知的情報処理論 第9回
Introduction

2023年6月13日（火）
武田

後半の内容

題材: 音声信号処理（離散時間）
視点: 統計的信号処理（not 物理音響）
立場: 理論の応用（確率的生成モデル）

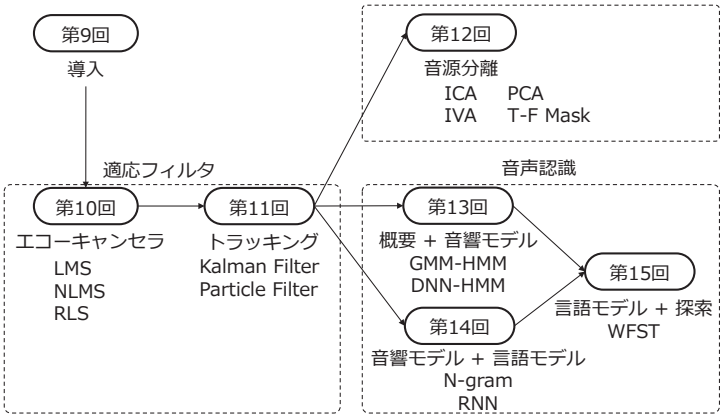
目的: 具体的な**問題設定**や**基本**技術に触れる
「音声を聞く・認識する」を広く浅く（～2000年頃）
”イメージを平たくつかむ”ことを重視

<u>基本的な概念は現在でも通ずる</u>

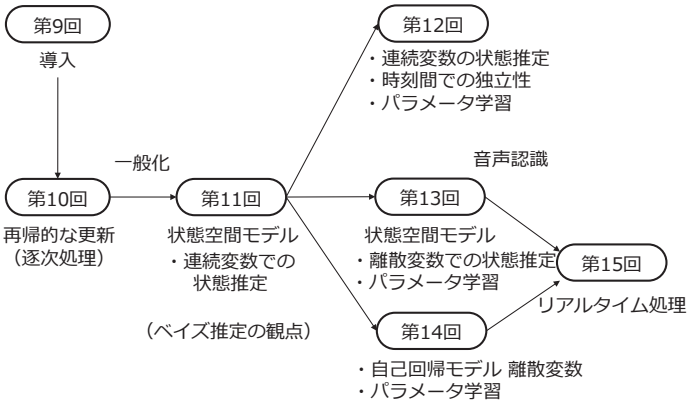
– 適応フィルタ
– 音源分離
– 音声認識

} 音声対話処理（駒谷先生後半分）の前段部分（信号～テキスト変換）が題材

各回の内容（予定）



各回の内容（予定）

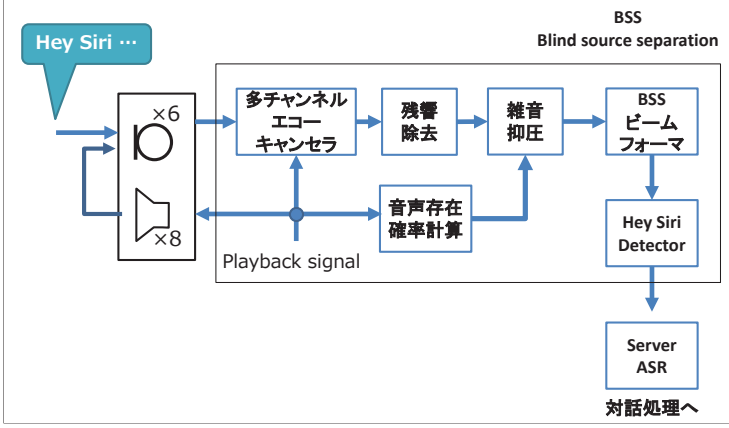


本日の内容: 導入

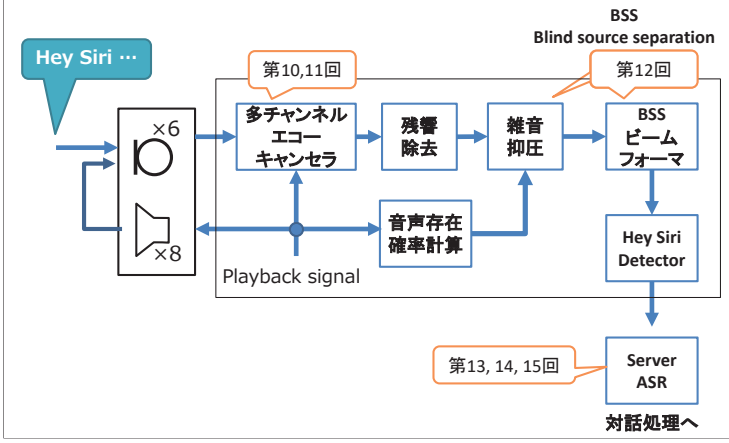
1. 具体的な問題設定
2. 音声信号の計算機表現
3. 音の伝達モデル
4. スペクトログラム
5. プログラミング・ツール

