コンピュータ基礎演習

追加課題(2) 音の利用

理工学部 情報科学科 隅田 康明

sumida@ip.kyusan-u.ac.jp

追加課題について

- 本来の授業内容から削った内容をいくつか出題
  - 14回授業が13回になったことの補填
  - 遠隔授業により削減した内容の提示

- 制作課題まで提出して、余裕があれば取り組みましょう
  - あくまで、やる気のある学生向けの内容

- いくつか出題する追加課題のうち、1つを提出すれば加点
  - 追加課題の提出で最大+5点
  - ただし、合計の評点は100点を上限とする

想定動作端末

- 追加課題は大学のパソコン教室等での動作を想定
  - 所持端末によっては動作しないものもあるので、動作するものを選んで作業すること

追加課題２
マルチメディア：音の利用

スマートフォンでは難しいので、
やってみたい場合は大学の自習室等で

Processingで音声ファイルを扱う

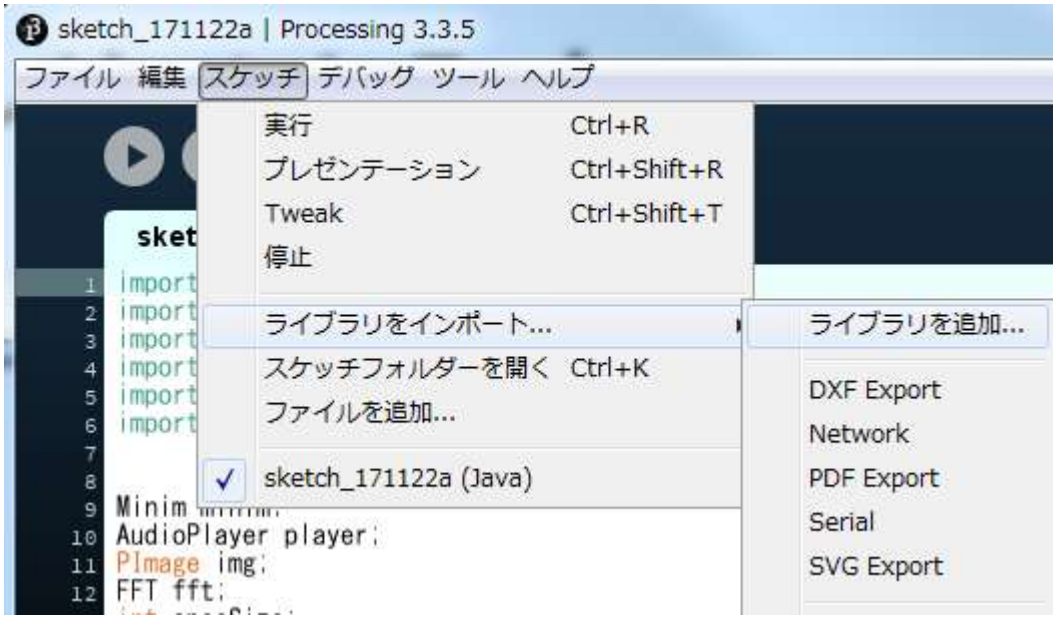
- 音楽を再生する
  - Minimライブラリを利用する
- 音声ファイルの情報を表示する
- 音に連動して図形を動かす

