に 1, 2, 3, 4, 0, ... となりました。

防備録: jfftpack.jar の作り方 ca.uol.aig.fftpack という package 名であることに注意。

```
mkdir -p ca/uol/aig/fftpack      (ディレクトリを用意)
cp -p どこか/*.java ca/uol/aig/fftpack  (ソース・プログラムを用意)
javac ca/uol/aig/fftpack/*.java  (ソース・プログラムをコンパイル)
jar -cvf /fftpack.jar .          (jar ファイルを作る)
jar -tvf /fftpack.jar            (jar ファイルの中身を確認する)
```

6 MATLAB

(準備中)

さすがに便利そうなのだけれど、最近は Mathematica で色々やっているの、こちらで何かするのは、いつになるか分からない。さすがに Octave では、`sound()` は `undefined` になる…

ネットで検索すると、<http://lis2.huie.hokudai.ac.jp/~toyo/MATLAB/> という便利なページがある。

6.1 `sound()` で音を鳴らす

`sound()` は Mathematica の `ListPlay[]` に相当する。


```
samplerate=44100;
Tmax=1;
freq=440;
t=0:1/samplerate:Tmax;
sampledata=sin(2*pi*freq*t);
sound(sampledata,samplerate);
```

6.2 wavrecord() で録音

```
samplerate=44100;
Tmax=10;
sampledata=wavrecord(Tmax*samplerate,samplerate);

wavwrite(sampledata,samplerate,'sample.wav');
```

6.3 wavread() で WAVE ファイルを読む

```
sampledata=wavread('sample.wav');

[sampledata,samplerate,bits]=wavread('sample.wav');
```

`wavread('sample.wav', numtoread);` や `wavread('sample.wav', [from to]);` などのようなことが出来る。

6.4 wavwrite()

```
wavwrite(sampledata,'file.wav');
wavwrite(sampledata,samplerate,'file.wav');
wavwrite(sampledata,samplerate,N,'file.wav');  N=8,16,24,32
```

A Mac の相場

A.1 QuickTime Player での録音

QuickTime Player で一応録音は出来る。QuickTime Player は Finder でアプリケーション・フォルダーを探すと見つかる。[ファイル]メニューから新規オーディオ収録を選択し、右側に

あるボタンで録音品質を選択する。



品質は「高」、「最高」の二つから選択できる (何だそれは)。

- 「最高」にすると拡張子が `.aifc` の AIFC というフォーマットになる。
- 「高」にすると拡張子が `.m4a` というフォーマットになる。

AIFC は、AIFF (Audio Interchange File Format) という 1990 年頃に Apple が決めたフォーマットをもとに、圧縮を取り入れたものである。

AIFF そのものは Linear PCM であり、圧縮はしていない。

「最高」で録音すると、拡張子は `.aifc` であるが、どうも 24 ビット Linear PCM, ステレオ、サンプリング周波数 44.1kHz であるようだ (以下のやり方で 24 ビット WAVE に変換するとファイルのサイズが同じになる — いい加減な根拠だ…まあ、そのうち調べよう)。ちなみに Mathematica 9 で `Import[]` で読もうとしても駄目だった (けっこう `Import[]` へたれです)。

`afconvert` という Apple 純正のコマンドライン・ツールでフォーマット変換が出来る。例えばターミナルで次のようにして「普通の」WAVE ファイルに変換できる。

nantoka.aifc を WAVE 形式のファイル nantoka.wav に変換

```
afconvert -f WAVE -d LEI16 nantoka.aifc nantoka.wav
```

(-d はデータのフォーマットの指定で、(a) エンディアン, (b) 値のタイプ (U or LEI or LEF), (c) ビット数という3つの項目を表す文字列で Linear PCM の形式を表す、ということらしい。例えば LEI16 は、リトルエンディアンの符号あり整数 16 ビットという意味である。ここで他に指定できるフォーマットは、UI8 (符号なし整数 8 ビット), LEI24 (リトルエンディアン符号あり整数 24 ビット), LEI32 (リトルエンディアン符号あり整数 32 ビット), LEF32 (リトルエンディアン符号あり浮動小数 32 ビット), LEF64 (リトルエンディアン符号あり浮動小数 64 ビット) だとか。)

LEI16 の後に @ 数値をくっつけて、サンプリング周波数の指定もできるみたい。-d LEI16@22050 (22050Hz)

Mathematica 9 では、LEI16 はもちろんだが、LEI24 も読むことが出来た。

その他、afconvert で扱えるフォーマットにどのようなものがあるかは、afconvert -hf で表示される短いヘルプメッセージを見ると良い (というか、長い説明は用意されていない)。

-c 1 とするとモノラルに出来る (チャンネル数を 1 にする)。

B iPhone での録音

(工事中。)

標準のアプリである **ボイスメモ** で録音出来る。

iPhone の外への転送の仕方は、iTunes を使う方法、AirDrop を使う方法、メールに添付する方法などがあるそう (こういうのは学生の方が詳しそう)。

