

平成28年度 日本大学文理学部オープンキャンパス 特別講義

音声認識の仕組みを知ろう

日本大学 文理学部 情報科学科 准教授

北原 鉄朗

kitahara@chs.nihon-u.ac.jp

<http://www.kthrlab.jp/>

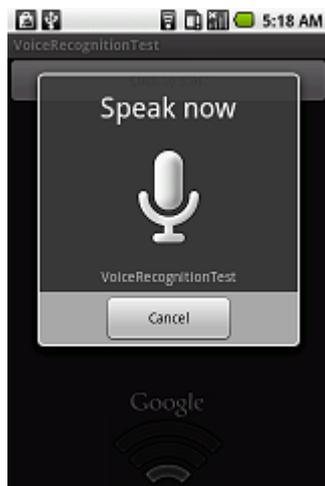
Twitter: @tetsurokitahara

皆さんに実際に試していただけるように
操作方法などを詳しく説明していますが、
時間と施設の制約から、講義中に皆さんに
お試しいただく機会をつくることができません。
講義内で用いるソフトウェアは、私のWebサイト
(<http://www.kthrlab.jp/>)からダウンロードできる
ようにしてありますので、ぜひ帰宅後に
お試しください。

音の認識と合成

音の認識

音 → 文字など



音声認識

声による
個人認証



音の合成

文字など → 音



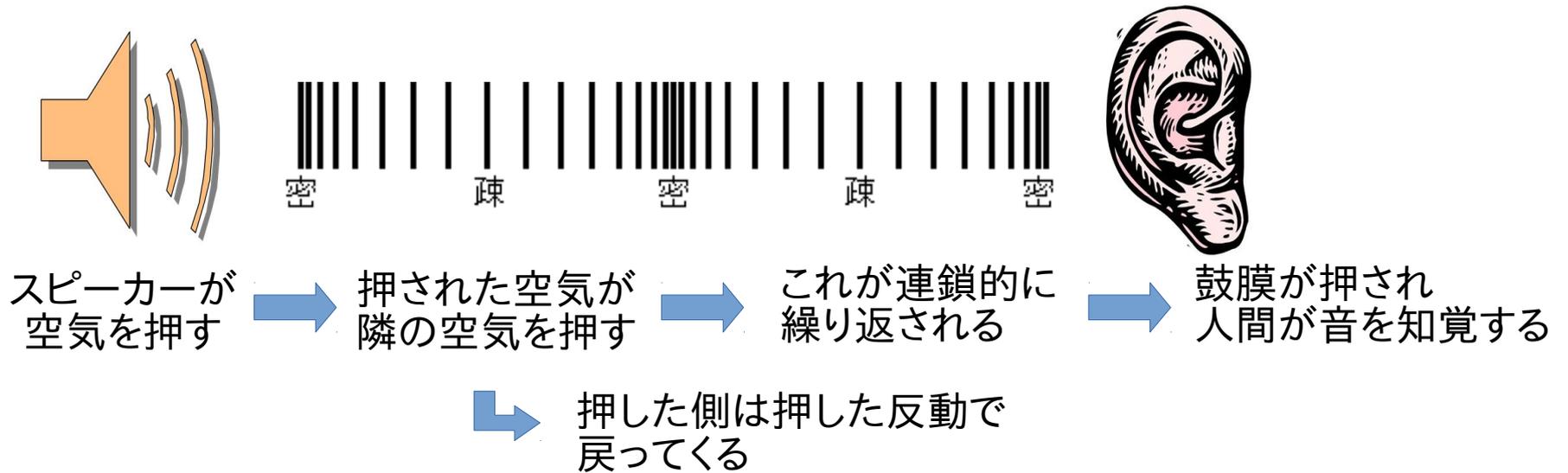
音声案内
(カーナビなど)

歌声合成
(Vocaloidなど)



