

計算機科学実験及演習 4 音響信号処理 レポート

工学部 情報学科 計算機科学コース

学生番号: 1029358455 氏名: 登古紘平

2026年1月8日

課題内容

音響信号ファイルを読み込み、音響信号のさまざまな情報を表示するグラフィカルユーザーインターフェースを作成せよ。少なくとも以下の 3 つを同時に表示させること。

1. 音響信号のスペクトログラム
2. 音響信号の基本周波数
3. 母音などの何らかの識別を行った結果

加えて、このインターフェースがより便利なものになるように改良せよ。

1 作成したプログラムの説明

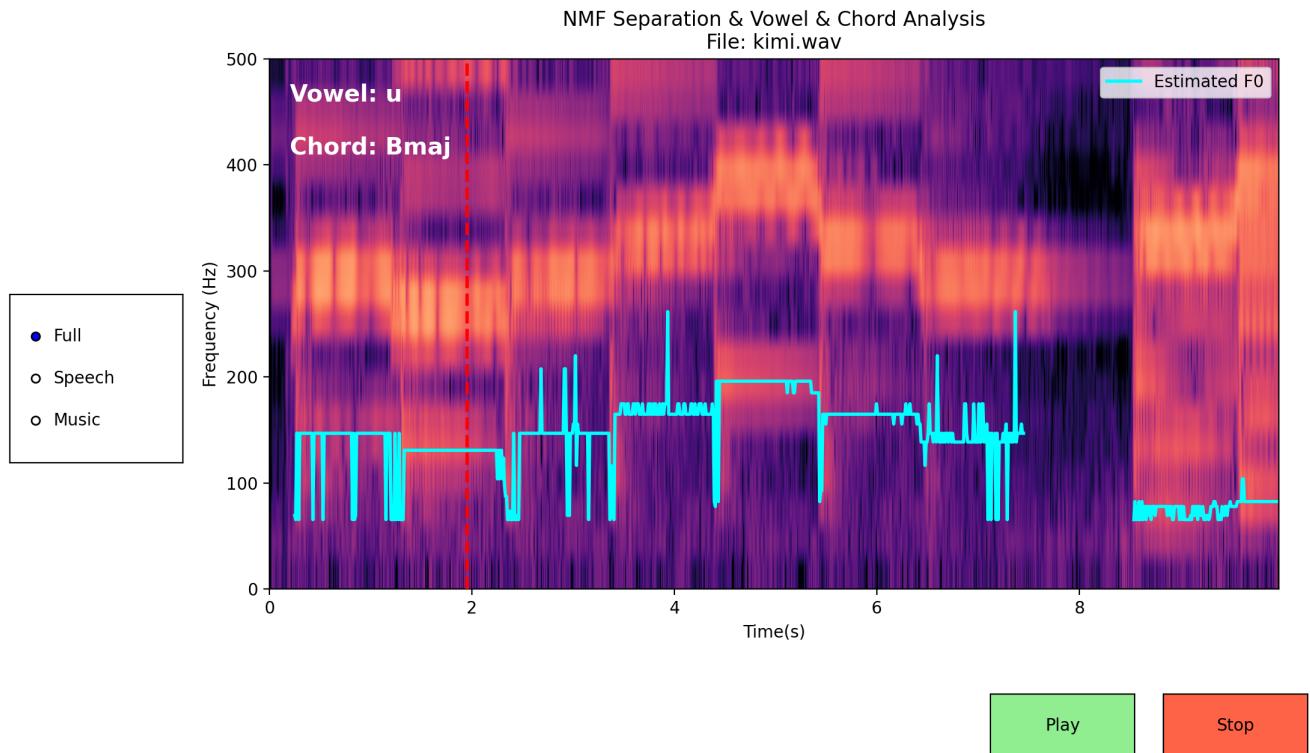


図 1 実際の GUI の画面

図 1 に、今回作成した GUI の画面を示す。このプログラムを実行すると、wav ファイルを選択できる。選択すると、GUI が、選択したファイル名とともに表示される。GUI 上には、音響信号のスペクトログラムおよび推定した基本周波数をプロットしている。Play ボタンを押すと、音声が再生されるとともに、赤い点線のアニメーションにより再生部分を可視化している。また、画面左上に、リアルタイムで母音、和音を推定したもの表示させている。加えて、GUI 左側に、NMF を用いて楽曲から音声と音楽を分離させる機能も実装した。Full はオリジナルの楽曲、Speech は音声、Music は音楽を再生する。以下、ソースコードとその説明