拡張子が .m4a のファイルが出来るが、ffmpeg¹⁶ で WAVE ファイルに変換できる。例えば、piano.m4a を WAVE ファイル piano.wav に変換するには、ターミナルで

```
ffmpeg -i piano.m4a piano.wav
```

とすれば良い。

C 調べないと

Audacity とか色々フリーのツールがある。

C.1 音高 (ピッチ) を調べる

難波編 [9] 第 7 章に

1. 位相差計測法
2. 複素スペクトル内挿法

¹⁶フリーソフトで、現象数理学科 Mac にはインストール済みのはず。

というのが紹介されている。記述が簡単すぎて良く分からないが、原典 (井口 [10], 原・井口 [11]) にあたって勉強してみよう。

C.2 インハーモニシティ

太田 [12] 土橋 [13]

- Wikipedia の Inharmonicity
- Isoharu Nishiguchi, Recent research on the acoustics of pianos, http://www.jstage.jst.go.jp/article/ast/25/6/413/_pdf
- Neville H. Fletcher, Harmonic? Anharmonic? Inharmonic?, Am. J. Phys. 70 12!, December 2002, <http://www.phys.unsw.edu.au/music/people/publications/Fletcher2002.pdf>
- Numerical simulations of piano strings. I. A physical model for a struck string using finite difference methods, <http://people.cs.uchicago.edu/~ridg/stabil/pianostring.pdf>
- <http://piano.s20.xrea.com/mecha/doc03.html>
- 川口和也, 中井幹雄, 打弦時に発生するインハーモニシティに関する研究, 日本機械学会 Dynamics and Design Conference 2001, <http://www.jsme.or.jp/monograph/dmc/2001/data/pdf/729.pdf>
これと同じか? Dynamics & Design Conference. アブストラクト集, Vol.2003, No.abstract(20030915) p. 218, The Japan Society of Mechanical Engineers ISSN:13480235
- Dynamics & Design Conference. アブストラクト集 Vol.2003, No.abstract(20030915) p. 218 The Japan Society of Mechanical Engineers ISSN:13480235
- Dynamics & Design Conference. アブストラクト集 Vol.2003, No.abstract(20030915) p. 218 The Japan Society of Mechanical Engineers ISSN:13480235
- ROBERT W. YOUNG, Inharmonicity of Plain Wire Piano Strings, The Journal of the Acoustical Society of America, Volume 24, Number 3, MAY 1952, <http://www.afn.org/~afn49304/youngnew.htm>
- 『もっと知りたいピアノのしくみ』 (西口磯春、森太郎 著／音楽之友社)

D 個人的記憶

D.1 レコード

D.2 テープレコーダー

D.3 CDプレーヤー

D.4 ベネット『パソコンプログラム 理科系のための問題演習』

ベネット [14]

参考文献

- [1] 横山和正：常微分作用素の固有値問題の数値解析, <http://nalab.mind.meiji.ac.jp/~mk/labo/report/open/2003-yokoyama.pdf> (2004).
- [2] 桂田祐史：卒業研究テーマ 固有値問題とそのシミュレーション, <http://nalab.mind.meiji.ac.jp/~mk/labo/2003/eigenvalue/> (2003年11月11日).
- [3] N. H. フレッチャー, T. D. ロッシング：楽器の物理学, シュプリンガー・フェアラーク東京 (2002), 岸 憲史, 久保田 秀美, 吉川 茂 訳.
- [4] 城都健一：デジタルフーリエ解析 (I) —基礎編—, コロナ社 (2007).
- [5] 城都健一：デジタルフーリエ解析 (II) —上級編—, コロナ社 (2007).
- [6] 松山周五郎：音の Fourier 解析, 2003 年度卒業研究レポート, <http://nalab.mind.meiji.ac.jp/~mk/labo/report/open/2003-matsuyama.pdf> (2004).
- [7] ^{あかませいき}赤間世紀：JavaSound 教科書, 工学社 (2007).
- [8] ^{はやと}戸川隼人：演習と応用 Java, サイエンス社 (2004).
- [9] 難波 精一郎編：音の科学, 朝倉書店 (1989).
- [10] 井口征士：音楽情報の処理 — 電算機を用いた自動採譜, 計測と制御, Vol. 19, pp. 314–319 (1980).
- [11] 原裕一郎, 井口征士：複素スペクトルを用いた周波数同定, 第 19 卷 (1983).
- [12] 太田高正：ピアノ弦の性質と調律曲線, 数学セミナー, 2007 年 12 月号, pp. 37–41 (2007).
- [13] 土橋光義：Sound and Temperament Lab ピアノの音と音律：<http://park11.wakwak.com/~md440/>.

- [14] W. R. ベネット：パソコンプログラム 理科系のための問題演習, 現代数学社 (1983), William Ralph Bennett, Scientific and Engineering Problem-solving with the Computer, Prentice Hall (1976) の邦訳。学生の頃、邦訳の方が愛読書でした (どこかに行ってしまった…)。今古本で買うと高いので、原著の方を買ってみました (ハードカバーでなかなか立派な本です)。