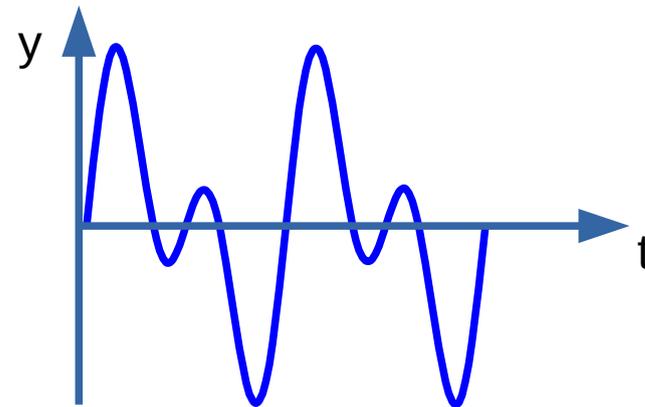
音声認識・音声合成の基本的な仕組みを知ろう

そもそも音って何？



このような、ものが行ったり来たりする運動が、遥か遠くまで伝わる現象を**波動**（または**波**）という

スピーカーが空気をどのくらい押すのかを**波形**として表す

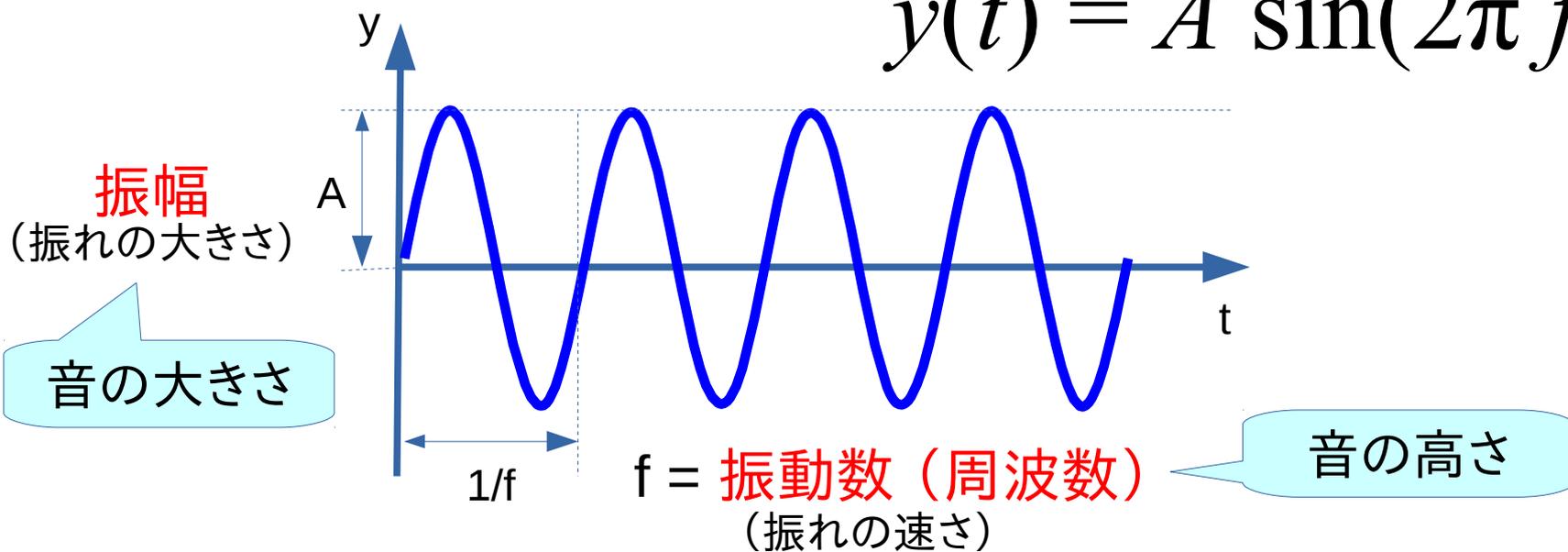


参考

- <http://www.wakariyasui.sakura.ne.jp/2-2-0-0/2-2-1-1onnpa.html>
- <http://www.ne.jp/asahi/tokyo/nkgw/gakusyuu/hadou/tate-yoko-wave/wave1.html>

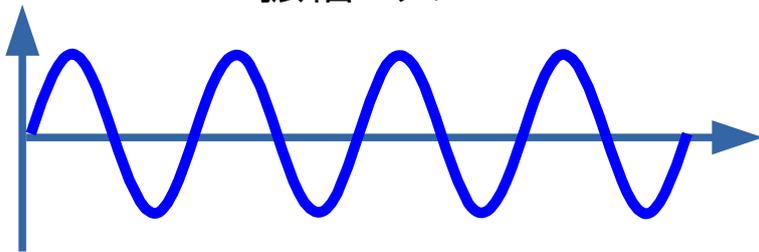
音の波形を数式で表してみよう

$$y(t) = A \sin(2\pi f t)$$

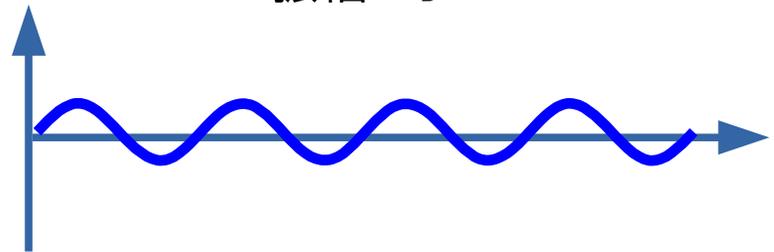


振幅=大

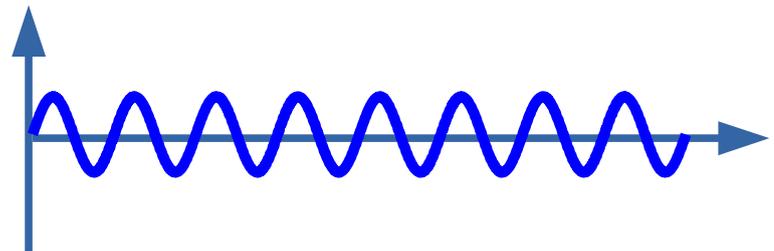
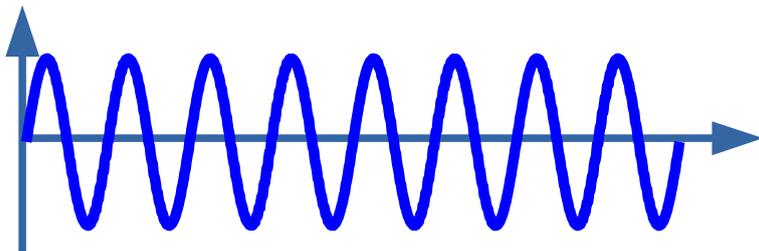
周波数 || 低



振幅=小



周波数 || 高



プログラミング用ソフトウェア「FreeMat」を 起動してみよう

The image shows a Windows desktop environment. On the left, a file explorer window is open to the 'FreeMatPortable' folder. The folder contains subfolders 'App', 'Data', and 'FreeMat'. The 'FreeMatPortable' folder is circled in orange, and an arrow points from it to the 'FreeMat v4.0 Command Window' on the right. The desktop background is dark blue with icons for 'お客様満足度アンケート', '初期設定ツール', 'WinSCP', and 'FreeMatPortable'. The taskbar at the bottom shows the Start button, Internet Explorer, File Explorer, and Firefox.