この課題に取り組む上での注意

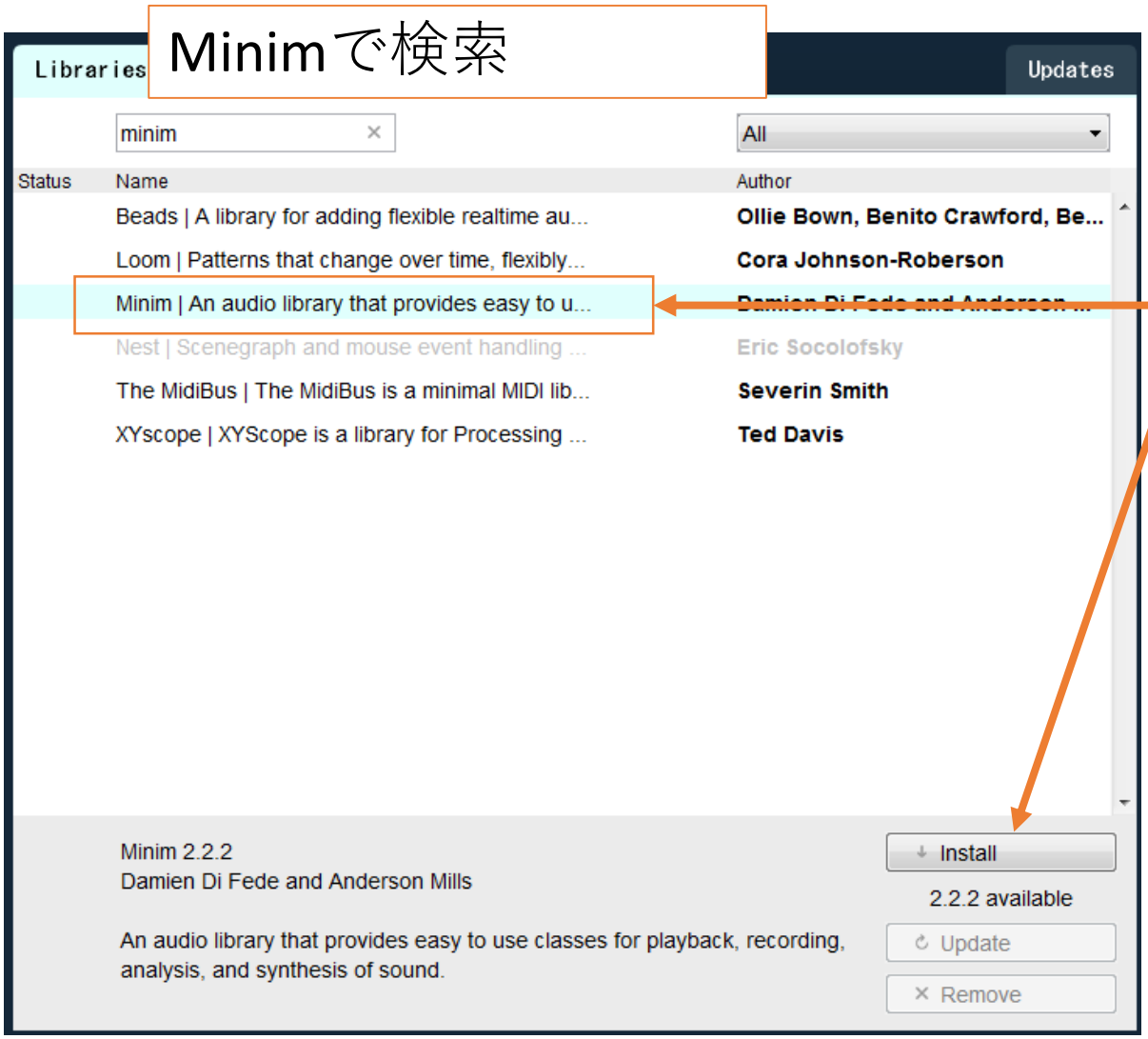
- 説明をかなり省いています
  - FFT解析などは説明しても難しいため
- 取り合えず、音を鳴らす、程度なら簡単
- サンプルを変えたら動きが変わる、面白い、程度で遊んでみましょう

Minimライブラリをインポート（１）

- [スケッチ]⇒[ライブラリをインポート…]
  ⇒[ライブラリを追加]



Minimライブラリをインポート（２）

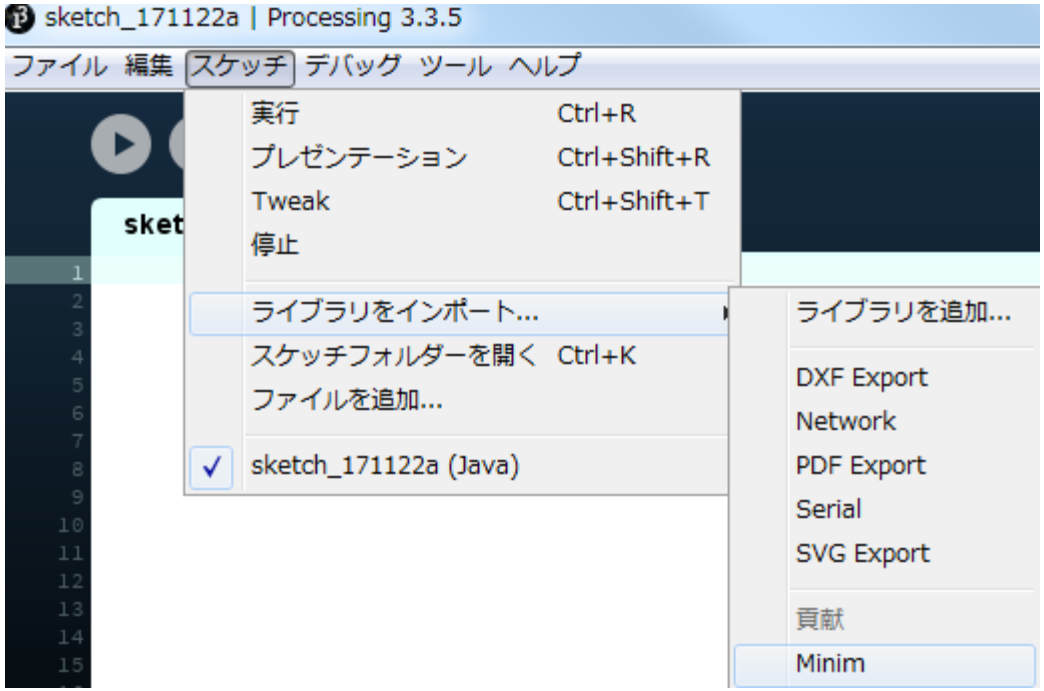


Minim | An audio library・・・

を選択してInstall

Minimライブラリをインポート（３）

- [スケッチ]⇒[ライブラリをインポート...]
  ⇒[Minim]



