第3回：音声信号とデジタル化

音声処理と認識

担当：小野原彩香
形式：動画回（非同期・自分のペースで）

この回は動画回です。Google Formsでテストを受けてください。

本日の問いと学習のキーワード

> マイクが拾った音は、どのようにしてコンピュータが扱える数値の列になるのか？

今回のキーワード

- アナログ / デジタル
- サンプリング（標本化）・サンプリングレート F_s
- ナイキスト定理
- エイリアシング
- 量子化・ビット深度
- PCM（パルス符号変調）
- WAVファイル

音は「波」、コンピュータは「数値の列」

アナログ信号

時間と振幅が連続する信号。マイクは空気の圧力変化を電気信号へ変換する。

デジタル信号

一定間隔で取り出した値を、有限の精度で表した数値列。

> 帯域制限された信号の理想的な標本化では、元の波を復元できる。
> 振幅を有限段階に丸める量子化では、誤差が生じる。

必要な帯域・録音レベル・機器の雑音を考えて、標本化周波数とビット深度を選ぶ。

デジタル化 = サンプリング + 量子化

① サンプリング：時間を離散化

1秒間に取り出す標本の数がサンプリングレート F_s（Hz）。

$$x[n] = x(n/F_s)$$

② 量子化：振幅を有限段階で表す

ビット深度 b は1標本の値を表すビット数。線形整数PCMでは2^b段階。

$$b = 16 \quad \Rightarrow \quad 2^{16} = 65{,}536$$

ここでは、等間隔の量子化を用いる線形PCMを中心に扱う。

サンプリング — 時間を「飛び石」にする

上段：連続したサイン波（アナログ）
中段：等間隔の垂直な破線（サンプリングタイミング）が引かれた同じ波
下段：各タイミングでの波の値のみが点（●）として残っている

$$F_s\ (\mathrm{Hz}) = \frac{\text{サンプル数}}{1\ \text{秒}}$$

具体例

- $F_s = 44100\ \mathrm{Hz}$ のとき：1サンプルあたりの時間間隔

$$\frac{1}{44100} \approx 22.7\ \mu\mathrm{s}$$

サンプリングレートが低すぎると何が起きるか？ → 次のスライドのナイキスト定理で答える

標本化定理とナイキスト周波数

$$F_s > 2f_{\max} \qquad (f_{\mathrm{Nyquist}} = F_s/2)$$

最高周波数 f_{max} に帯域制限された信号は、この条件と理想的な補間のもとで復元できる。

> **ちょうど2倍の境界には注意：**位相によって標本がすべて0になる。
>
> $$f = F_s/2 \quad \Rightarrow \quad x[n] = \sin(\pi n) = 0$$

- F_s = 8 kHz ならナイキスト周波数は4 kHz。
- 実機では4 kHz未満に通過帯域を置き、フィルタの遷移帯域を確保する。
- 音声には倍音や摩擦音もあるため、F0だけでは必要帯域を決められない。

エイリアシング（折り返し）

異なる周波数の波が同じ標本列になり、区別できなくなる現象。

$$F_s = 8{,}000\ \mathrm{Hz}: \quad 5{,}000\ \mathrm{Hz} \longrightarrow 3{,}000\ \mathrm{Hz}$$

実数正弦波では位相・符号の違いを含めて低い周波数に重なる。
9,000 Hzも1,000 Hzの成分と区別できなくなる。

対策：標本化・間引きの前に帯域を制限

- ADCの前：アナログのアンチエイリアシングフィルタ。
- ダウンサンプリングの前：新しいナイキスト周波数に合わせたデジタルフィルタ。

> 車輪の逆回転も時間的なエイリアシングの例。スポークなどの模様の反復と撮影のタイミングが関係する。

量子化 — 振幅を「段階」に変換する

表現	段階数・性質	例
8 bit整数	256段階	一部のレトロ音源
16 bit整数	65,536段階	CD・整数PCM音声
24 bit整数	16,777,216段階	録音・制作
32 bit float	指数部＋仮数部・間隔は不均一	librosaの波形配列（既定）

量子化誤差：同じ振幅範囲なら、整数のビット深度を増やすほど量子化幅は小さくなる。誤差は雑音だけでなく歪みとして現れる場合もある。

> 理想的な一様量子化・フルスケール正弦波などを仮定したSNR：
> 約6.02b + 1.76 dB（16 bitで約98 dB）。実機の性能を保証する式ではない。

PCMとデータ量

PCM（Pulse Code Modulation）：標本の量子化値を符号で表す。
ここでは非圧縮の線形PCMを扱う。

量子化後の標本値をそのまま保存する。アナログ信号の情報が無条件にすべて残るという意味ではない。

$$\text{データ速度}\ [\mathrm{byte/s}] = F_s \times (b/8) \times \text{チャンネル数}$$

例：CDの音声データ

$$44{,}100 \times (16/8) \times 2 = 176{,}400\ \mathrm{byte/s}$$

1分で10,584,000 byte ≈ 10.58 MB（約10.09 MiB）。
ファイルヘッダ等の容量は別に必要。

WAVは音声を格納するコンテナ

RIFF/WAVE形式には複数のチャンクがある。PCM以外の格納方式もある。

```
RIFFヘッダ：12 byte
  "RIFF" ／ RIFFサイズ（通常ファイル全体 − 8）／ "WAVE"

fmtチャンク：8 byteのヘッダ ＋ フォーマット情報
  格納方式、チャンネル数、サンプリングレート、ビット深度など
  基本的なPCMの情報部分は16 byte（合計24 byte）

dataチャンク：8 byteのヘッダ ＋ 音声データ
  データの解釈はfmtチャンクの格納方式に従う
```