FreeMat v4.0 Command Window

File Edit Debug Tools Help

Stack: base C:/Users/kitahara/Desktop/FreeMatPortable/Data

File Browser

Filename	Size	Date
.. (Parent...		
analyze.m	107	24 No
maketone...	150	24 No

History

```
%% 火 11 24 23:38:14 2015
%% 水 11 25 19:48:21 2015
mysynth
```

Variables

Name	Class	Value
ans	double	

Debug

```
19:49:41.636: JIT compile failed:"JIT co
complex, string, END, matrix or cell def
"C:/Users/kitahara/Desktop/FreeMatPo
```

Ready

「音の波形を作るプログラム」を作ってみよう

計算機にして欲しいことを書き並べたものが「プログラム」。

先ほどの式 $y(t) = A \sin(2\pi f t)$ を使って音の波形を作る

プログラムを作ってみましょう。 ← 実際は作成済みで、読み込むだけ

The screenshot shows the FreeMat v4.0 interface. On the left, the File Browser shows a folder icon circled in orange. An arrow points from this icon to a file explorer window in the center. In this window, the file 'maketone.m' is selected and circled in orange. Another arrow points from this file to the main editor window on the right. The editor window shows the code for the 'maketone' function, with the text 'プログラムが読み込まれる' (Program is loaded) in an orange box at the bottom right.

maketone.m を選ぶ

```
function maketone
t = 0 : 1/48000 : 1;
f1 = 440;
a1 = 0.1;
y = a1 * sin(2 * pi * f1 * t);
plot(t(1:1000), y(1:1000));
wavplay(y, 48000);
```

プログラムが読み込まれる



解説

```
function maketone
```

↑ maketoneという名前のプログラムを作ります。

```
t = 0 : 1/48000 : 1;
```

↑ tの値は、1/48000[秒]刻みで0[秒]から1[秒]までとします。

```
f = 440; ← 周波数は440[Hz]とします。
```

```
a = 0.1; ← 振幅は、音が割れるぎりぎりの振幅の0.1倍とします。
```

```
y = a * sin(2 * pi * f * t);
```