自動で挿入される

ライブラリのインポートの補足

- パソコン教室のPCでは、
  ログイン時にインポートしたライブラリが消える
  - 次回Minimライブラリを使う場合には、
    もう一度インストールしないといけない

※普通のPCでは、
一度インストールしたライブラリは消えない

ライブラリとは

- 再利用可能な形でまとめられたプログラム群
  - ライブラリは「利用」する目的で作られ、単体では動作しない

- 自作メソッドを他のプログラムに持っていくのに近く、
更に色々なプログラム、多くの人に利用可能な形にしたものがライブラリ

- 要するに、便利な道具の集まり

Minimを利用して音楽を再生する

```
//ddf.minimのインポート文
Minim minim;
AudioPlayer music;

void setup() {
  size(400, 400);
  minim = new Minim(this);
  String s = "http://www.is.kyusan-u.ac.jp/~goshi/d/irish.mp3";
  //レポート提出時には↑に変えておくように
  music = minim.loadFile(s, 512);
  music.play();
}
void draw() {
}
```

再生する音声ファイルを入力

loadFile()で読み込む
2048はバッファサイズ

play()で再生

今の音源データの紹介

http://hirofuminakamura.com/

- 提供してくれた方：中村大史さん
- 曲名：Reels
  - tricolor, John John Festival, O'Jizo 等のアイルランド音楽のバンドメンバーとして活躍

http://tricolor-web.com/

http://tricolor-web.com/

Minim.loadfile()

```
music = minim.loadFile(s, 2048);
```

- 音声ファイルを扱うときには、
  音声ファイルをメモリに読み込まないといけない
- 一気に全部読み込むと、メモリを圧迫するので、
  再生しながら少しずつ読み込んでいく
  - 一度に読み込むデータ量がバッファサイズ

音楽を再生する

- MP3データを読み込んで再生

```
//ddf.minimのインポート文
Minim minim;
AudioPlayer music;
String s;

void setup() {
  size(400, 400);
  minim = new Minim(this);
  s = “music.mp3";
  music = minim.loadFile(s, 2048);
  music.play();
}
void draw() {
}
```

音楽を再生する

- **[sketch]⇒[ファイルを追加]** で音声ファイルを選択

- スケッチフォルダ内に[data]フォルダが出来て、
  その中に音声ファイルが追加される

- 音声ファイルの名前を[music.mp3]に変更
  - レポート確認のため
  - 自分で作る場合には、そのままのファイル名で良い

各自で好きな音楽を再生しよう

- 著作権フリーのBGMなどで検索し、気に入った音楽をダウンロード、music.mp3に名前を変更
  - 大きすぎるファイルは再生出来ない事もあるので注意

```
//使用楽曲（～～ : http://～～～）
//ddf.minimのインポート文
Minim minim;
AudioPlayer music;

void setup() {
  size(400, 400);
  minim = new Minim(this);
  String s = "music.mp3";
  music = minim.loadFile(s, 2048);
  music.play();
}
void draw() {
}
```

