計算機科学実験 4 音声 レポート 2

京都大学工学部情報学科
計算機科学コース 3 回生
学生番号: 1029-33-1415
氏名: 安済翔真

2023 年 12 月 28 日

目次

1　機能・プログラム説明

課題 2 では音響信号を操作するグラフィカルユーザーインタフェースを作成した。この章では作成した具体的な機能について説明する。画面全体は図 1 のようになっている。

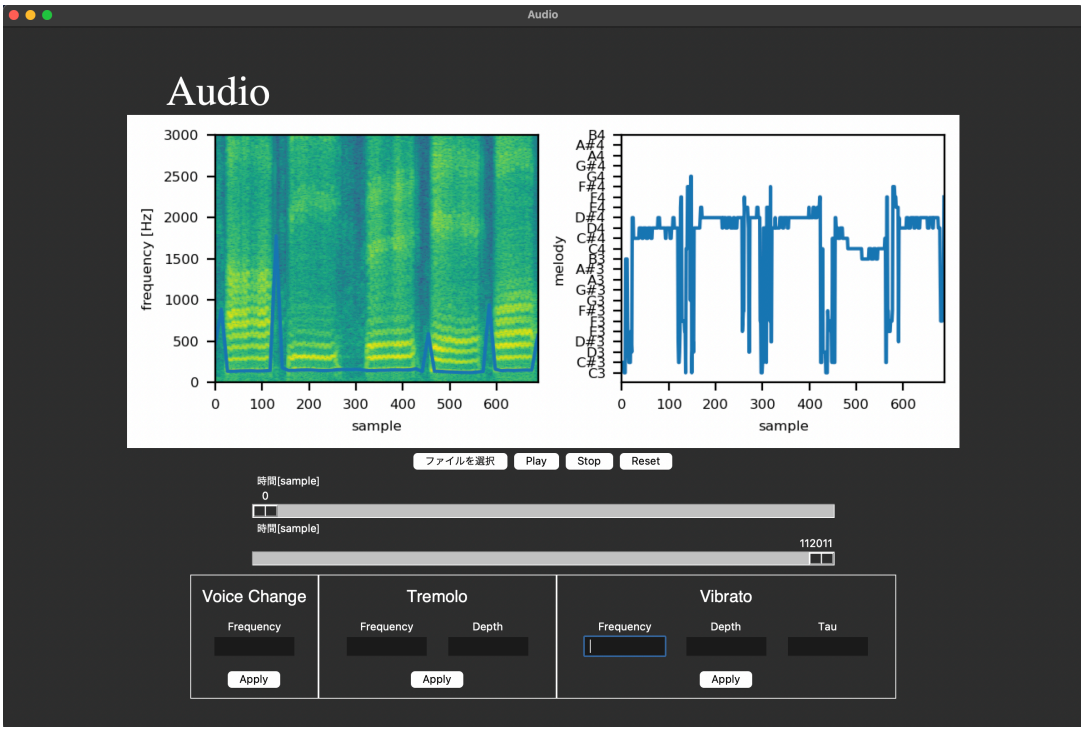

図 1　画面全体

1.1　ファイル選択

ファイル選択機能を実装した。ファイル選択ボタン (図 2) を押すと、ファイル選択ダイアログが表示される。選択したファイルを読み取り、操作対象の音声データとして扱う。

図 2　ファイル選択ボタン

ファイル読み取りのプログラムはコード 1, 2 の通りである。コード 1 では tkinter の filedialog を用いてファイル選択ダイアログを表示し、選択したファイル名を取得する。コード 2 では librosa を用いて音声データを読み取る。

1　ファイル読み取り 1

```
filename = tk.filedialog.askopenfilename()
self._c.load_file(filename)
```

2　ファイル読み取り 2

```
def load_waveform(filename):
  x, _ = librosa.load(filename, sr=SR)
  return x
```

1.2　ビブラート

選択したファイルの音声にビブラートをかける機能を実装した。コード 3 はビブラートをかけるプログラムである。generate_sinusoid 関数は、サンプリング周波数 S、周波数 F、長さ T を引数に取り、サンプリング周波数 S で周波数 F の正弦波を長さ T 秒分生成する関数である。引数に、元の波形、サンプリング周波数 S、ビブラートの周波数 F、ビブラートの深さ D、ビブラートの遅れ時間 τ を取り、次の式に従って波形を変形させた。

$$\begin{cases} x'_t = x_{t-\tau(t)} \\ \tau(t) = \tau 0 + D\sin(2\pi F t) \end{cases} \tag{1.1}$$

