

audio-textbook

1 この文書について


この文書は音響信号処理の初歩的事項に関する資料です。おおよそ大学 1~2 年生くらいが 2~3 時間くらいで取り組むことをイメージしています。R を使います。

2009 年の初稿以降少しずつマイナーチェンジしています。

1.1 今後の改訂予定

随時コンテンツの追加を予定。

1.2 凡例



- ♠: 検索
- : エンター

2 R のインストール、その他下準備

2.1 パソコンの用意

1. パソコンを用意してください。
2. ネットに接続してください。

2.2 R

1. 各自の環境に R をインストールします。♠
2. R を起動します。
3. `install.packages("tuneR")` と入力してエンター  を押します。するとオーディオファイル入出力などの機能が一式揃った tuneR というパッケージがインストールされます。
4. `library(tuneR)`  これで tuneR パッケージがロードされます。

2.3 データの用意

1. 適当に*.wav 形式のファイルを用意します。PC の録音機能を使ったり、ネットから適当に拾ってきたりします。(https://github.com/tachi-hi/sampleSounds など)

3 正弦波をプロット

R のコンソールで以下のように入力します。(“#”以降はコメントなので入力しなくていいです.)

```
f <- 440                # 周波数 440 Hz
fs <- 16000             # サンプリングレート 16000 Hz
t <- 1:(3 * fs) / fs     # 時間.長さ 3 秒
x <- sin (2 * pi * f * t) # 信号
plot (x)                # 信号をプロット
```

これで

$$x(t) = \sin 2\pi ft, \quad (f = 440\text{Hz}, 0[s] < t \leq 3[s])$$

という信号をプロットできます。

3.1 文法の解説

R では代入を `a <- 1` というふうに表記します。`<-`という記号は左向き矢印 \leftarrow を ASCII 文字だけで表現したものです。

コロンの“:”は数列の意味です。`1:10` はベクトル

```
c(1,2,3,4,5,6,7,8,9,10)
```

と同じ意味です。`c(...)` というのは R でベクトルを意味します。`1:10 / 100` は、`1:10` の各成分を 100 で割ることを意味します。つまり以下の二つは同じ意味です。

```
a <- 1:10 / 100
a <- c(0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1)
```

4 図を拡大

前章ではわけの分らない図しか表示されませんでした．しかし，この一部をアップして表示してみると，正弦波になっていることがわかります．ここでは図の拡大の方法を試します．

前章で作った信号 x は長さ 16000 のベクトルですが、以下のようなコマンドで最初の 100 要素だけを見ることができます。

```
plot(x[1:100])
```

違う部分を見たい場合は $[1:100]$ の代わりに $[1000:1100]$ などとすれば OK です．以下の Exercise を実際にいろいろ試してみてください．

Exercise

- x の最初の 10 要素を見るにはどうすればいいか
- x の最後の 100 要素を見るにはどうすればいいか
- `plot(x[1:100], type = "b")` とするとどうなるか．
- 同じく，`type = "l"` とするとどうなるか．
- 周波数が 1000 Hz の信号 y を作って，同様にいろいろな箇所を表示
- `x <- sin(2 * pi * f * t)` で，`sin` の代わりに `cos` や `tan` にした場合について，同様にいろいろな箇所を表示
- `tan` にした場合はよくわからない波形が表示されるはずです．縦軸に注目して，その理由について考えてみてください．

5 音を聴く

上で作った音 x がどんな音なのか聞いてみます。tuneR をロードしている状態で以下のように入力すると x を test.wav に保存することができます。

```
obj <- Wave(x, samp.rate = 16000, bit = 16)
writeWave( normalize (obj, unit = "1"), filename = "test.wav" )
```

test.wav を再生してみましょう。(鳴らない場合は PC のボリュームがどうなっているか試しにニコニコ動画に接続して調べてみてください。) これは 440Hz の正弦波です。

Excercise

- 先程作った x は 440Hz で長さ 3 秒の信号です。880Hz で長さ 3 秒の信号を作って同様に聞いてみてください。
- 220Hz, 長さ 5 秒
- 330Hz, 長さ 5 秒
- 660Hz, 長さ 5 秒
- ドレミファソラシド全部の音を作ってみましょう。それぞれの周波数は以下の通りの通りです。

音名	周波数	R での入力方法
ド	$440 \times 2^{-9/12}$ Hz	$440 * 2^{**(-9/12)}$
レ	$440 \times 2^{-7/12}$ Hz	$440 * 2^{**(-7/12)}$
ミ	$440 \times 2^{-5/12}$ Hz	$440 * 2^{**(-5/12)}$
ファ	$440 \times 2^{-4/12}$ Hz	$440 * 2^{**(-4/12)}$
ソ	$440 \times 2^{-2/12}$ Hz	$440 * 2^{**(-2/12)}$
ラ	440 Hz	440
シ	$440 \times 2^{2/12}$ Hz	$440 * 2^{**(2/12)}$