色んな問題設定とデモ

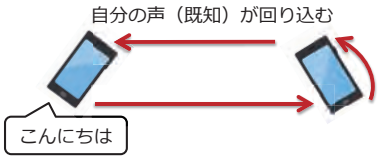
Apple Home Pod の構成 (SLT2018より)



Apple Home Pod の構成 (SLT2018より)



エコーキャンセラ (適応フィルタ)



音声が回り込む

⇩

明瞭度の低下
エコーの増幅（ハウリング）

デバイスの声も入力

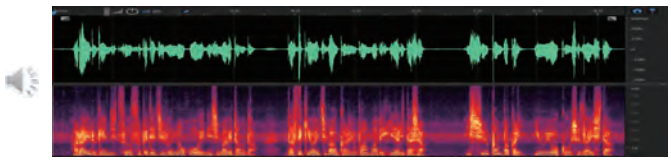
⇩

音声認識が困難

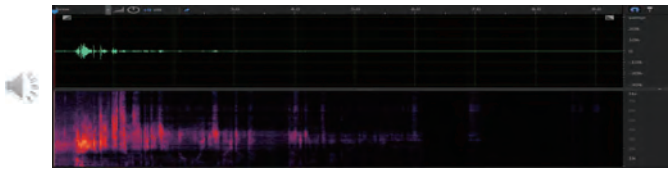
エコー（既知音）を抑圧したい

エコーキャンセラ

キャンセル前（観測信号）



逐次的なキャンセル



音源定位

音の到来方向を検知したい

音源定位

音源分離

混ざった音を分離/特定方向の音を強調

例: 正面方向以外の音を抑圧

音声認識

音声信号をテキストに変換

音声認識

音響モデル: (Infinite) GMM

言語モデル: 教師なし単語分割

音声信号の計算機表現

音声信号の録音

マイク
空気の疎密波を電気信号に変換

アナログ-デジタル変換

アナログ信号
（電気信号）

デジタル信号

変換 0100011101010…

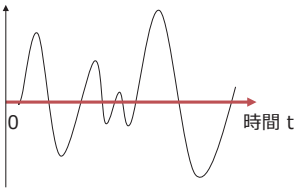
サンプリング・量子化

計算機/PC

オーディオ
インターフェース

サンプリング

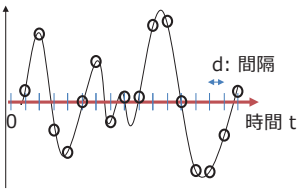
連続時間信号



$s(t)$

変換

離散時間信号



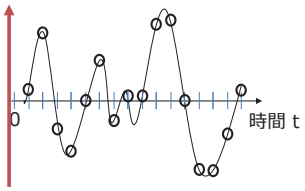
$[s(0), s(d), s(2d), ...]$

連続信号を一定時間毎に測定

関連キーワード: 帯域制限, サンプリング定理

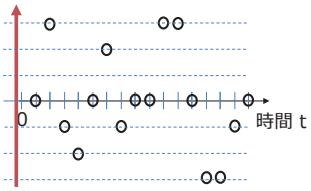
量子化

離散時間信号



変換

離散時間化+量子化



信号の大きさを離散的な値で近似

データサイズと精度のトレードオフ

具体的な音声データ形式の例

音声ファイル

サンプリング周波数: 16kHz

量子化: 符号付き 16 bit 整数
-32768～32767

音声ファイル

中身＝
整数のベクトル

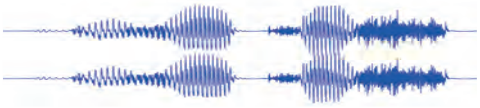
$\mathbf{s} = [0, -3, 58, -38, 44, -10, 573, -125, ...]$