3　ビブラート

```
def vibrato(waveform, sampling_rate: int, frequency: float, depth: float, tau: float):
  vibrato_waveform = generate_sinusoid(sampling_rate, frequency, len(waveform) / sampling_rate)
  changed = waveform.copy()
  for i in range(len(waveform)):
    taui = int(tau + depth * vibrato_waveform[i])
    if taui < 0:
      taui = 0
    elif taui >= len(waveform):
      taui = len(waveform) - 1
    changed[i] = waveform[i - taui]
  return changed
```

1.3　ボイスチェンジ

選択されたファイルの音声にボイスチェンジをかける機能を実装した。コード 4 はボイスチェンジをかけるプログラムである。sin 波を生成して、元の波に乗算することで、ボイスチェンジを実現した。

4　ボイスチェンジ

```
def voice_change(waveform, sampling_rate, frequency):
  duration = len(waveform)
  sin_wave = generate_sinusoid(sampling_rate, frequency, duration / sampling_rate)
  sin_wave = sin_wave * 0.9
  return waveform * sin_wave
```

1.4 トレモロ

選択したファイルの音声にトレモロをかける機能を実装した。コード 5 はトレモロをかけるプログラムである。次の式で得られる信号を入力音声に乗算することで、トレモロを実現した。

$$a(t) = 1 + D\sin(2\pi F t) \tag{1.2}$$

5　トレモロ

```
def tremolo(waveform, sampling_rate, frequency, depth):
  tremolo_waveform = generate_sinusoid(sampling_rate, frequency, len(waveform) / sampling_rate)
  changed = waveform * (1.0 + depth * tremolo_waveform)
  return changed / np.max(np.abs(changed))
```

1.5 加工リセット機能

音声への加工をリセットする機能を実装した。コード 6 は加工をリセットするプログラムである。表示対象の波形データに元の波形データを代入することで、加工をリセットした。

6　加工リセット

```
  self.__waveform = self.__original.copy()
```

1.6 スペクトログラムの表示

加工後の音声のスペクトログラムを表示する機能を実装した。ビブラート、ボイスチェンジ、トレモロの各加工を行うと図が自動で更新され、加工後の音声のスペクトログラムが表示される。コード 7 はスペクトログラムを計算するプログラムである。numpy の hamming 関数を用いて窓掛けを行い、FFT を行う。その後、対数振幅スペクトルを計算し、配列に保存する。

7　スペクトログラム計算

```
def spectrogram(waveform, size_frame, size_shift):
  spectrogram = []
  hamming_window = np.hamming(size_frame)

  for i in np.arange(0, len(waveform) − size_frame, size_shift):
```

```
        idx = int(i)
        x_frame = waveform[idx: idx + size_frame]

        # 窓掛けしたデータを FFT
        fft_spec = np.fft.rfft(x_frame * hamming_window)

        # 振幅スペクトルを対数化
        fft_log_abs_spec = np.log(np.abs(fft_spec))

        # 配列に保存
        spectrogram.append(fft_log_abs_spec)
    return spectrogram
```

コード 8 はスペクトログラムを表示するプログラムである。matplotlib の imshow 関数を用いてスペクトログラムを表示する。

8　スペクトログラム表示

```
self.__ax.imshow(
    np.flipud(np.array(spectrogram).T),
    extent=[0, len(spectrogram), 0, SR / 2],
    aspect='auto',
    interpolation='nearest',
)
self.__ax.set_ylim(0, 3000)
```

1.7　基本周波数の表示

加工後の音声の基本周波数を表示する機能を実装した。ビブラート、ボイスチェンジ、トレモロの各加工を行うと図が自動で更新され、加工後の音声の基本周波数が表示される。コード 9 は基本周波数を計算するプログラムである。まず、numpy の correlate 関数を用いて自己相関係数を計算する。その後、ピークを検出し、ピークのインデックスを取得する。最後に、ピークのインデックスから基本周波数を計算する。

9　基本周波数計算

```
def get_f0(waveform, sampling_rate):
    autocorr = np.correlate(waveform, waveform, 'full')
    autocorr = autocorr[len(autocorr) // 2:] # 不要な前半を捨てる

    # ピークを検出
    peak_indices = [i for i in range(len(autocorr)) if is_peak(autocorr, i)]
    peak_indices = [i for i in peak_indices if i != 0] # 最初のピークは除く

    if len(peak_indices) == 0:
```

```
    return 0

  max_peak_index = max(peak_indices, key=lambda index: autocorr[index])

  # 基本周波数を推定
  f0 = sampling_rate / max_peak_index
  return f0
```