プログラムが終わるときに
音声ファイルを閉じる

```
void stop() {
  music.stop();
  super.stop();
}
```

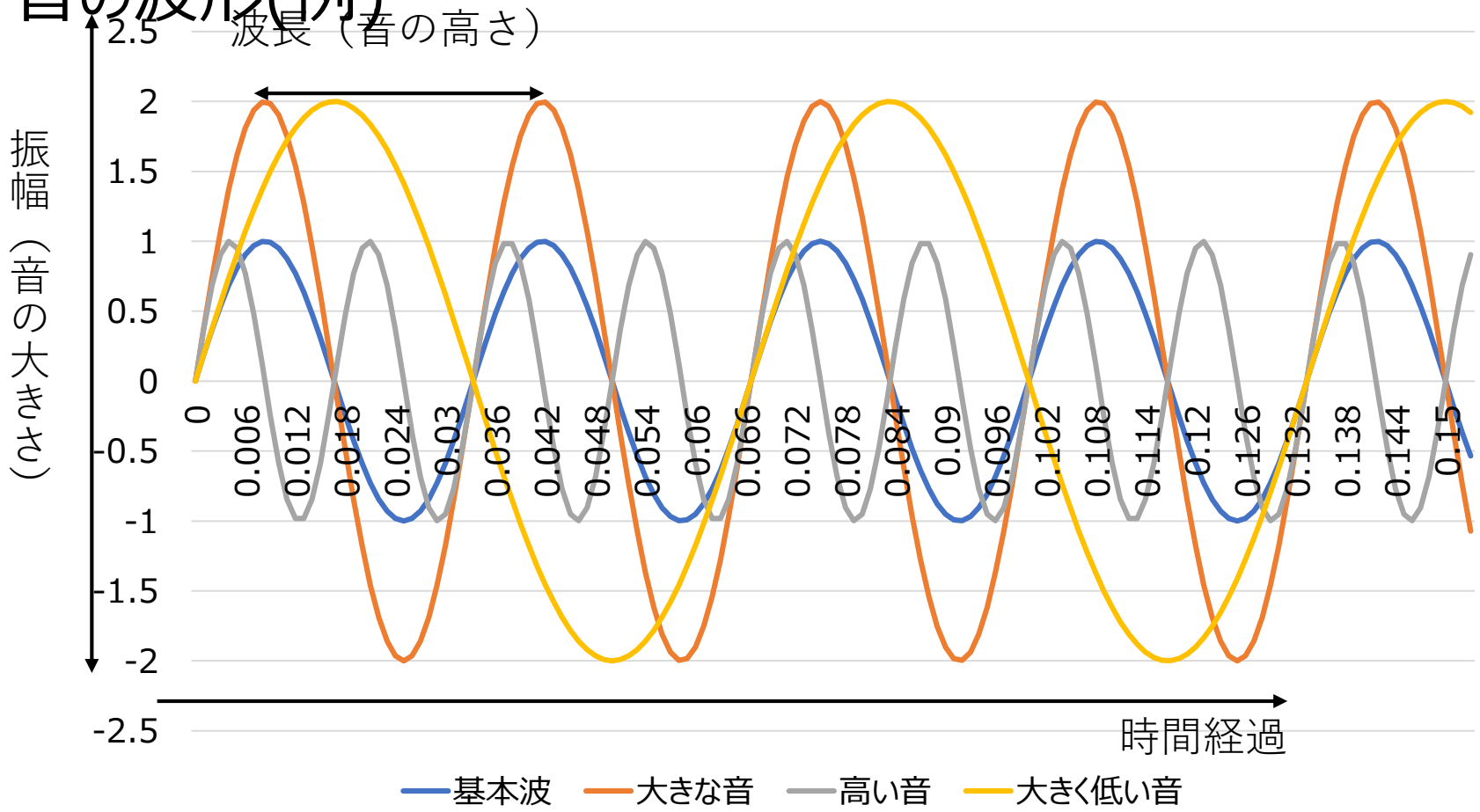
参考：マイク入力

- music = minim.loadFile(s, 512);
  の代わりに、
  music = minim.getLineIn(Minim.STEREO, 512);