5秒の音声データ → 5*16000=80000 次元のベクトル

多チャンネルマイクの場合

Nチャネルマイク

Nチャネル
音声信号



⋮

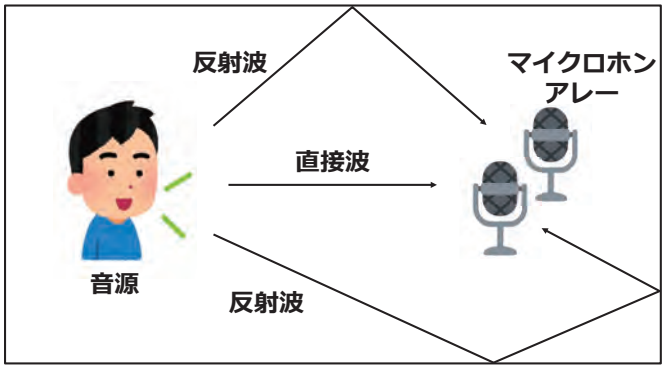
モノラル（1チャネル）の N倍のデータ

Mini quiz #1

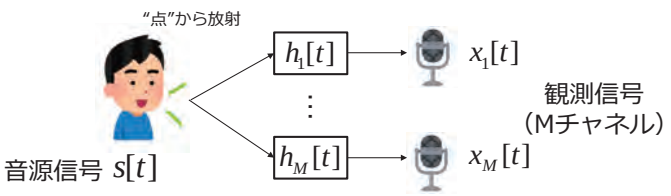
- サンプリング周波数 48kHz で録音した音声信号の場合，5秒間でいくつのデータ点が含まれるだろうか？

音の伝達モデル

閉空間における音の伝搬イメージ



インパルス応答によるモデル



$h_m[t]$ ：音源から m 番目までの伝搬経路の
インパルス応答（伝達関数）

インパルス応答を測定し，フィルタとして用いれば，静的な音響系のシミュレーションが可能

線形時不変システムとして見た場合

線形時不変システム（離散時間）

$s_1[t], s_2[t]$：入力
$x_1[t], x_2[t]$：対応するシステム応答

線形性 $L\{as_1[t]+bs_2[t]\} = ax_1[t]+bx_2[t]$
時不変性 $L\{s_1[t-d]\} = x_1[t-d]$
a, b, d：任意の定数