コード 10 は基本周波数を表示するプログラムである。matplotlib の plot 関数を用いて基本周波数を表示する。スペクトログラムと同じ x 軸を用いるため、x 軸のデータはスペクトログラムのデータを用いる。

10　基本周波数表示

```
  x_data = np.linspace(0, len(spectrogram), len(f0s))
  self.__ax.plot(x_data, f0s)
```

1.8　音程の表示

加工後の音声の音程を表示する機能を実装した。ビブラート、ボイスチェンジ、トレモロの各加工を行うと図が自動で更新され、加工後の音声の音程が表示される。コード 11 は音程を計算するプログラムである。nn2hz 関数は MIDI ノートナンバーを周波数に変換する関数である。shs 関数はスペクトルを用いて音程を計算する関数である。候補の音程 (NOTES) の各周波数について、スペクトルの対数振幅スペクトルを用いて尤度を計算する。尤度を計算する際は、対象の音程の「倍音」の強さも考慮した。特に、倍音の強さは 0.8 の指数関数的な減衰を考慮することで精度が大幅に向上した。

11　音程計算

```
def nn2hz(nn):
  return 440.0 * 2 ** ((nn - 69) / 12.0)

def shs(spectrum, sample_rate, size_frame):
  likelihood = np.zeros(len(NOTES))
  for i in range(len(likelihood)):
    base_freq = nn2hz(NOTES[i])
    for j in range(1, 16):
      freq = base_freq * j
      fft_idx = int(freq * size_frame / sample_rate)
      likelihood[i] += 0.8**j * np.exp(spectrum[fft_idx])
  return NOTES[np.argmax(likelihood)]
```

コード 12 は音程を表示するプログラムである。matplotlib の plot 関数を用いて音程を表示する。yticks で y 軸のラベルを設定している。ラベルの内容は C3 から B4 までの音程を表示することで、表示される音程がわかりやすくなるようにした。

12　音程表示

```
plt.plot(list(map(lambda x: x - NOTES[0], melody)))
plt.yticks(np.arange(24),
        list(["C3", "C#3", "D3", "D#3", "E3", "F3", "F#3", "G3",
              "G#3", "A3", "A#3", "B3", "C4", "C#4", "D4", "D#4",
              "E4", "F4", "F#4", "G4", "G#4", "A4", "A#4", "B4"])
        )
```

1.9　音声の再生・停止

加工後の音声を再生・停止する機能を実装した。コード 13 は音声を再生するプログラムである。既に音声が再生中の場合は、一旦再生を停止してから再度再生をすることで、同じ音声を連続して再生することができる。

13　音声再生

```
if self.__play_obj is not None:
    self.__play_obj.stop()
self.__play_obj = self.__wave_obj.play()
```

コード 14 は音声を停止するプログラムである。音声が再生中の場合のみ、音声を停止する。

14　音声停止

```
if self.__play_obj is None:
    return
self.__play_obj.stop()
self.__play_obj = None
```

1.10　音声加工区間の限定

加工対象の音声の区間を限定する機能を実装した。選択した区間に対してのみ、ビブラート、ボイスチェンジ、トレモロの各加工が行われる。コード 15 は音声加工区間を限定するプログラムである。start の値は end の値より大きくならないようバリデーションを行った。

15　音声加工区間の限定

```
def set_start(self, start: int):
    if start >= self.__end - self.MIN_SIZE:
        return
    self.__start = start

def set_end(self, end: int):
    if end <= self.__start + self.MIN_SIZE:
        return
```

```
9       self.__end = end
```

コード 16 は区間を限定する UI を表示するプログラムである。tkinter の Scale を用いて、スライダーの UI を表示している。このコードにより、図 3 のような UI が表示される。

16　区間を限定する UI

```
1   self.slider = tk.Scale(
2     command=self.__command,
3     master=self._frame,
4     from_=from_,
5     to=to,
6     label=u' 時間 [sample]',
7     orient=tk.HORIZONTAL,
8     length=700,
9     width=15,
10  )
```

図 3　区間を限定する UI

2　実行例とテスト

2.1　ファイル選択

ファイル選択ボタンを押すと、図 4 のようなファイル選択ダイアログが表示される。これにより、ファイル選択ボタンが正しく動作していることが確認できる。

2.2　ビブラート

ファイルを選択すると、図 5 のようにビブラートの周波数と深さと定数値を指定する UI が表示される。frequency, depth, tau の値を入力して Apply ボタンを押すと、音声にビブラートがかかることを確認した。