- マイクの音量を利用したプログラムを作れる
  - マイクがなければエラーになる

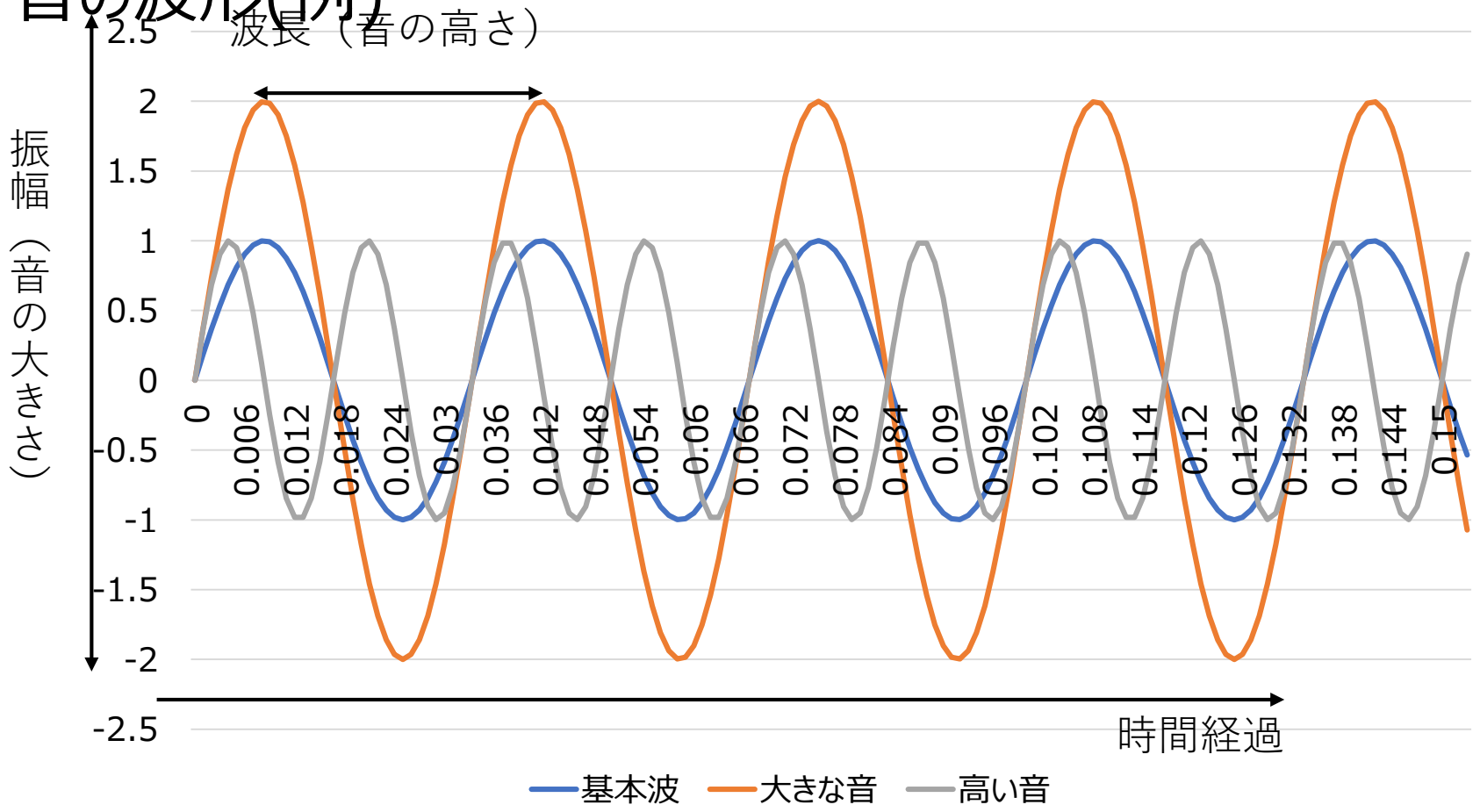
音声の情報化

- 音の波形(例)



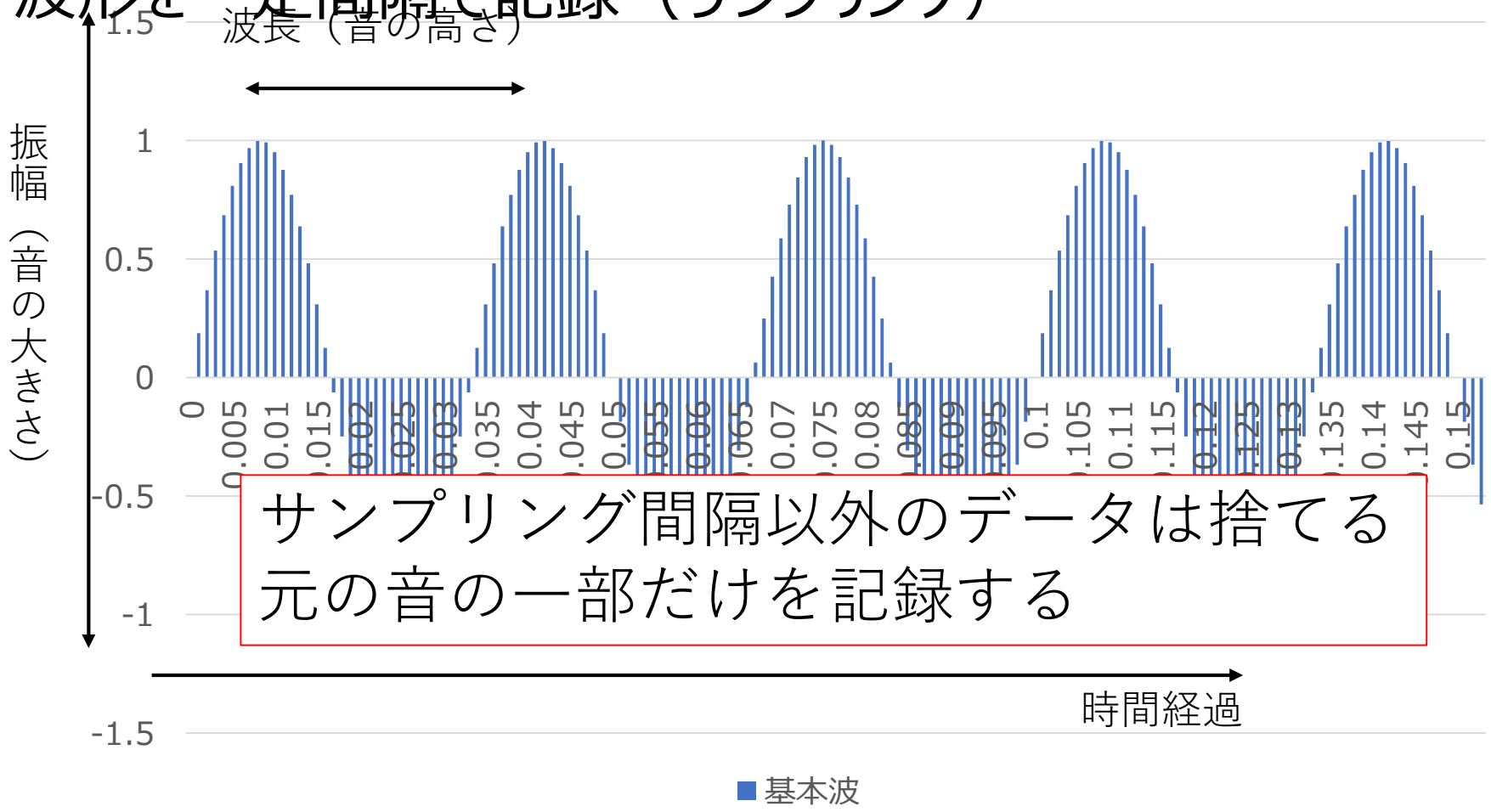
音声の情報化

- 音の波形(例)