追加チャンクやパディングが入る場合もある。

「常に44 byteのヘッダを飛ばせばよい」とは限らない。

サンプリングレートごとの用途例

レート	ナイキスト周波数	用途例
8,000 Hz	4,000 Hz	G.711など狭帯域電話
16,000 Hz	8,000 Hz	16 kHz入力を前提とする音声認識モデル
22,050 Hz	11,025 Hz	librosa.loadの既定のリサンプリング先
44,100 Hz	22,050 Hz	CD
48,000 Hz	24,000 Hz	映像・放送の音声

44.1 kHzは、20 kHz付近までの帯域とフィルタの遷移帯域を確保できる。標本化定理だけから44.1 kHzという値が一意に決まるわけではない。

> 音声認識の入力レートはモデルの仕様に合わせる。
> 16 kHzはすべての機器・サービスに共通する標準ではない。

PythonでWAVを読む（1）— wave

```
import wave
import numpy as np

# この例は16 bit整数PCMのRIFF/WAVE用（Python 3.12で確認）
with wave.open("sample.wav", "rb") as wf:
    n_channels = wf.getnchannels()
    sample_width = wf.getsampwidth()
    sr = wf.getframerate()
    n_frames = wf.getnframes()  # 1チャンネルあたりの標本数
    if sample_width != 2:
        raise ValueError("16 bit整数PCMのWAVを指定してください")
    raw = wf.readframes(n_frames)

audio = np.frombuffer(raw, dtype="<i2").reshape(-1, n_channels)
print(f"{sr} Hz, {n_channels} ch, {n_frames / sr:.3f} 秒")
print(audio.shape)
```

2秒・16 kHz・モノラルなら形状は(32000, 1)。

24 bitやfloat等には、格納方式に対応したsoundfileなどを使う。

PythonでWAVを読む（2）— librosa

```
import librosa
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio, display

# sr=None：元のレートを保持。mono=True：モノラルへ変換
y, sr = librosa.load("sample.wav", sr=None, mono=True)
print(f"{sr} Hz, {len(y) / sr:.3f} 秒, {y.dtype}")

fig, ax = plt.subplots(figsize=(12, 3))
ax.plot(np.arange(len(y)) / sr, y, linewidth=0.5)
ax.set(xlabel="時間 (秒)", ylabel="振幅")
plt.tight_layout()
plt.show()
display(Audio(y, rate=sr))
```

整数PCMは通常±1付近の浮動小数点数に変換される。
ピークを1にそろえる処理ではなく、float形式等では±1を超える値も扱える。

サンプリングレートを変えると音はどう変わるか

```
import librosa
import numpy as np
from IPython.display import Audio, display

y_original, sr_original = librosa.load("sample.wav", sr=None)
print(f"元のサンプリングレート: {sr_original} Hz")

# 異なるサンプリングレートにリサンプリングして耳で比較
for target_sr in [8000, 16000, 22050, 44100]:
    y_resampled = librosa.resample(y_original,
                                   orig_sr=sr_original,
                                   target_sr=target_sr)
    print(f"\n--- {target_sr} Hz ---")
    display(Audio(y_resampled, rate=target_sr))
```

体験の目的：帯域を変えたときの聞こえ方を比較する。了解度は素材や雑音に依存する。アップサンプリングで失われた高域は戻らない。Audioの既定の再生はピーク正規化されるため音量比較には使わない

よくある誤解と注意点

サンプリングレートは目的とモデル仕様で選ぶ

高くすると保存量・計算量が増える。上げ直しても、既に失われた高域は復元できない。

標本化の前に帯域制限が必要

理論上の条件と、実機のフィルタの通過帯域・遷移帯域を区別する。

librosa.loadの対応形式はバックエンドに依存

WAV以外にも対応する。常にFFmpegを経由するわけではない。

整数PCMのスケール変換とピーク正規化は別

16 bit整数を32768で割ると−1以上1未満。音声ごとの最大振幅を1にそろえる操作ではない。

次回（第4回）の予告

形式：オンライン同期回（Zoom）
テーマ：音声の「周波数の時間変化」を可視化する
内容：フーリエ変換の直感的理解・短時間フーリエ変換（STFT）・スペクトログラムの読み方

事前学習（取り組んでおくと第4回の理解度が大幅に上がります）

1. このスライドのコード例を Google Colab に貼り付けて、自分の声や手持ちの音楽ファイルで実行してみる
2. 「フーリエ変換 わかりやすく」などで検索し、動画や図解を1つ見る（数式の理解は不要）

考えておく問い（任意・次回冒頭で取り上げます）

- 「ア」と「イ」の音は、スペクトログラムでどんな違いがあると思うか？

第3回のまとめ

概念	要点
サンプリング	一定の時間間隔で標本を取り出す
標本化定理	帯域制限した信号について $F_s > 2f_{max}$
量子化	有限の精度で振幅を表す。整数b bitでは2^b段階
エイリアシング	異なる周波数が同じ標本列になり、区別できなくなる
線形PCM	量子化した標本値の符号表現
WAV	fmt・data等のチャンクを持つコンテナ
読み込み	格納方式・ビット深度・レート・チャンネルを確認

テスト：Google Forms（5問・4択）。締め切りは授業の案内を確認。
詳しい導出・出典・実行コードは講義本文に掲載。