2.3　ボイスチェンジ

ファイルを選択すると、図 6 のようにボイスチェンジの周波数を指定する UI が表示される。frequency の値を入力して Apply ボタンを押すと、音声にボイスチェンジがかかることを確認

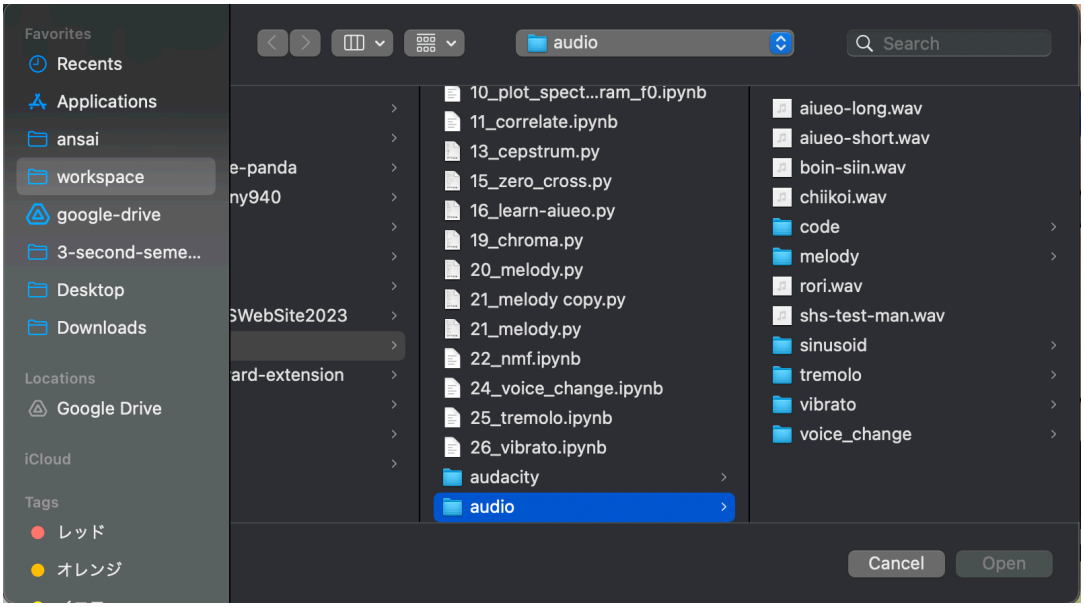

図 4　ファイル選択ダイアログ

図 5　ビブラートの操作パネル

した。

2.4　トレモロ

ファイルを選択すると、図 7 のようにトレモロの周波数と深さを指定する UI が表示される。frequency, depth の値を入力して Apply ボタンを押すと、音声にトレモロがかかることを確認した。