音声の情報化（デジタル化）

- 波形を一定間隔で記録（サンプリング）



サンプリング間隔以外のデータは捨てる
元の音の一部だけを記録する

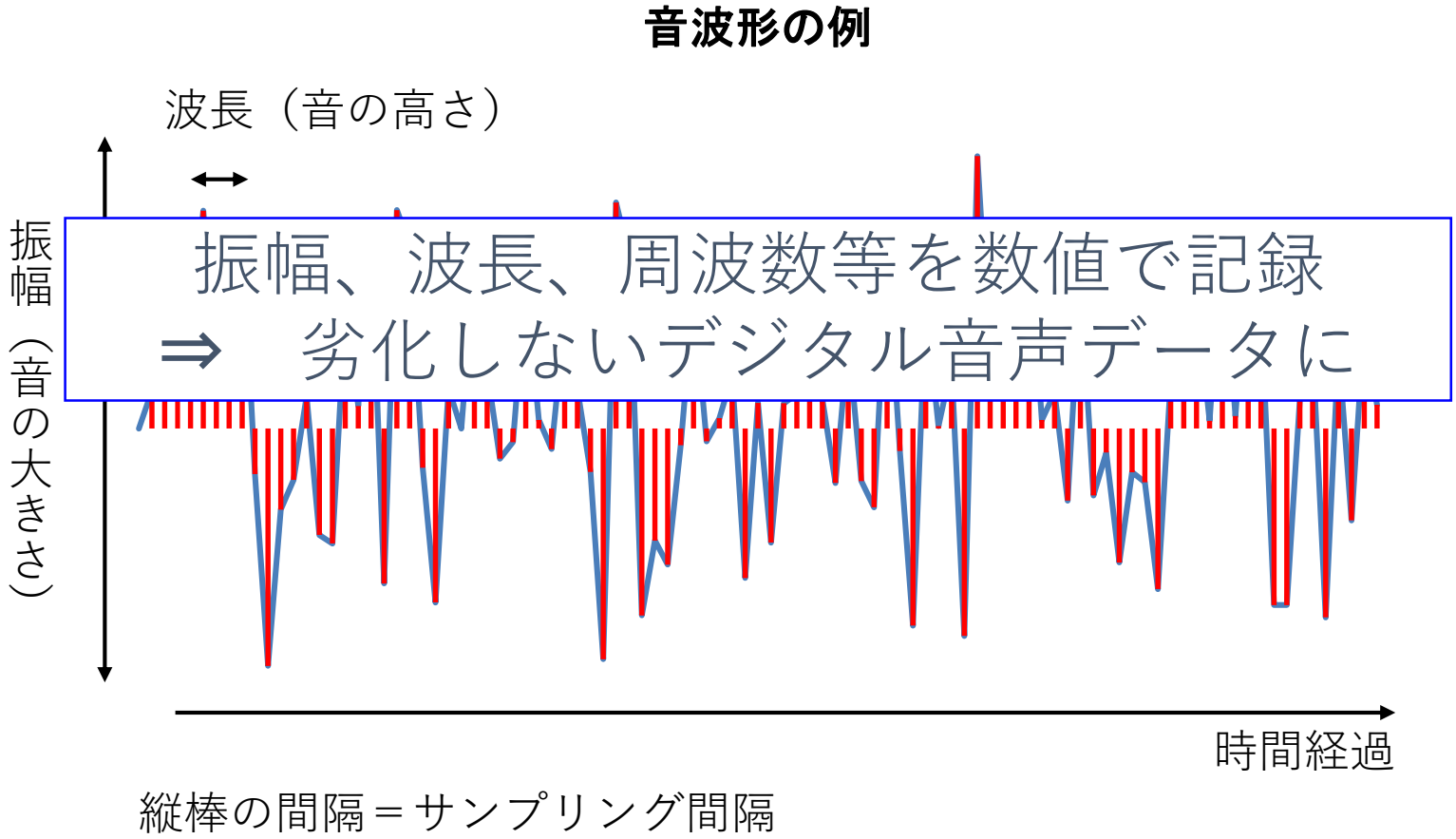
アナログとデジタル

- アナログ：連続量
  - アナログ時計：針の位置で時間が決まる
    - はっきりと何時何分何秒は分からない

- デジタル：離散量
  - デジタル時計：数字で時間が分かる
    - はっきりと何時何分何秒か分かる
    - 表示されていない桁は分からない

音声のデジタル化

- サンプリング間の情報は失われる



アナログ⇔デジタル変換

- アナログ→デジタル
  - 元の情報を飛び飛びに数値化
  - 利点：情報が劣化しない
  - 欠点：サンプリング間の情報が失われる
    - デジタル化によって、情報が変質する
    - 限りなく細かくサンプリングすることで、アナログに近いデジタル化は可能