↑ yの値を、 $y(t) = A \sin(2\pi f t)$ で計算します。

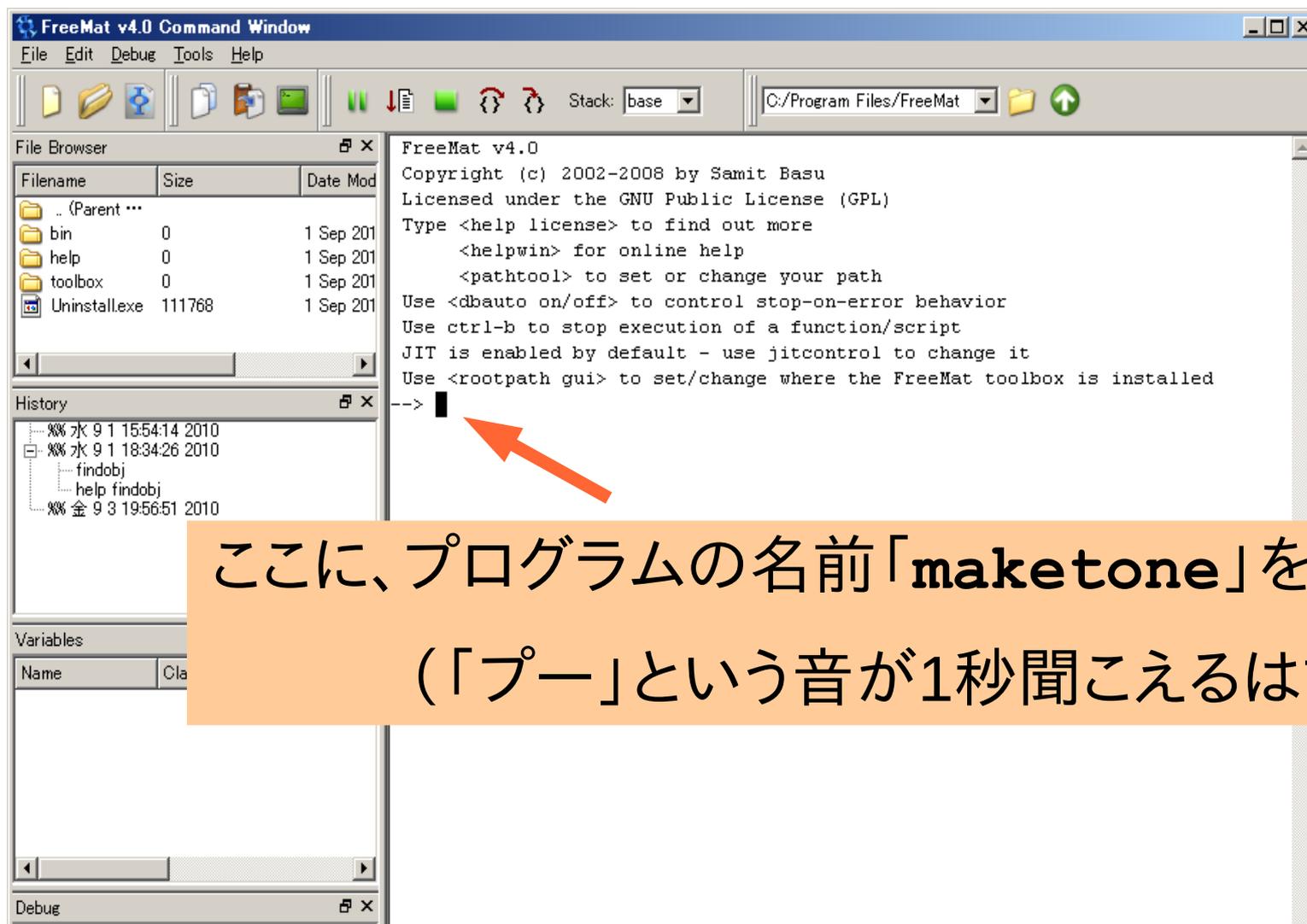
```
plot(t(1:1000), y(1:1000));
```

↑ $y(t) = A \sin(2\pi f t)$ のグラフを1～1000番めの点だけ描画します。

```
wavplay(y, 48000);
```

↑ 作った波形をスピーカーから鳴らします。

プログラムを実行してみよう



ここに、プログラムの名前「**maketone**」を打ち込む。
(「プー」という音が1秒聞こえるはず)



注意

何も聞こえないときは、ヘッドフォンの音量設定が0になっている可能性があります。確認しましょう。

2つの音を出してみよう

2つの音からなる和音を作ってみましょう。

プログラムを以下のように書き換えて実行してみよう。

```
function maketone

t = 0 : 1/48000 : 1;
f1 = 440;
f2 = 660;
a1 = 0.1;
a2 = 0.1;
y = a1 * sin(2 * pi * f1 * t) + ...
    a2 * sin(2 * pi * f2 * t);
plot(t(1:1000), y(1:1000));
wavplay(y, 48000);
```

※下線部が変更点

周波数や振幅をいろいろ変えて、
きれいな和音やきたない和音を作ってみよう。

2つの音が1つになる!?

片方の周波数を440Hz、もう片方を880Hzにしてみましょう。

```
function maketone

t = 0 : 1/48000 : 1;
f1 = 440;
f2 = 880;
a1 = 0.1;
a2 = 0.1;
y = a1 * sin(2 * pi * f1 * t) + ...
    a2 * sin(2 * pi * f2 * t);
plot(t(1:1000), y(1:1000));
wavplay(y, 48000);
```

どうなりましたか？

f2がf1のちょうど2倍のときだけ、1つの音に聞こえるかも!?

f2を色々変えて、確かめてみましょう。

3つの音も1つになる!?

さきほどの音に、1320Hzの音も加えてみましょう。

```
function maketone

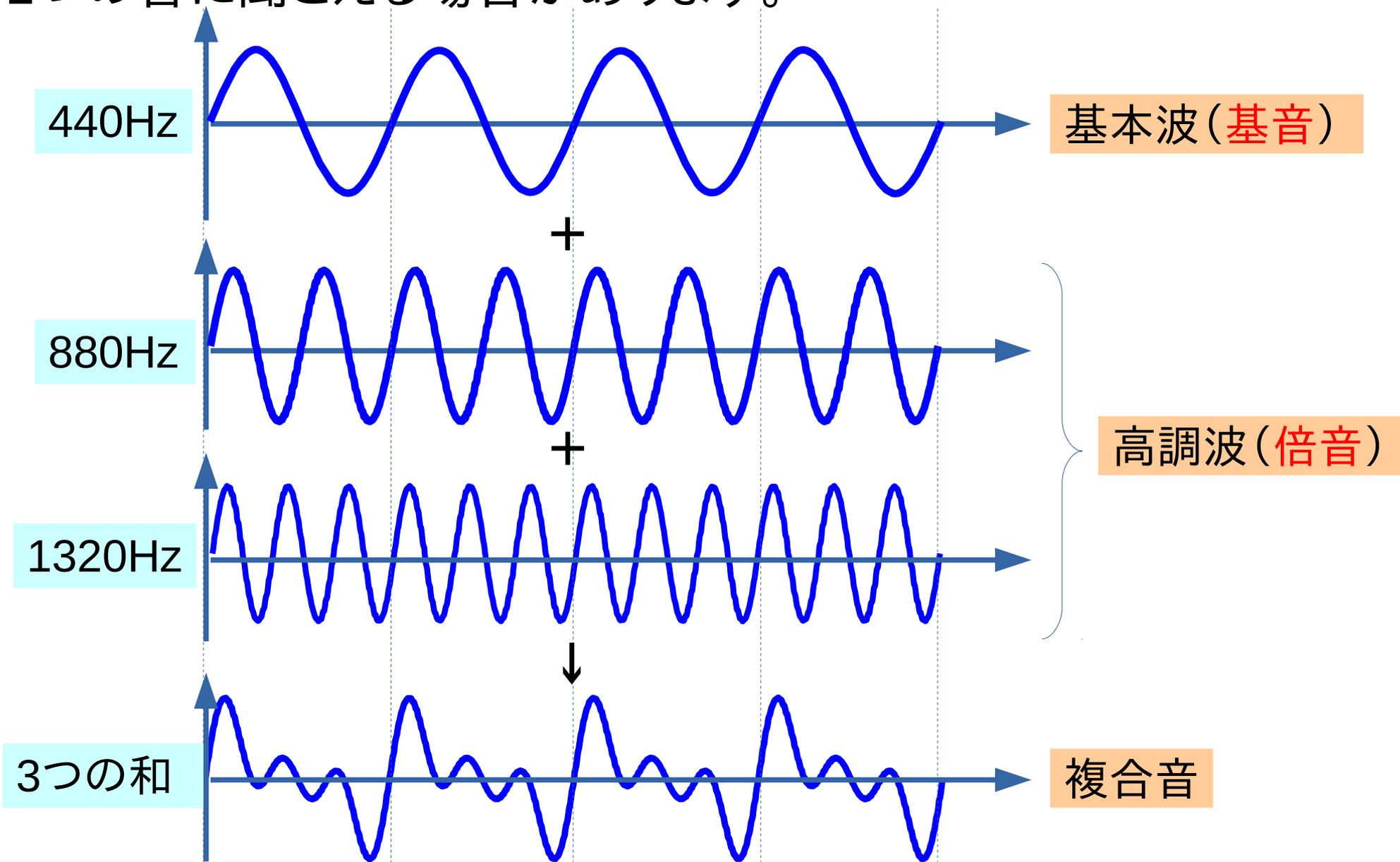
t = 0 : 1/48000 : 1;
f1 = 440;
f2 = 880;
f3 = 1320;
a1 = 0.1;
a2 = 0.1;
a3 = 0.1;
y = a1 * sin(2 * pi * f1 * t) + ...
    a2 * sin(2 * pi * f2 * t) + ...
    a3 * sin(2 * pi * f3 * t);
plot(t(1:1000), y(1:1000));
wavplay(y, 48000);
```