である。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Button, RadioButtons
import librosa
import sounddevice as sd
import math
import tkinter as tk
from tkinter import filedialog
import sys

# ファイル選択ダイアログを表示
def select_file():
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.askopenfilename(
        title="使用する wav ファイルを選択してください",
        filetypes=[("WAV files", "*.wav")]
    )
    root.destroy()
    return file_path

# ファイル選択を実行
selected_path = select_file()
if not selected_path:
    print("ファイルが選択されませんでした。終了します。")
    sys.exit()

# 音響解析のパラメータ設定
SR = 16000 # サンプリングレート
size_frame = 512 # フレームサイズ
size_shift = int(SR / 100) # シフト幅
db_lim = -30 # 音量の閾値
spectral_envelope_number = 13 # ケプストラム係数の数
max_harmonix = 5 # 倍音の最大次数
hamming_window = np.hamming(size_frame) # ハミング窓

# ノートナンバーから周波数への変換
```

```

def nn2hz(notenum):
    return 440.0 * (2.0 ** ((notenum - 69) / 12.0))

# 周波数からノートナンバーへの変換
def hz2nn(frequency):
    return int(round(12.0 * (math.log(frequency / 440.0) / math.log(2.0)))) + 69

# ケプストラムを計算
def calculate_cepstrum_feature(x_frame, num_coeffs=spectral_envelope_number):
    fft_spec = np.fft.fft(x_frame)
    fft_abs_spec = np.abs(fft_spec) + 1e-6
    log_spec = np.log(fft_abs_spec)
    ceps = np.real(np.fft.fft(log_spec))
    return ceps[:num_coeffs]

# モデル学習用の関数
def calculate_params(all_features):
    features_np = np.array(all_features)
    return np.mean(features_np, axis=0), np.var(features_np, axis=0)

# 母音識別のための確率計算
def calculate_log_likelihood(feature, average, variance):
    D = len(feature)
    variance_safe = variance + 1e-9
    term1 = - (D / 2.0) * np.log(2 * np.pi)
    term2 = - 0.5 * np.sum(np.log(variance_safe))
    diff = feature - average
    term3 = - 0.5 * np.sum((diff ** 2) / variance_safe)
    return term1 + term2 + term3

# 倍音加算法により基本周波数を推定
def estimate_pitch_harmonic_sum(log_spec, sr, size_frame, min_nn=36, max_nn=60):
    best_nn = -1
    max_sum = -float('inf')
    delta_f = sr / size_frame
    nyquist = sr / 2
    num_bins = len(log_spec)

    # 各ノートナンバーについて倍音の強度を合計
    for nn in range(min_nn, max_nn + 1):

```

```

f0 = nn2hz(nn)
h_sum = 0.0
for h_idx in range(1, max_harmonix + 1):
    f_harmonic = h_idx * f0
    if f_harmonic > nyquist: break
    bin_index = int(round(f_harmonic / delta_f))
    if bin_index >= num_bins: break
    h_sum += log_spec[bin_index]

if h_sum > max_sum:
    max_sum, best_nn = h_sum, nn
pitch_lim = -20.0
return best_nn if max_sum > pitch_lim else -1

# クロマベクトルを計算
def chroma_vector(spectrum, frequencies):
    cv = np.zeros(12)
    for s, f in zip(spectrum, frequencies):
        if f <= 0:
            continue
        nn = hz2nn(f)
        cv[nn % 12] += abs(s)
    return cv