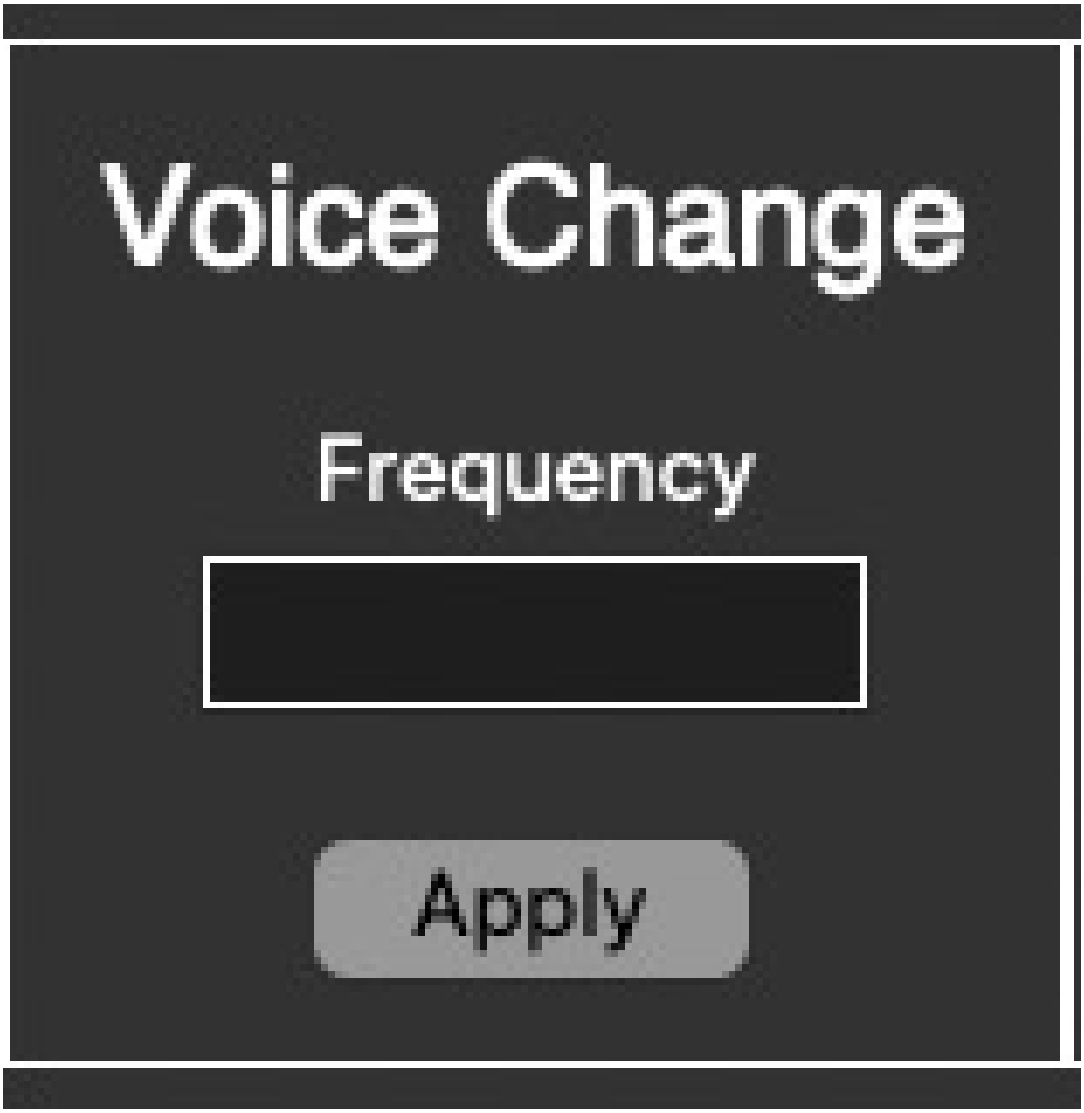

図 6　ボイスチェンジの操作パネル

2.5　加工リセット

ファイルを選択すると、図 8 のようにリセットボタンが表示される。リセットボタンを押すと、加工がリセットされることを確認した。

2.6　スペクトログラムの表示

ファイルを選択すると、図 9 のようにスペクトログラムが表示される。また、音声に加工処理を施すと、スペクトログラムが自動で更新される。図 9 は「あいうえお」の音声にトレモロをかけた後のスペクトログラムである。これにより、スペクトログラムが正しく表示されていることが確認

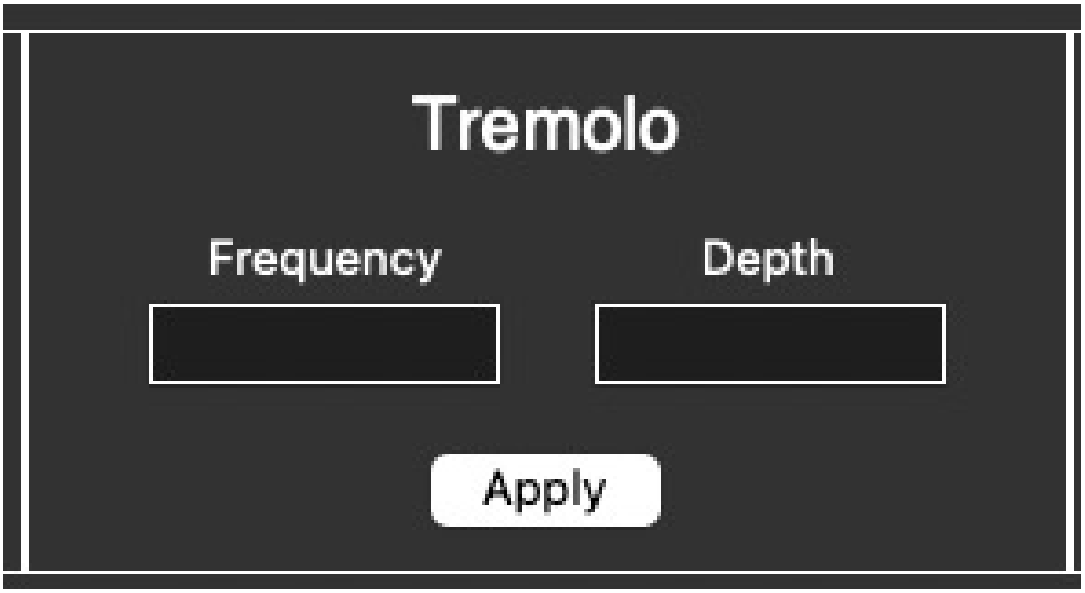

図 7　トレモロの操作パネル

図 8　加工リセットボタン

できる。

2.7　基本周波数の表示

ファイルを選択すると、図 10 のように基本周波数が表示される。図 10 は「あいうえお」の音声にトレモロをかけた後の基本周波数である。これにより、基本周波数が正しく表示されていることが確認できる。