先の例だと音の伝搬経路

インパルス応答

単位インパルスとそのシステム応答

入力 $\delta[t]=\begin{cases}1 & t=0\\ 0 & otherwise\end{cases}$

応答 $L\{\delta[t]\}=h[t]$

例: 手を「パンッ」と鳴らしたときに聞こえる音の響き
手 → 耳の伝搬経路

畳み込みによる観測信号の生成

音源信号のインパルス列を用いた表現

$$s[t]=\sum_d s[d]\delta[t-d]$$

システム出力=畳み込み

$$x[t]=L\{s[t]\}=L\{\sum_d s[d]\delta[t-d]\}$$

$$=\sum_d s[d]L\{\delta[t-d]\}$$ 線形性

$$=\sum_d s[d]h[t-d]$$ インパルス応答

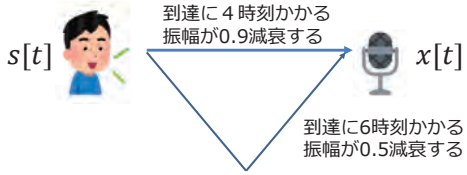
$\left(x[t]=\sum_k s[t-k]h[k]\right)$

具体的なイメージ

モデル $x[t]=\sum_k s[t-k]h[k]$

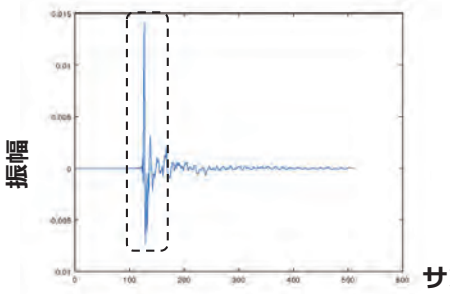
数値例 $x[t]=0.9s[t-4]+0.5s[t-6]$

– 現在時刻で観測した波形の値=
3時刻前と6時刻前の源信号（波形）の値の和



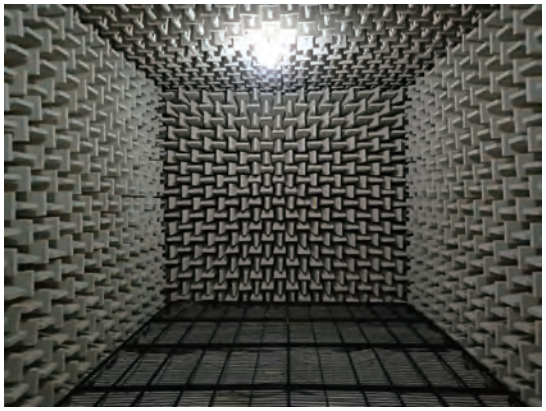
※ 実際はこんな都合の良い対応関係ではない

（無響室の）インパルス応答



ピーク: 直接波の到来　他: 反射・マイク特性
無響室: 音の反射をほとんどなくした部屋
（産研にあります．見学・研究利用も可．）

無響室



インパルス応答の測定例

測定用信号

Swept-sine 信号: 長い継続時間長，高SN比
Signal-to-noise ratio

Swept-sine 信号の生成

単位パルスのフーリエ変換の位相を2乗に比例して増加させ，逆フーリエ変換（DFTは後述）

測定後: 逆 swept-sine を畳み込み

$$S[k]=\begin{cases}\exp(\frac{-j\pi k^2}{N}) & 0\le k\le\frac{N}{2}\\ S^*[N-k] & \frac{N}{2}\le k\le N\end{cases}$$

N　長さ（2のべき乗）
S^* 複素共役

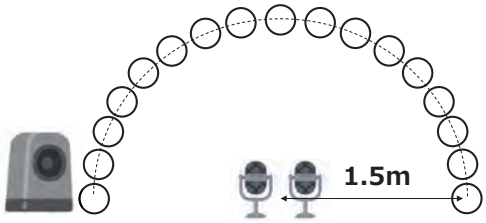
Swept-sine 信号

色々な位置でのデータ収集

スピーカ・マイクの位置

– 利用するパターンだけ測定する必要あり

例: マイクを中心に方向角10度づつ測定



一度測定しておけば，様々な方向からの音を再現可

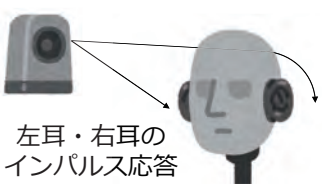
バイノーラル録音

臨場感を再現した録音

– 音源から鼓膜までの伝搬経路を再現

– 静的な音源ならインパルス応答で実現可

• ダミーヘッドなどを用いて収録
MIT で公開: https://sound.media.mit.edu/resources/KEMAR.html



注意点: 線形モデル

有限インパルス応答の場合

– Finite impulse response (FIR) フィルタ

$$x[t]=\sum_{d=0}^{N} h[d]s[t-d]=\begin{bmatrix}h[0] & h[1] & \cdots & h[N]\end{bmatrix}\begin{bmatrix}s[t]\\ s[t-1]\\ \vdots\\ s[t-N]\end{bmatrix}$$

$$=\mathbf{h}^T\mathbf{s}[t]$$

– ベクトルの内積で表現可能

$$\mathbf{h}=\begin{bmatrix}h[0] & h[1] & \cdots & h[N]\end{bmatrix}^T$$

$$\mathbf{s}[t]=\begin{bmatrix}s[t] & s[t-1] & \cdots & s[t-N]\end{bmatrix}^T$$

注意点: 線形モデル

前半（駒谷先生分）での線形識別関数と対比

$$g_i(\mathbf{x})=\mathbf{w}_i^T\mathbf{x}$$

$$x[t]=\mathbf{h}^T\mathbf{s}[t]$$

形は同じ線形: 2変数の内積表現

☹ 「形が同じだからパーセプトロンと同じか」

確かに似た側面はあるが，何でもかんでも同じだと思い込むのはよくない

– 変数の意味・役割は, 何をモデル化しているか, どんな問題/データに適用しているか，に依存

例: データが時間構造を持つなら，重みベクトル $\mathbf{h}$ の要素の**順番**にも意味があり，伝達関数としての側面も持つ
＝周波数特性を考えてみることもできる

Mini quiz #2

• バイノーラル録音音声を聞いた感想を述べよ

– これは今聞けないので，採点等に影響しません

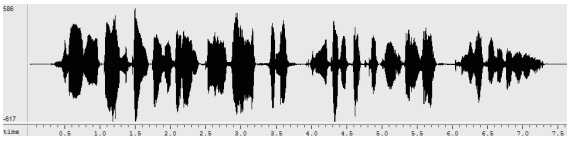
– が，講義のサイトにおいてあるので，後で聞いてください

• ダウンロード後，ヘッドセットやイヤホンで聞くことを推奨します
• CLE上で再生すると，モノラルに変換されて再生される可能性があります
• 人によっては方向感を感じないこともあります