6 関数定義

いちいちコマンドを打つのは面倒なので、いろいろな処理をひとまとめにした関数 (function) を作ると便利です。たいていのプログラミング言語には関数を作る機能があります。R でも関数が作れます。

```
sig.gen <- function(freq, sec){
  fs <- 16000                      # サンプリングレート 16000 Hz
  t  <- 1:(sec * fs) / fs          # 時間.長さ sec 秒
  x  <- sin (2 * pi * freq * t)    # 信号
  return (x)
}

my.save <- function(signal, fn){
  obj <- Wave(signal, samp.rate = 16000, bit = 16)
  writeWave( normalize (obj, unit = "16"), filename = fn )
}
```

これらの関数を使うと、ドレミファソラシドのセーブは以下のように書けます。

```
my.save(sig.gen(440 * 2**(-9/12), 5), "do.wav")
my.save(sig.gen(440 * 2**(-7/12), 5), "re.wav")
my.save(sig.gen(440 * 2**(-5/12), 5), "mi.wav")
my.save(sig.gen(440 * 2**(-4/12), 5), "fa.wav")
my.save(sig.gen(440 * 2**(-2/12), 5), "sol.wav")
my.save(sig.gen(440 * 2**(0/12), 5), "la.wav")
my.save(sig.gen(440 * 2**(2/12), 5), "si.wav")
```

7 倍音を付加して様々な音色を作る

7.1 倍音とは何か？

ある周波数 f の整数倍の周波数 nf ($n = 2, 3, 4, \dots$) を, f の倍音 (harmonics) といいます。倍音に対して, f のことは基本周波数 (fundamental frequency) といいます。音楽などで使われる音は, おおむね基本周波数と倍音の重ね合わせによって表現される音が多いです。

音楽に限らず, 周期信号 $x(t - 1/f) = x(t)$ は, 基本周波数と倍音の重ね合わせによって表現されます。これはフーリエ級数 (Fourier series) と呼ばれます。

$$x(t) = \sum_{k=0}^{\infty} a_k \sin(2\pi k f t + \phi_k)$$

具体的には

$$x(t) = \text{定数} + a_1 \sin(2\pi f t + \phi_1) + a_2 \sin(2\pi 2 f t + \phi_2) + a_3 \sin(2\pi 3 f t + \phi_3) + a_4 \sin(2\pi 4 f t + \phi_4) + \dots$$

です。音楽信号の分析を考えるとときには, 左辺 $x(t)$ が与えられたときに, 右辺の a_k (振幅) と ϕ_k (位相) を求めることが問題になりますが, ここでは逆に, 右辺の a_k, ϕ_k を適当に決めて, 左辺の $x(t)$ をいろいろと生成してみることを考えてみましょう。

7.2 倍音を含む音を作る

前章の sig.gen を次のように改造します。具体的にいうと引数に位相 ϕ をとれるようにします。

```
sig.gen <- function(freq, sec, phi){  
  fs <- 16000 # サンプリングレート 16000 Hz  
  t <- 1:(sec * fs) / fs # 時間. 長さ sec 秒  
  x <- sin(2 * pi * freq * t + phi) # 信号  
  return(x)  
}
```

これを使って, 以下のような振幅と位相を持った信号を作ってみましょう。

$$a_k = \frac{1}{k}, \phi_k = 0$$

これは「のこぎり波」と呼ばれている信号です。

```
x <- 1 * sig.gen(440, 5, 0) +  
  0.5 * sig.gen(880, 5, 0) +  
  0.333 * sig.gen(1320, 5, 0) +  
  0.25 * sig.gen(1760, 5, 0) +  
  0.2 * sig.gen(2200, 5, 0) +  
  0.167 * sig.gen(2640, 5, 0) +  
  0.143 * sig.gen(3080, 5, 0) +  
  0.125 * sig.gen(3520, 5, 0) +  
  0.111 * sig.gen(3960, 5, 0)
```

波形を見てみましょう。

```
plot(x[1:100], type = "b")
```

音を聞いてみましょう．

```
my.save(x, "nokogiri.wav")
```

上の定義では第 9 倍音まで考えていましたが，今度は第 18 倍音まで含めた信号を作ってみましょう．for 文を使うと便利です．

```
x <- sig.gen(440, 5, 0)
for(k in 2:18){
  x <- x + sig.gen(440 * k, 5, 0) / k
}
```

その波形を見て，音を聞いてみましょう．

Excercise 第 30 倍音まで含めるとどのような音になるか？ 標本化定理 ♠

Excercise その 2

$$a_k = \begin{cases} \frac{1}{k}, & k \text{ は奇数} \\ 0, & k \text{ は偶数} \end{cases}, \phi_k = 0$$

としてみるとどうなるか？ (for 文を使って書いてみる) 矩形波 ♠

Excercise その 3 a_k, ϕ_k を自分の好きなように適当に決めていろいろな音を作ってみてください．