※下線部が変更点

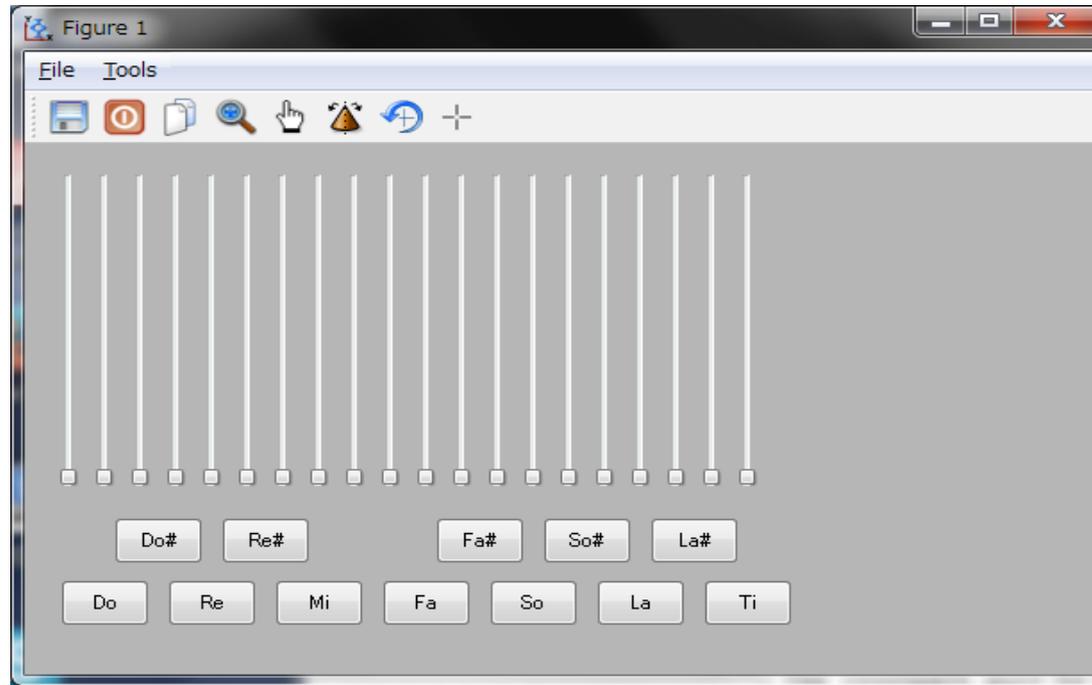
どうなりましたか? こちらも1つの音に聞こえましたか?

音に関する大事な性質

ある正弦波に、周波数がその整数倍の正弦波を足し合わせると、1つの音に聞こえる場合があります。



私が作ったプログラムでさらに試そう



- 1) コマンドを打ち込む場所に「**mysynth**」と入力して起動
- 2) 基音から20次倍音までの振幅（大きさ）を指定（=音色）
- 3) ボタンを押すと、その高さの音を合成
（内部でやってることは、先ほどのプログラムを一緒）