スペクトログラム

音声信号の基本的な解析

音声信号の何を感じているのか？



時間波形を見てもよくわからない

→ 声に含まれる周波数成分と変化

計算機で周波数分析を扱う必要

（離散）フーリエ変換: DFT

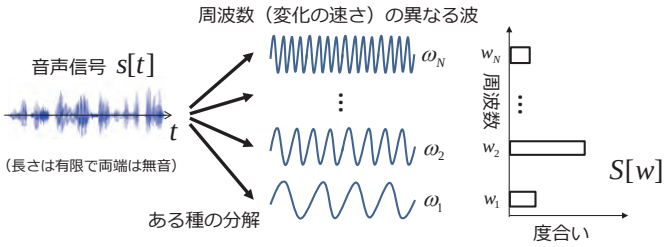
discrete Fourier transform

離散時間・周波数上での周波数解析

– ある周波数の強さの “度合い” を計算

– 高速計算が可能なアルゴリズム（FFT）も存在

fast Fourier transform



（離散）フーリエ変換: DFT

discrete Fourier transform

離散時間・周波数上での周波数解析

– ある周波数の強さの “度合い” を計算

– 高速計算が可能なアルゴリズム（FFT）も存在

fast Fourier transform

$$S[w]=\sum_{t=0}^{N-1} s[t]\exp(-j\frac{2\pi wt}{N}) \qquad (w=0,1,\ldots,N-1)$$

$$s[t]=\frac{1}{N}\sum_{w=0}^{N-1} S[w]\exp(-j\frac{2\pi wt}{N}) \qquad (t=0,1,\ldots,N-1)$$