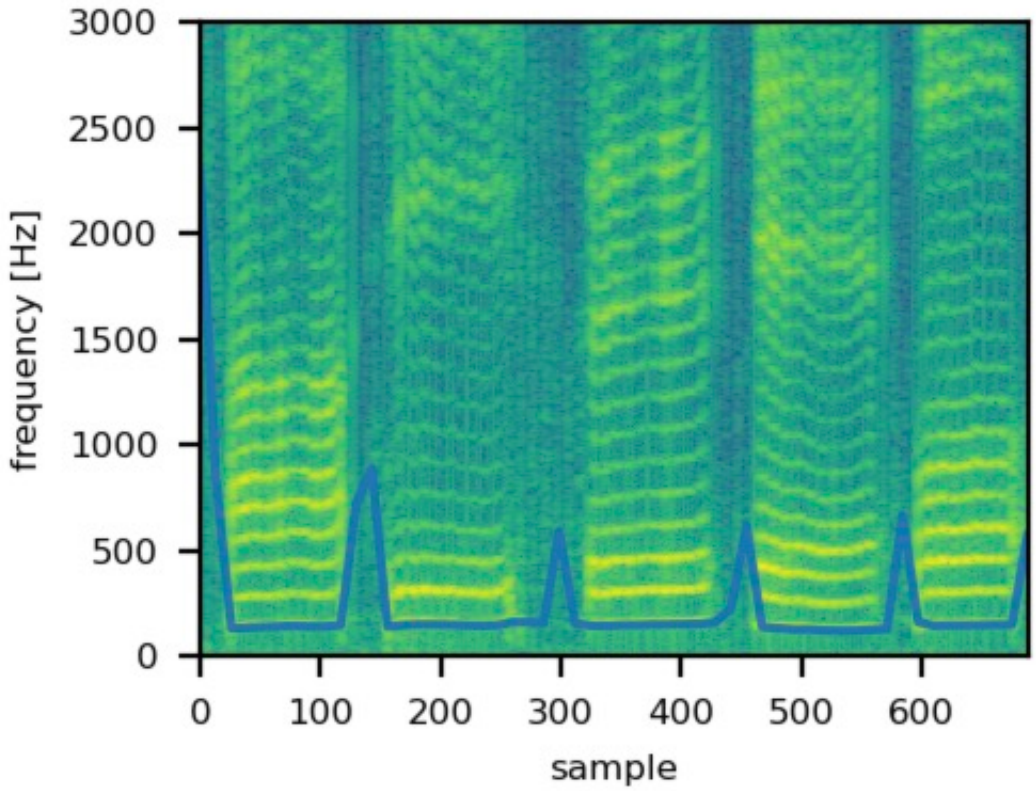

図 9　スペクトログラム

2.8　音程の表示

ファイルを選択すると、図 11 のように音程が表示される。図 11 は計算機科学実験及演習 4 音響信号処理の SHS 確認用サンプル音声にトレモロをかけた後の音程である。これにより、音程が正しく表示されていることが確認できる。

2.9　音声の再生・停止

ファイルを選択すると、図 12 のような再生・停止ボタンが表示される。再生ボタンを押すと音声が再生され、停止ボタンを押すと音声が停止することを確認した。音声を加工した後に再生ボタンを押すと、加工後の音声が再生される。また、再生中に再度再生ボタンを押すと、一旦再生を停止してから再度再生をすることを確認した。