# 和音テンプレートとの尤度を計算
def calculate_likelihood(cv, indices, weights):
    likelihood = 0.0
    for weight, index in zip(weights, indices):
        likelihood += weight * cv[index]
    return likelihood

# 母音モデルを学習 ※この引数には、今回私が録音した音声ファイルが渡される
def train_vowel_models(filename):
    try:
        x_s, _ = librosa.load(filename, sr=SR)
    except:
        print(f"学習用ファイル {filename} が見つかりません。")
        return None

# 各母音の時間区間を定義

```

```

vowel_segments = {
    'a': (0.5, 1.7), 'i': (1.7, 2.5), 'u': (2.5, 4.0), 'e': (4.0, 5.0),
    'o': (5.0, 6.0)
}
models = {}
for v, (start, end) in vowel_segments.items():
    segment = x_s[int(SR * start):int(SR * end)]
    features = []
    for i in np.arange(0, len(segment) - size_frame, size_shift):
        x_f = segment[int(i):int(i) + size_frame]
        features.append(calculate_cepstrum_feature(x_f))
    models[v] = calculate_params(features)
return models

# NMF による音声と音楽の分離
def separate_speech_music(y, sr):
    D = librosa.stft(y, n_fft=size_frame, hop_length=size_shift)
    Y = np.abs(D)
    K, eps, update_times = 2, 1e-10, 100 # 2成分に分解
    F, T = Y.shape
    np.random.seed(0) # 初期化
    H = np.random.rand(F, K) # 特徴行列
    U = np.random.rand(K, T) # 時間行列

    # NMF の反復更新
    for i in range(update_times):
        Y_hat = np.dot(H, U)
        H = H * (np.dot(Y, U.T) / (np.dot(Y_hat, U.T) + eps))
        Y_hat = np.dot(H, U)
        U = U * (np.dot(H.T, Y) / (np.dot(H.T, Y_hat) + eps))

    # 音声と音楽の成分を分離
    Y_speech_mag = np.dot(H[:, 0:1], U[0:1, :])
    Y_music_mag = np.dot(H[:, 1:2], U[1:2, :])
    Y_total_mag = Y_speech_mag + Y_music_mag + eps
    mask_speech = Y_speech_mag / Y_total_mag
    y_speech = librosa.istft(D * mask_speech, hop_length=size_shift)
    y_music = librosa.istft(D * (1 - mask_speech), hop_length=size_shift)
    return librosa.util.fix_length(y_speech, size=len(y)),
           librosa.util.fix_length(y_music, size=len(y))

```

```

# 和音テンプレート作成（メジャーとマイナーコード）
a_root, a_3rd, a_5th = 1.0, 0.5, 0.8 # 根音、3度、5度の重み
weights = [a_root, a_3rd, a_5th]
major_intervals = [0, 4, 7] # メジャーコードの音程
minor_intervals = [0, 3, 7] # マイナーコードの音程
chroma_labels = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
all_chord_templates = {}
chord_names_list = []

# 24種類の和音テンプレートを生成
for i in range(12):
    name = chroma_labels[i] + 'maj'
    all_chord_templates[name] = [(i + interval) % 12 for interval in major_intervals]
    chord_names_list.append(name)
for i in range(12):
    name = chroma_labels[i] + 'min'
    all_chord_templates[name] = [(i + interval) % 12 for interval in minor_intervals]
    chord_names_list.append(name)

# 母音モデルの学習
vowel_models = train_vowel_models('short.wav')

# 音響信号の読み込みと前処理
print(f"読み込み中: {selected_path}")
x_l, _ = librosa.load(selected_path, sr=SR)
x_speech, x_music = separate_speech_music(x_l, SR)

spectrogram = [] # スペクトログラム
estimated_pitch_nn = [] # 推定ピッチ（ノートナンバー）
likelihoods_results = [] # 母音識別結果
estimated_chord_indices = [] # 和音推定結果
vowel_list = ['a', 'i', 'u', 'e', 'o']
frequencies = np.fft.rfftfreq(size_frame, d=1.0 / SR)

for i in np.arange(0, len(x_l) - size_frame, size_shift):
    idx = int(i)
    x_frame = x_l[idx:idx + size_frame]