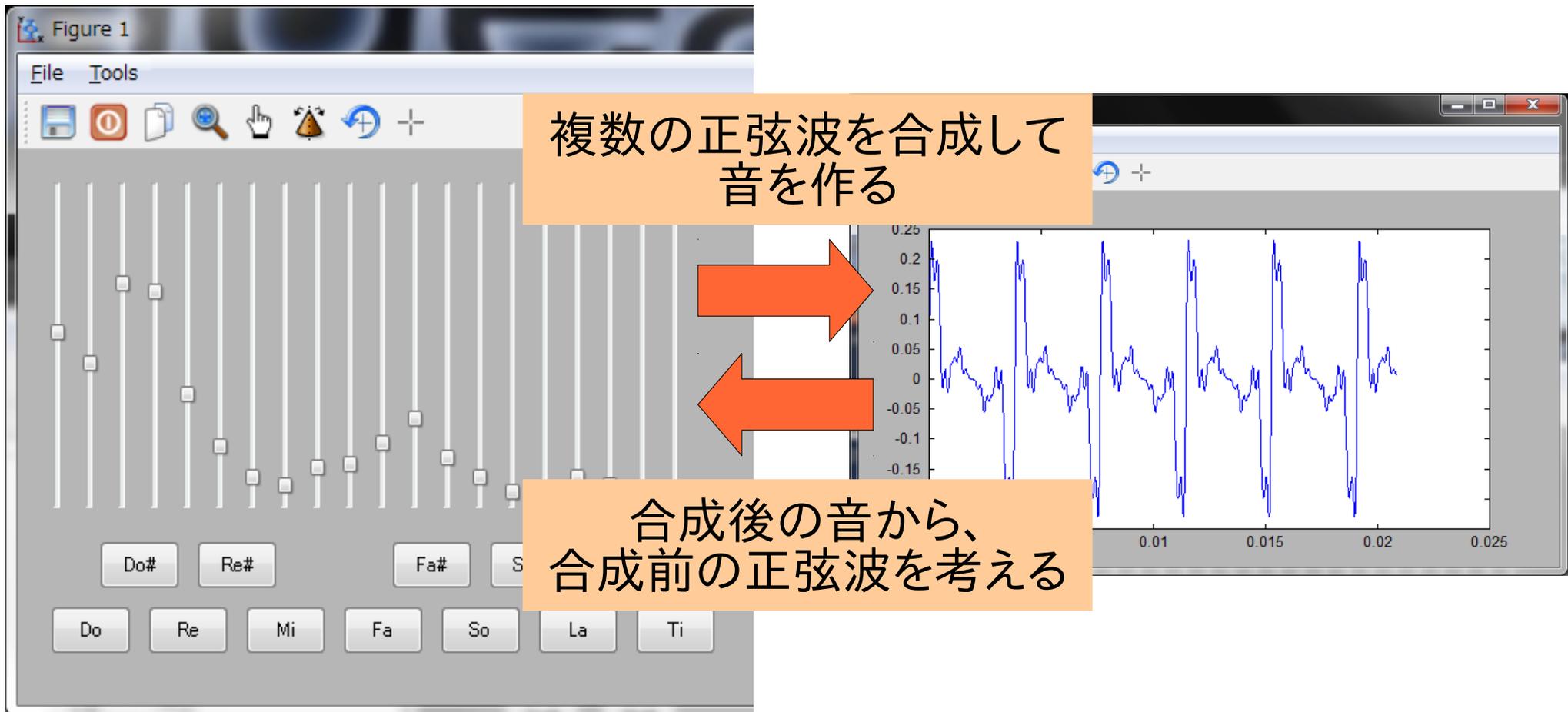
各倍音の振幅をいろいろ変えて試してみよう

音の合成から分析へ

周波数が整数倍の正弦波を足し合わせると、1つに聞こえる



1つ(に聞こえる)音も、周波数が整数倍の正弦波が
いくつも足し合わされてできている

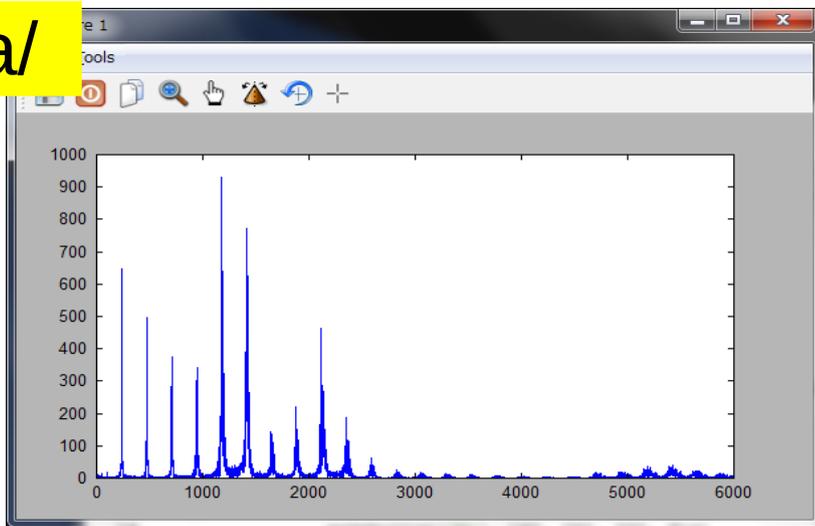


フーリエ変換

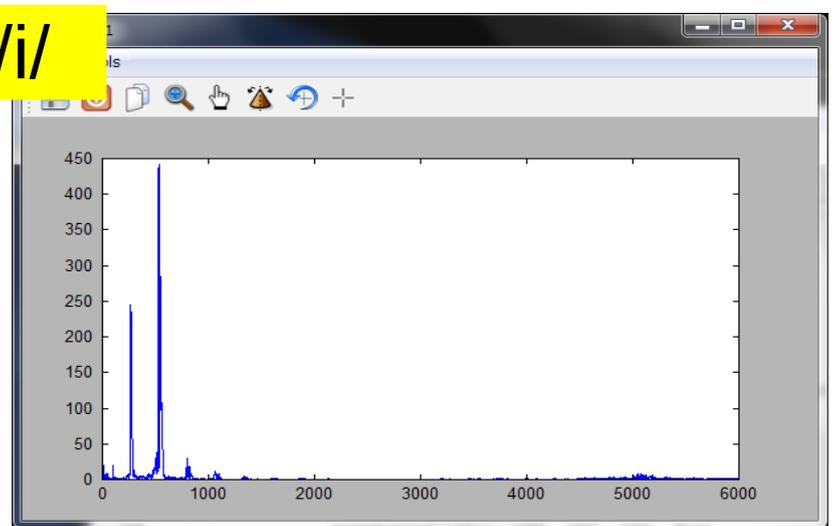
音を周波数ごとに分解する技術

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \quad \leftarrow \text{理解しなくていいです}$$