- デジタル→アナログ
  - 現実世界はアナログの世界
  - 情報を人が見えるようにする→アナログ化
  - 音声：スピーカーで出力
  - 画像：プリンタで印刷

デジタル化の利点

- 情報が劣化しない
  - 長期的な保存が可能
    - アナログデータは劣化する
      - 音声ノイズ、映像ノイズ
      - 写真や絵は時間がたつと色褪せる
  - 劣化しないデータの伝送が可能
    - 離れた場所に元と全く同じデータを送れる
  - 完全な複製が可能
    - 違法コピーなどの問題にも

音の情報を取得する

- 音の大きさを取得する（ステレオの左右）
  - music.left ：左の音
  - music.right ：右の音

```
float left = music.left.get(i);
float right = music.right.get(i);
```

i はバッファに入っている音声データの番号
（配列のような形で音のデータが入っている）

```
//左右の音量を円で表現
float left = 0;
 float right = 0;
 for(int i = 0; i < music.bufferSize() - 1; i++){
  left += abs(music.left.get(i));  //バッファ分の音量を全部足す
  right += abs(music.right.get(i));
 }
 left = 1000*left/music. bufferSize();  //平均値を計算して1000倍(最大値)を掛ける
 right = 1000*right/music. bufferSize();
 ellipse(width/3, height/2, left, left);
 ellipse(width*2/3, height/2, right, right);
```

```
//左右の音量を線で描く
void draw() {
 background(128);
 stroke(255);
 for(int i = 0; i < music.bufferSize() - 1; i++){
  line(i, 50 + music.left.get(i) * 50, i + 1,  50 + music.left.get(i + 1) * 50);
  line(i, 150 + music.right.get(i) * 50, i + 1,  150 + music.right.get(i + 1) * 50);
 }
}
```

Leftとrightの大きさ

- left.get(i)とright.get(i)
  - 返り値：-1～+1の小数
  - バッファ分の平均値を計算している
- 利用したいものの最大値に合わせて掛ける数を調整
  - 最大100なら ＊100、360なら ＊360
  - ただし、元の音量が小さい場合には最大値より大きな数を掛ける
    - 1に近い値は殆ど出てこない
- println(left + “, “ + right); を実行すると、数値をテキストコンソールで確認できる

周波数スペクトルの表示

- 音声データは音の波を数値化して記録している
- 波を解析すると、色々なことが分かる
  ⇒　波の解析：FFT（高速フーリエ変換）など

- Processingで音をFFT解析
  - 周波数ごとの振幅を調べられる

FFT解析：高速フーリエ変換

- setup()の上に
  FFT fft;
  を追加
- setup()の最後に
  fft = new FFT(music.bufferSize(), music.sampleRate());
  を追加

```
Minim minim;
AudioPlayer music;
float r2 = 1;
int count = 0;
FFT fft;  //高速フーリエ変換をするための変数
void setup() {
  size(400, 400);
  minim = new Minim(this);
  String s = "music.mp3";
  music = minim.loadFile(s, 512);  //loadFile("音声ファイル", バッファサイズ)
  music.play();
  fft = new FFT( music.bufferSize(), music.sampleRate() );  //音声ファイルをFFTにセット
}
void draw() {
  background(0);
  stroke(255);
  fft.forward( music.mix );    //FFTを実行
  for (int i = 0; i < fft.specSize(); i++) {  //周波数ごとに線描く
    line(i*4, height, i*4, height - fft.getBand(i)*4);  //各周波数の大きさで線を描く
  }
}
```

バッファサイズと繰り返し回数に注意

- バッファサイズは2の累乗の数のいずれか
  - 2,4,8,16,32,64,128,256,512,1024,2048

```
music = minim.loadFile(s, 512);  //loadFile("音声ファイル", バッファサイズ)
```

- 繰り返し回数はfft.specSize()か、バッファサイズ以下の回数

```
for (int i = 0; i < fft.specSize(); i++) {  //周波数ごとに線描く
   line(i*4, height, i*4, height - fft.getBand(i)*4);  //各周波数の大きさで線を描く
 }
```