    # dB値計算

```

```

current_rms = np.sqrt(np.mean(x_frame**2))
current_db = 20 * np.log10(current_rms + 1e-6)

fft_spec = np.fft.rfft(x_frame * hamming_window)
mag_spec = np.abs(fft_spec)
fft_log_abs_spec = np.log(mag_spec + 1e-6)
spectrogram.append(fft_log_abs_spec)

# ピッチ推定
best_nn = estimate_pitch_harmonic_sum(fft_log_abs_spec, SR, size_frame)
estimated_pitch_nn.append(best_nn if current_db >= db_lim else -1)

# 母音識別
if vowel_models:
    ceps_feat = calculate_cepstrum_feature(x_frame)
    v_likelihoods = [calculate_log_likelihood(ceps_feat, vowel_models[v][0],
                                              vowel_models[v][1]) for v in vowel_list]
    likelihoods_results.append(np.argmax(v_likelihoods))
else:
    likelihoods_results.append(0)

# 和音推定
cv = chroma_vector(mag_spec, frequencies)
max_l, best_c = -float('inf'), -1
for name, indices in all_chord_templates.items():
    l_h = calculate_likelihood(cv, indices, weights)
    if l_h > max_l:
        max_l, best_c = l_h, chord_names_list.index(name)
estimated_chord_indices.append(best_c if max_l > 5.0 else -1)

# 時間軸とピッチデータの準備
total_duration = (len(spectrogram) * size_shift) / SR
times = np.linspace(0, total_duration, len(spectrogram))
idx_500 = np.where(frequencies <= 500)[0]
spec_data = np.array(spectrogram).T[idx_500, :]
pitch_hz = [nn2hz(nn) if nn > 0 else np.nan for nn in estimated_pitch_nn]

# GUI の構築
fig, ax = plt.subplots(figsize=(12, 7))
plt.subplots_adjust(bottom=0.25, left=0.2)

```

```

# スペクトログラムの表示
img = ax.imshow(spec_data, aspect='auto', origin='lower', extent=[times[0], times[-1],
frequencies[idx_500[0]], frequencies[idx_500[-1]]], cmap='magma', vmin=-5, vmax=5, zorder=1)

# ピッチ曲線の表示
pitch_line, = ax.plot(times, pitch_hz, color='cyan', linewidth=2, label='Estimated F0', zorder=2)

# 現在位置を示す縦線
v_line = ax.axvline(x=0, color='red', linestyle='--', linewidth=2, zorder=3)

# 母音と和音を表示するテキスト
vowel_text = ax.text(0.02, 0.92, '', transform=ax.transAxes, fontsize=14,
                      fontweight='bold', zorder=4, color='white')
chord_text = ax.text(0.02, 0.82, '', transform=ax.transAxes, fontsize=14,
                      fontweight='bold', zorder=4, color='white')

ax.set_xlim(0, 500)
ax.set_title(f'NMF Separation & Vowel & Chord Analysis\nFile: {selected_path.split("/")[-1]}')
ax.set_xlabel('Time(s)')
ax.set_ylabel('Frequency (Hz)')
ax.legend(loc='upper right')

# 音声再生と可視化を管理するクラス
class AudioVisualizer:

    def __init__(self, full, speech, music, sr):
        # 音声データのマッピング (Full: 元音声、Speech: 音声成分、Music: 音楽成分)
        self.audio_map = {"Full": full, "Speech": speech, "Music": music}
        self.current_audio, self(sr, self.ani, self.is_playing, self.stream,
        self.current_out_pos = full, sr, None, False, None, 0