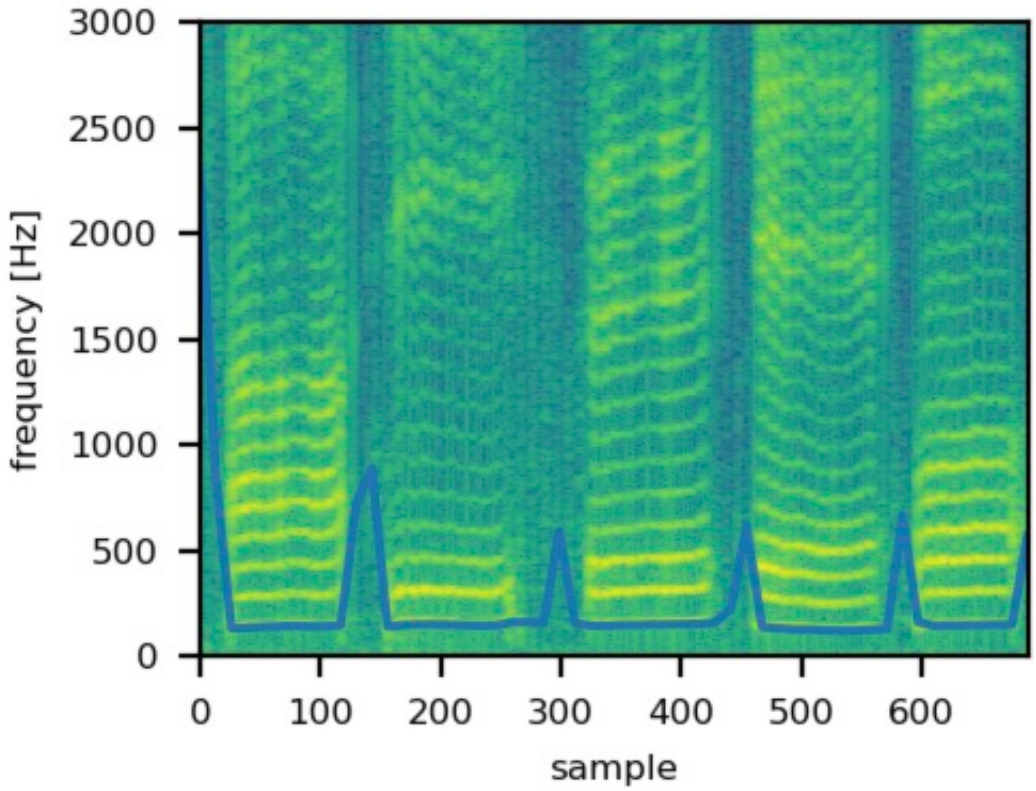

図 10　基本周波数

2.10　音声加工区間の限定

ファイルを選択すると、図 3 のようなスライダーが表示される。スライダーで区間を限定すると、加工対象の音声の区間が限定されることを確認した。また、開始を表すスライダーの値が終了を表すスライダーの値より大きくならないようになっていることを確認した。

3　工夫点

3.1　ファイル選択機能

分析対象の音声のファイルは、コードに直接記述するのではなく、ファイルを選択する UI を用いて選択するようにした。これにより、分析対象の音声を簡単に変更できるようになった。