平たく言うと２つのベクトル
（データとテンプレート）の内積（類似度）

$s[t] \Leftrightarrow S[w]$

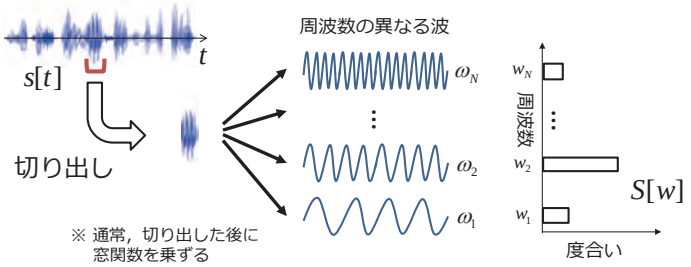
短時間フーリエ変換: STFT

short-time Fourier transform

長い信号・非定常信号の分析方法

– 波形の一部の区間を切り出す: 25ms～64ms

– 区間を周期的・定常だと仮定してフーリエ変換

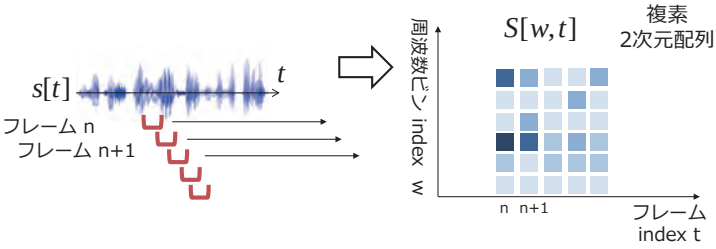


スペクトログラム

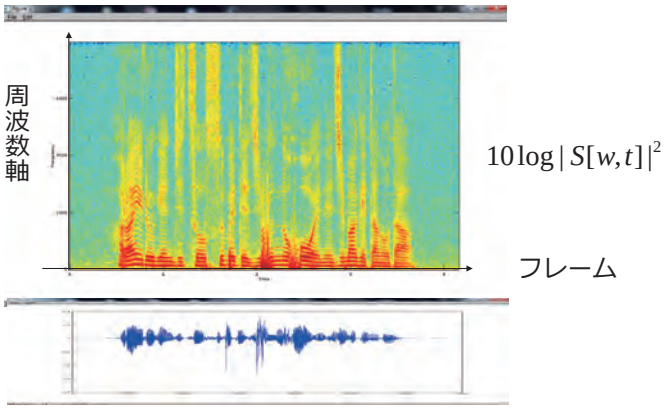
時間-周波数成分のプロット

– 解析区間をずらしながら，短時間フーリエ変換を適用: ずらし幅 10ms

– 信号の強さ（パワー）/度合いを色で表現



音声のスペクトログラム



性質

線形性

$s_1[t] \Leftrightarrow S_1[w]$, $s_2[t] \Leftrightarrow S_2[w]$ の時

$as_1[t]+bs_2[t] \Leftrightarrow aS_1[w]+bS_2[w]$ a,b 任意の定数

畳み込み

$(h*s)[t] \Leftrightarrow H[w]S[w]$ 畳み込み後の信号長には注意

- 周波数領域では積の関係 → 高速な畳み込み計算
- インパルス応答・伝搬経路の特性も積の関係

実際に録音した音声信号の "$H[w]$"に関しては，マイクの特性，ローパスフィルタ，など諸々の特性が積になっている点に注意

プログラミング・ツール

実装/演習やレポート課題に関するヒント

実装＋動作確認を推奨

– 実際に試した方が経験になる

☺ 音声信号は聞いて確認可能

☺ インパルス応答も公開されている

比較的使いやすいソフトを紹介

– ここではレポート課題で必要な演算を記載

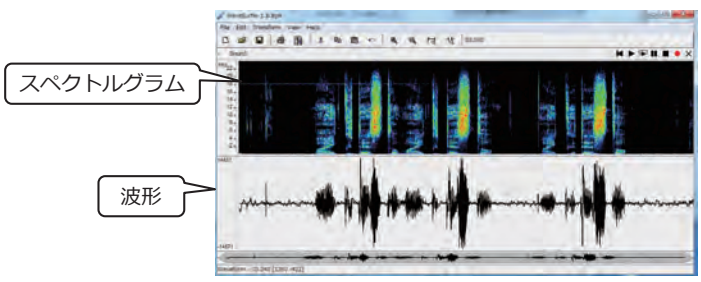
☹ 本来は簡単な課題を自分で難解にして（時間をかけて）解く人が見受けられました

→ かなり丁寧に情報を追記（が，なぜか見てない人も多い）

wavesurfer

音声の分析ソフト

– 音声の録音再生，スペクトル解析などが可能



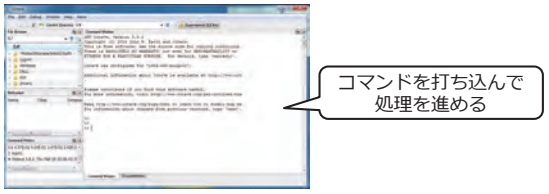
Octave

数値計算ソフト: win, linux 版など

– ベクトル，行列演算，線形代数等の基本ライブラリも充実．録音・再生も可能．

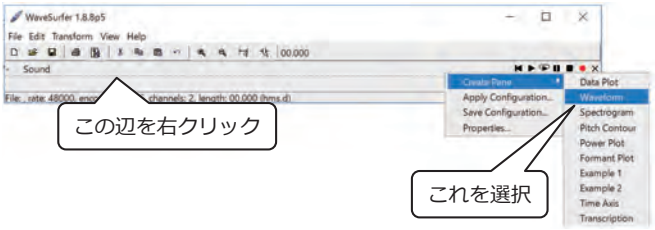
- 信号処理，統計解析の関数, GUI, グラフも用意
- Matlab のクローン → numpy 等と関数名共通

– 「式」をそのまま書きやすい



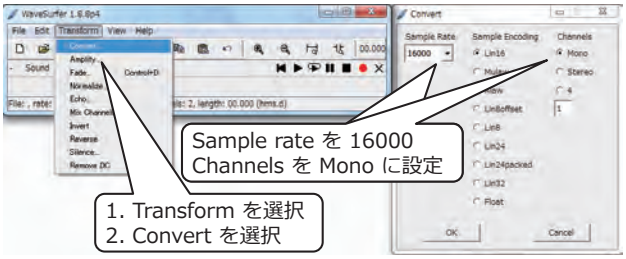
音声の録音 1/3

1. wavesurfer を起動
2. 何もないところを「右クリック」 → Create Pane → 「WaveForm」を選択



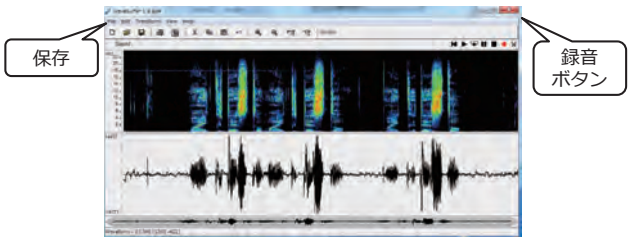
音声の録音 2/3

3. 設定変更: transform → convert を選択



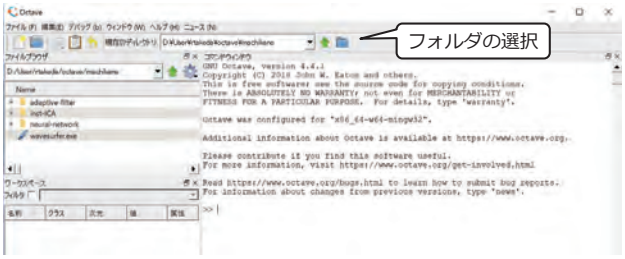
音声の録音 3/3

4. 「赤い○」で録音開始
5. 「黒い■」で録音停止
6. File → Save As … で名前を付けて保存
   record.wav と言う名前を付ける