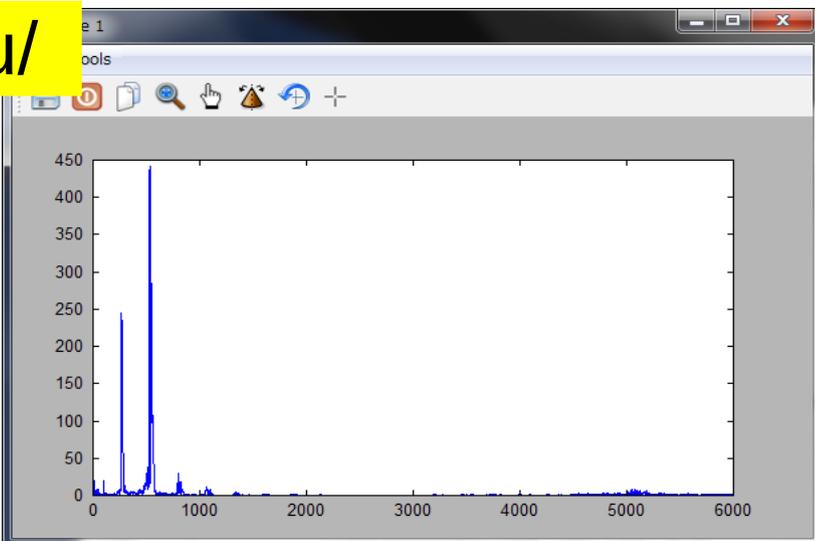
/a/



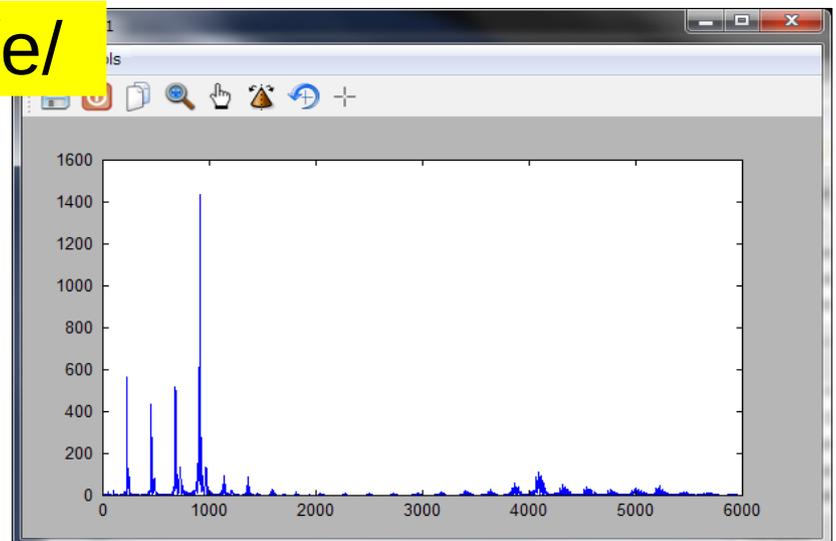
/i/



/u/



/e/



フーリエ変換を試そう

1.FreeMat のコマンドを打ち込む部分に

`analyze` と入力してプログラムを実行させよう

2.すぐに録音状態になるので、2秒間「あー」と言おう

3.フーリエ変換の結果がグラフとして表示されます

```
function analyze
```

```
y = wavrecord(2 * 16000, 16000);
```

```
Y = fft(y);
```

```
A = abs(Y);
```

```
plot(A(1 : 6000));
```

← 2秒間マイクから録音

← フーリエ変換を実行

← 複素数なので絶対値

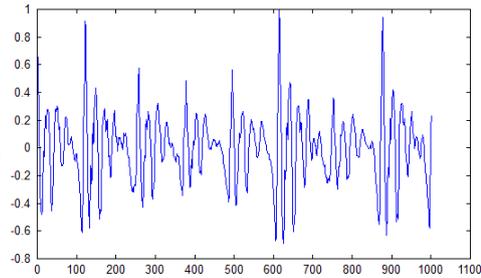
← 結果をグラフ表示

フーリエ変換のポイント

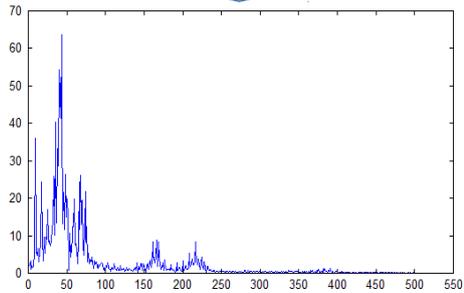
- どの周波数の倍音が強いかは、「あ」には「あ」に、「い」には「い」に特有の傾向がある(フォルマントという)
- フォルマントを人工的に再現して音を合成すれば、そのフォルマントの母音の音に聞こえるようになる(音声合成の基本)
- マイクから入力された音のフォルマントが、あらかじめ集めておいた「あ」の音のフォルマントと近ければ、その音は「あ」に違いないと判断できる(音声認識の基本)