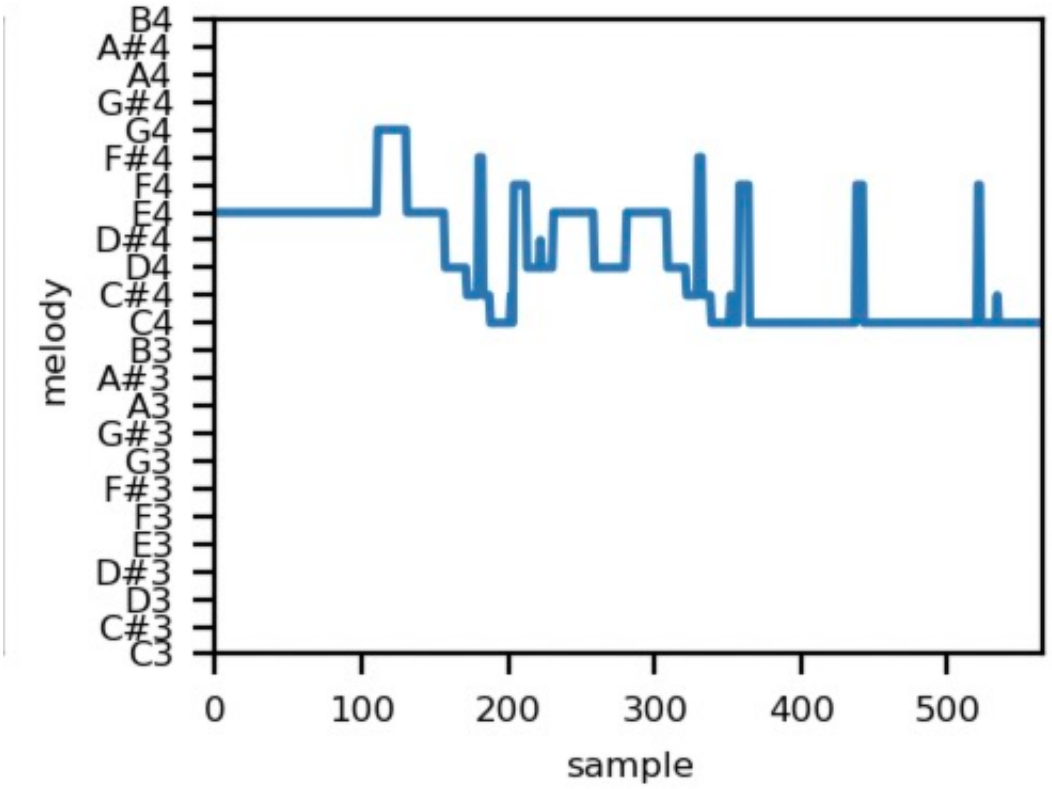

図 11　音程

3.2　区間選択機能

加工対象の音声の区間を限定できるように工夫を行なった。また、start の値が end の値より大きくならないようバリデーションを行った。

3.3　加工リセット機能

音声の加工をリセットできるように工夫を行なった。

3.4　音程の表示

音程の図を表示する際に、y 軸には C3 から B4 までの音程を表示するようにした。これにより、ノート番号を表示する場合と比べて、表示される音程がわかりやすくなった。

図 12　再生・停止ボタン

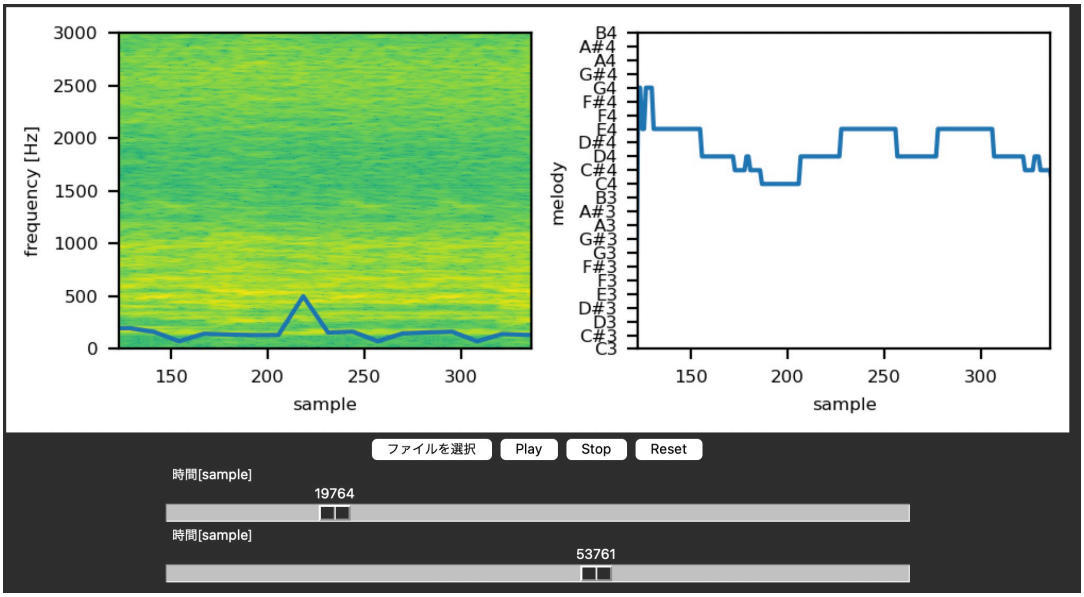

図 13　区間を限定した図

3.5　音声の再生・停止機能

音声の再生・停止ボタンを実装した。これにより、加工後の音声を UI 上で実際に確認することが可能になった。

3.6　コード設計の工夫

今回作成した GUI アプリでは、コードを core, view, controller の 3 つに分割した。core は音声の分析を行う部分、view は GUI の表示を行う部分、controller は view と core の橋渡しを行う部分である。このように分割することで、コードの可読性・保守性を向上させた。

4　考察

今回は、音声信号を操作する GUI アプリを作成した。コードを MVC アーキテクチャに従って整理をすることで、コードの見通しをよくすることができた。また、音声を加工した時に図が自動的に更新されるといった複雑な UI を作成する勉強になった。次の課題では音声の分析をリアルタイムで行う GUI システムを作成する予定である。

5　参考文献

計算機科学実験及演習 4 音響信号処理
http://www.sap.ist.i.kyoto-u.ac.jp/members/inoue/le4-audio/