音声信号のプロット 1/3

1. Octave の起動
2. 「フォルダ」を wav ファイルを保存したディレクトリに移動

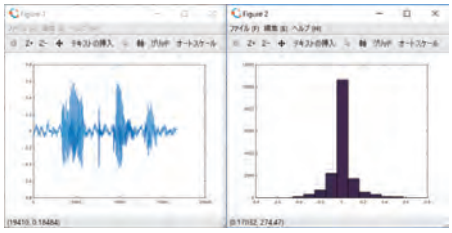


音声信号のプロット 2/3

3. Octave 上で下記のコマンドを打つ

```
>> [y fs] = audioread('record.wav');
>> plot(y);
>> figure; hist(y);
```

4. 波形とヒストグラムがプロットされる



音声信号のプロット 3/3

5. データの中身の一部を表示

```
>> y(1:32)
ans =

  -0.0104980
  -0.0114136
  -0.0098572
  -0.0098877
  -0.0094910
  -0.0097046
  -0.0144348
  -0.0192566
  -0.0209656
  -0.0163574
...
```

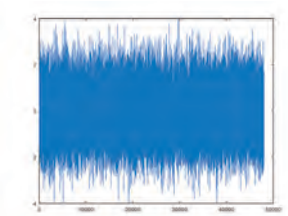
中身は点列（ベクトル）

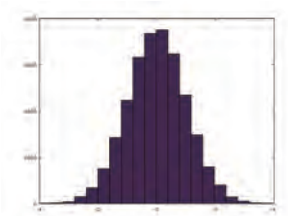
ガウス雑音のプロット

1. Octave 上で下記のコマンドを打つ

```
>> n = randn(16000*3, 1);
>> plot(n);
>> figure; hist(n);
```

2. 波形とヒストグラムがプロットされる





四則演算など

```
>>  5 + 3
ans =  8
>>  5 - 3
ans =  2
>>  5 * 3
ans =  15
>>  5 / 3
ans =  1.6667
>>  5\3
ans =  0.60000
>>  5 ^ 3
ans =  125
>>  mod(5, 3)
ans =  2
>>   1 – 3 * i
ans =  1 - 3i
```

左除算: 3 を 5 で割る

べき乗

剰余

複素数

ベクトル

```
>> x = [1 2 3]
x =

  1  2  3

>> x = [1, 2, 3]
x =

  1  2  3

>> x = [1; 2; 3]
x =

  1
  2
  3
```

行ベクトル

行ベクトル

列ベクトル

```
>> x = [1 2 3]'
x =

  1
  2
  3
>> x = [1+2*i 2-1*i 3]'
x =

  1 - 2i
  2 + 1i
  3 - 0i

>> length(x)
ans =  3
```

アポストロフィーは転置

複素数の場合は共役転置

長さを取得する関数

コロンによるベクトル表現

```
>> x = 1: 10
x =

  1  2  3  4  5  6  7  8  9  10

>> x = (1 : 10)'
x =

  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
```

```
>> x = 1: 2: 10
x =

  1  3  5  7  9

>> x = (1 : 5) * 2
x =

  2  4  6  8  10
```


行列

```
>> A = [1 2 3; 4 5 6]
A =

  1  2  3
  4  5  6

>> A = [1 2 3; 4 5 6]'
A =

  1  4
  2  5
  3  6

>> size(A)
ans =

  3  2
```

サイズを取得する関数

```
>> fliplr(A)
ans =

  4  1
  5  2
  6  3

>> flipud(A)
ans =

  3  6
  2  5
  1  4

>> transpose(A)
ans =

  1  2  3
  4  5  6
```

左右を反転させる関数

上下を反転させる関数

転置させる関数（共役は取らない）

行列

```
>> A = [1 2; 3 4];
>> B = [5 6; 7 8];
>> A + B
ans =

   6   8
  10  12

>> A * B
ans =

  19  22
  43  50
```

```
>> A .* B
ans =

   5  12
  21  32
```

* / \ ^ で通常の行列演算を表す
.* ./ .\ .^ で要素同士の演算を表す

変数がベクトルの場合も同様

関数の例: インパルス応答畳み込み

- conv 関数を利用

```
>> pkg load signal;
```
信号処理パッケージのロード

```
>> [hL, fs] = audioread('elev0/L0e045a.wav');
>> [hR, fs] = audioread('elev0/R0e045a.wav');
```
MIT HRTF のロード

https://sound.media.mit.edu/resources/KEMAR.html

```
>> hL = resample(hL, 16000, fs);
>> hR = resample(hR, 16000, fs);
```
ダウンサンプリング

```
>> [s, fs] = audioread('record.wav');
```
録音した音声ファイル

```
>> xL = conv(s, hL);
>> xR = conv(s, hR);
```
L, Rチャネルの畳み込み

```
>> x = [xL, xR];
```
2チャンネルデータに形成

```
>> audiowrite('aiueo_elev0_045_LR.wav', x, fs);
```
ファイルに書き込み