- バッファサイズを超えるとエラーになる

雛型で配布

アレンジ例：
音量に応じて動きが変わるジェネラティブアート

```
import ddf.minim.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
import ddf.minim.signals.*;
import ddf.minim.spi.*;
import ddf.minim.ugens.*;

int x = 50;
int y = 50;
float xnoise = random(1);
float ynoise = random(1);
int count = 0;
int sx, sy;
Minim minim;
AudioPlayer music;
void setup() {
  size(300, 300);
  stroke(random(255), random(255), random(255), 30);
  sx = width/2;
  sy = height/2;
  minim = new Minim(this);
  String s = "music.mp3";
  music = minim.loadFile(s, 2048);  //loadFile("音声ファイル", バッファサイズ)
  music.play();
  background(0);
}
void draw() {
  float left = 0;
  float right = 0;
  for(int i = 0; i < music.bufferSize() - 1; i++){
    left += abs(music.left.get(i));
    right += abs(music.right.get(i));
  }
  left = left/music.bufferSize()/50;
  right = right/music.bufferSize()/50;

  println(left + "  " + right );

  stroke(random(255), random(255), random(255), 100);
  x = (int)(noise(xnoise) * width);
  y = (int)(noise(ynoise) * height);
  line(sx, sy, x, y);
  xnoise += left;
  ynoise += right;

  fill(0,5);
  rect(0,0,width, height);
}
```

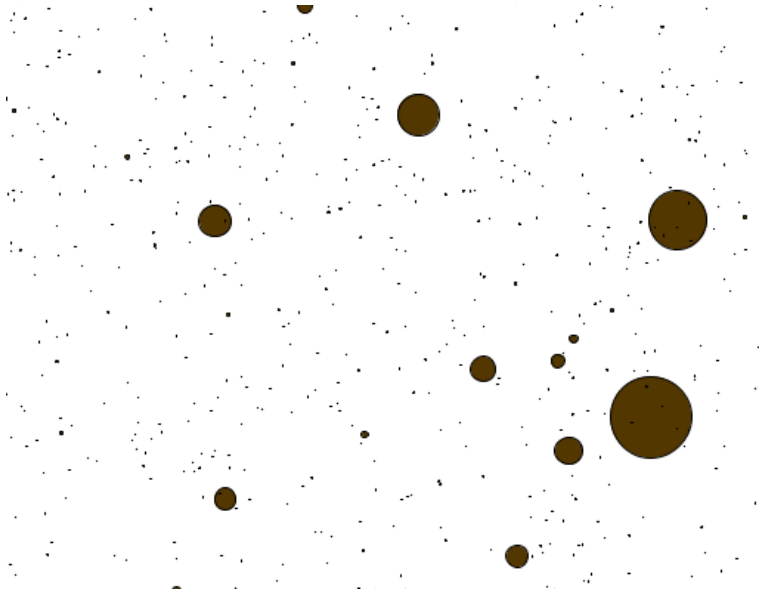
応用例：跳ね変える図形に応用

1. 沢山の跳ね返る図形のプログラムを用意
    - コピーして新しいスケッチに貼り付けておく

雛型で配布（さすがに長すぎる）


```
import ddf.minim.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
import ddf.minim.signals.*;
import ddf.minim.spi.*;
import ddf.minim.ugens.*;
Minim minim;
AudioPlayer music;
FFT fft;  //高速フーリエ変換をするための変数
float [] x = new float[512];  //バッファサイズより小さく
float [] y = new float[512];  //バッファサイズより小さく
float [] vx = new float[512];  //バッファサイズより小さく
float [] vy = new float[512];  //バッファサイズより小さく
void setup() {
 size(512, 400);
 for (int i = 0; i < x.length; i++) {
  x[i] = random(width);
  y[i] = random(height);
  vx[i] = random(3)-1.5;
  vy[i] = random(3)-1.5;
 }
 minim = new Minim(this);
 String s = "http://www.is.kyusan-u.ac.jp/~goshi/d/irish.mp3";
 music = minim.loadFile(s, 512);  //loadFile("音声ファイル", バッファサイズ)
 music.play();
 fft = new FFT( music.bufferSize(), music.sampleRate() );  //音声ファイルをFFTにセット
}
void draw() {
 background(255);
 float right = 0;
 float left = 0;
 for (int i = 0; i < music.bufferSize() - 1; i++) {
  left += abs(music.left.get(i));
  right += abs(music.right.get(i));
 }
 left = 1000*left/music. bufferSize();
 right = 1000*right/music. bufferSize();
 fft.forward( music.mix );   //FFTを実行
 for (int i = 0; i < x.length; i++) {
  fill(left, right, 0);
  ellipse(x[i], y[i], fft.getBand(i), fft.getBand(i));
  x[i] += vx[i];
  y[i] += vy[i];
  if (x[i] < 0 || x[i] > width) {
   vx[i] *= -1;
  }
  if (y[i] < 0 || y[i] > height) {
   vy[i] *= -1;
  }
 }
}
```

長くなっているが、基本的には第10回と合わせただけ



追加課題２のレポート

- 雛型のどちらかをアレンジしてレポートを提出
  - 雛型を使わなくてもいい