    # 再生モード切り替えメソッド
    def set_mode(self, label):
        self.current_audio = self.audio_map[label]

    def _callback(self, outdata, frames, time_info, status):
        if self.is_playing:
            rem = len(self.current_audio) - self.current_out_pos
            if rem <= 0:
                outdata.fill(0)

```

```

        return
    chunk = min(frames, rem)
    outdata[:chunk, 0] = self.current_audio[self.current_out_pos :
                                             self.current_out_pos + chunk]
    if chunk < frames: outdata[chunk:, 0] = 0
    self.current_out_pos += chunk

# アニメーション更新(再生位置に応じて表示を更新)
def update(self, frame):
    if self.is_playing:
        t = self.current_out_pos / self.sr
        if t >= total_duration:
            self.stop(None)
            return v_line, vowel_text, chord_text
        v_line.set_xdata([t])
        f_idx = int(t * SR / size_shift)
        if 0 <= f_idx < len(likelihoods_results):
            vowel_text.set_text(f"Vowel: {vowel_list[likelihoods_results[f_idx]]}")
            c_idx = estimated_chord_indices[f_idx]
            chord_text.set_text(f"Chord: {chord_names_list[c_idx]} if c_idx >= 0 else 'N'")
    return v_line, vowel_text, chord_text

# 再生開始メソッド
def play(self, event):
    if not self.is_playing:
        self.current_out_pos = 0
        self.stream = sd.OutputStream(samplerate=self.sr,
                                      channels=1, callback=self._callback)
        self.stream.start()
        self.is_playing = True
        self.ani.event_source.start()

# 再生停止メソッド
def stop(self, event):
    if self.stream:
        self.stream.stop()
        self.stream.close()
        self.stream = None
    self.is_playing = False
    self.ani.event_source.stop()

```

```

visualizer = AudioVisualizer(x_l, x_speech, x_music, SR)
visualizer.ani = animation.FuncAnimation(fig, visualizer.update, interval=20, blit=True,
cache_frame_data=False)
visualizer.ani.event_source.stop()

# GUI ボタンの配置
btn_play = Button(plt.axes([0.7, 0.05, 0.1, 0.075]), 'Play', color='lightgreen')
btn_stop = Button(plt.axes([0.82, 0.05, 0.1, 0.075]), 'Stop', color='tomato')
btn_play.on_clicked(visualizer.play)
btn_stop.on_clicked(visualizer.stop)

# モード切り替え用ラジオボタン
radio = RadioButtons(plt.axes([0.02, 0.4, 0.12, 0.2]), ('Full', 'Speech', 'Music'))
radio.on_clicked(visualizer.set_mode)

# GUI の表示
plt.show()

```

2 問題点と考察

スペクトログラム、基本周波数の表示に加え、今回実装した機能は、推定した母音、和音の表示、NMFによる元音声と音声、音楽の分離である。今回の課題1までに学習してきた音楽音響信号の処理のメソッドを参考にし、それらをこのプログラムにまとめて実装した。また、基本周波数の推定には、倍音加算法を用いた。各候補周波数について倍音成分の対数スペクトルを加算し、最大値を与えるものを基本周波数として決定した。

- 母音推定

今回学習に用いた音声は、私が”あいうえお”と発生したものを録音し、それぞれの母音を切り取ったものである。演習では、私の音声で学習したモデルを用いて私の音声を母音推定していたため、ある程度の精度はあったが、それ以外の音声で行った場合、かなり精度が悪かった。これは、男性と女性の声質の違い、音声だけではなく音楽も音響信号に含まれていることなどに起因していると考えられる。精度を上げる方法として、各母音にモデルを複数用意し、指定された結果が最も多かったものをその結果とする方法が考えられる。

- NMF

基底の数、それに伴う特徴行列 H、U を変化させたが、高い精度で音声と音楽を分離させることはできなかった。実際の音響信号を聞いてみると、音声だけ分離されている瞬間と、音楽だけが分離されている瞬間があった。