ベクトル/行列の要素へのアクセス

– 行と列を示す数値を代入
– コロン ： はある行/列のすべての要素を指す

```
>> A = [ 1 2 3; 4  5 6];
>> A
A =

  1  2  3
  4  5  6

>> A(1,1)
ans =  1
>> A(1,2)
ans =  2
```

```
>> A(1,:)
ans =

  1  2  3

>> A(:, 2)
ans =

  2
  5
```


ベクトル/行列の要素への部分アクセス

– コロン ： の前後に数値を設定
– 要素の最後のインデックスは end で参照可能

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
A =

   1   2   3   4
   5   6   7   8
   9  10  11  12

>> A(2, 2:end-1)
ans =

  6  7
```

```
>> A(2:end, end-1:end)
ans =

   7   8
  11  12
```


部分配列を用いたベクトルの内積

```
>> h = [1 2 3 4 5 6]'
h =

  1
  2
  3
  4
  5
  6
>> s = [10 9 8 7 6 5 4 3 2 1]'
s =
  10
   9
   8
   7
   6
   5
   4
   3
   2
   1
```

```
>> s(3:8)
ans =

   8
   7
   6
   5
   4
   3

>> h' * s(3:8)
ans =  98

>> t=8
t =  8

>> h' * s(t-5:t)
ans =  98
```

1*8+2*7+3*6+4*5+5*4+6*3=98

関数の例: 固有値分解

- eigs 関数を利用

```
>> A = [1 2; 2 9]
A =

  1  2
  2  9

>> [E L] = eigs(A)
E =

  0.22975  -0.97325
  0.97325   0.22975
```

```
L =

Diagonal Matrix

  9.47214        0
        0  0.52786

>> E * L * E'
ans =

  1.00000  2.00000
  2.00000  9.00000
```


関数の例: 逆行列

- inv/pinv 関数を利用

```
>> A = [1 2; 2 9]
A =

  1  2
  2  9

>> B = inv(A)
B =

  1.80000  -0.40000
 -0.40000   0.20000
```

```
>> A*B
ans =

  1.00000   0.00000
 -0.00000   1.00000
```

制御文

```
If > a 0
  command1;
elseif a == 0
  command2;
else
  command3;
end
```

```
for i = 1: 10
   x = x + i * i;
   fprintf(1, '%d¥n', x);
end
```

```
while result > 0
   result = command(a, b, c);
end
```

論理和には |, 論理積には &, 否定には ~ を用いる.
ベクトル内の全要素に対する論理和・論理積は any, all を用いる

行列関連の関数

- 詳しい説明や引数はモジュールのヘルプ等を参照すること

zeros(m,n) :要素がすべて 0 の m * n 行列を生成
ones(m,n) :要素がすべて 1 の m* n 行列を生成
reshape(A, m, n) ：行列Aを m * n 行列に変形
- ベクトルを行列等に変換するときに使う

find(A) ：行列Aの非ゼロのインデックスを出力
- 通常は find(X > 100) のように用いる

sum(A) ：行列Aの列毎の合計値を出力
mean(A) ：行列Aの列毎の平均値を出力
std(A) ：行列Aに対して，列毎の標準偏差を出力

関数のヘルプ

```
>> help zeros
'zeros' is a built-in function from the file libinterp/corefcn/data.cc

 -- zeros (N)
 -- zeros (M, N)
 -- zeros (M, N, K, ...)
 -- zeros ([M N ...])
 -- zeros (..., CLASS)
    Return a matrix or N-dimensional array whose elements are all 0.

    If invoked with a single scalar integer argument, return a square
    NxN matrix.

    If invoked with two or more scalar integer arguments, or a vector
    of integer values, return an array with the given dimensions.

    The optional argument CLASS specifies the class of the return array
    and defaults to double.  For example:

        val = zeros (m,n, "uint8")

    See also: ones.
```


参考文献

- 荒木雅弘「フリーソフトでつくる音声認識システム」
- 浅野太「音のアレイ信号処理」
- 飯田一博「頭部伝達関数の基礎と3次元音響システムへの応用」
- 大類重範「ディジタル信号処理」