実際にはもっと高度な値を求めるけど

音声認識の基本

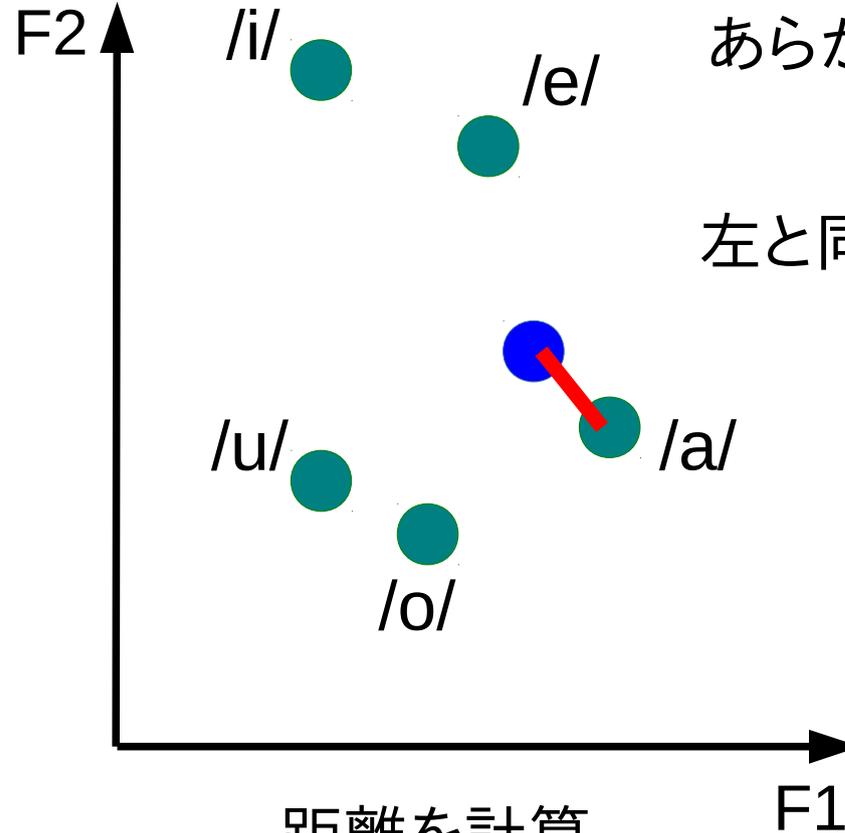


フーリエ変換



振幅が際立って
大きい周波数を探す

(実際はもっと複雑なことを行う)



/a/ ~ /o/ の音声を
あらかじめ用意

左と同じことをする

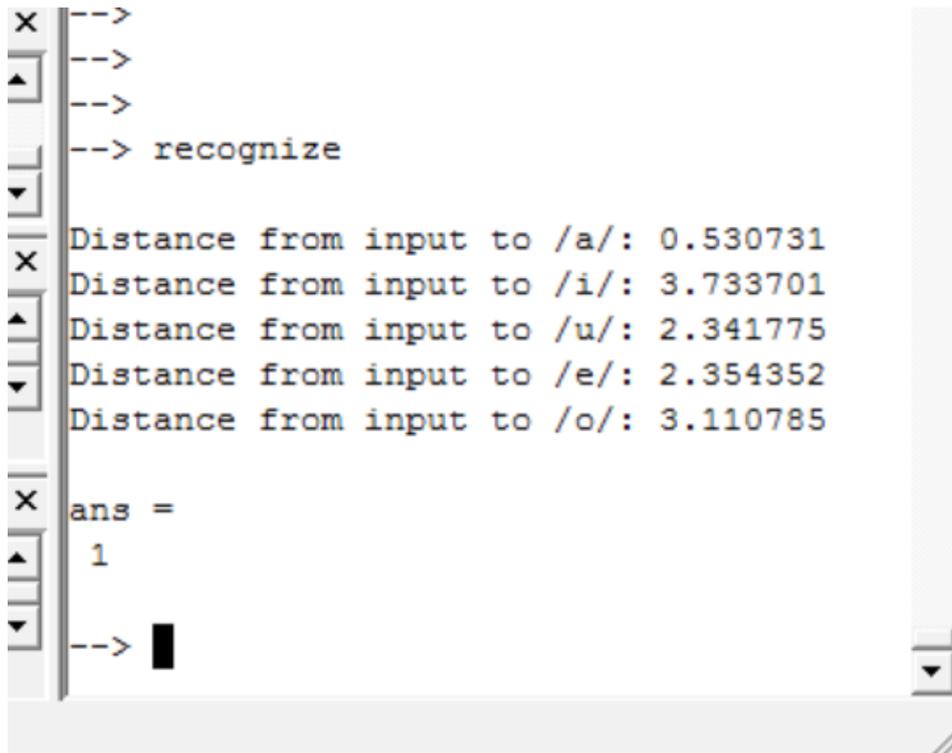
距離を計算

/a/ が最も距離が小さい

この人は /a/ と言ったに違いない

音声認識を試そう

- 1.FreeMatのコマンドを打ち込む部分に
`recognize` と入力してプログラムを実行させよう
- 2.すぐに録音状態になるので、2秒間「あー」と言おう
- 3.«ans = 1」と出れば成功



```
x -->
-->
-->
--> recognize
Distance from input to /a/: 0.530731
Distance from input to /i/: 3.733701
Distance from input to /u/: 2.341775
Distance from input to /e/: 2.354352
Distance from input to /o/: 3.110785
x ans =
  1
--> █
```

- 「いー」～「おー」も試そう
(「2」～「5」が返ればOK)
- 比較用の音声は私の声なので、
声質が違うとうまくいかないかも

まとめ

- \sin の式を入れれば、計算機で音を作れる
- ある周波数の正弦波に、その整数倍の周波数の正弦波を合成すると、1つに聞こえる(場合がある)
- 混ぜる正弦波の振幅をどのぐらいにするかが、音の音色を決める
- 合成後の波形から、合成前の正弦波を知るには、「フーリエ変換」を使う
- これらが、音声認識／音声合成の基本となる