

計算機科学実験及演習 4 (前半：音響信号処理)

課題 2 音響信号操作 GUI 作成

長谷真暉 (1029341181)

提出日: 2024 年 11 月 8 日

音響信号ファイルを読み込み、音響信号を操作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。少なくとも以下の操作を可能にすること。

1. ボイスチェンジ
2. トレモロ

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。例えば以下のような改良が考えられるが、もちろんこれ以外の改良でも構わない。創意工夫すること。

- エフェクトのパラメータを GUI 上で変更できる。
- 選択した区間だけにエフェクトを施す。
- ビブラート、エコー、コンプレッサー、ディレイ、LPF（ローパスフィルタ）など他のエフェクタを実装する。

1 概要

wav ファイルを読み込み (Load ボタン)、ボイスチェンジ・トレモロ・ビブラート・エコー・コンプレッサー・LPF の 6 種類のエフェクタを選択した区間に適用し、再生する (Play ボタン) ことができる GUI である (図 6)。再生ボタンには、2 種類あり、もとの音響信号の再生 (Play Default) と、エフェクタの適用後の再生 (Play) が可能である。それぞれのエフェクタは、重ねて適用される。また、effector clear ボタンを押すと、これまでの変更履歴がすべて消え、もとの音響信号に戻る。effecta ボタンでは、これまでに適用したエフェクタを確認することができる。再生時は、再生時間を表す再生バー、音響信号の振幅を表す振幅メーターと、周波数の振幅をあらわす周波数メーターが動く。画面下部では、変更した音響信号のスペクトログラム、波形、音量の 3 つをそれぞれ表示できる。

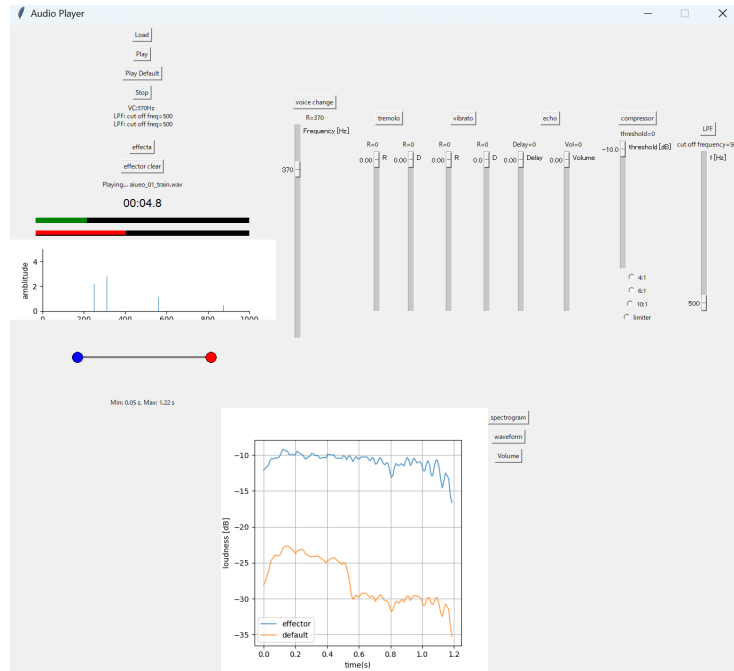


図 1: GUI

2 読み込み・再生時の実装

読み込み・再生時に関わる GUI について説明する。図 6 の左上にあるボタンやラベル、再生バーなどを、上から順に説明していく。

2.1 Load ボタン

読み込みファイルのフォルダを開き、任意の wav をクリックすることで読み込む。また、プログラム上で、GUI 実行時に初期画面で一つのファイルを初期音声データとしてロードしておくことができる。また、ロード時には、ロードした音声ファイルのパスが表示される（図 2）。

Loaded: C:/Users/Owner/KU/le4-audio-kuis-Nagaya/rec/a.wav

図 2: Load 中のラベル

2.2 Play ボタン / Play Default ボタン

pygame を用いて再生している。ロード時と同様に、再生している音声ファイルの名前が表示される（図 4）。また、エフェクタを適用した音声データに対しては、changed と表示される（図 3）。再生バー・再生時間・音量レベルメーター・スペクトルメーターの 4 つが同時に稼働する。

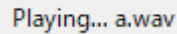


図 3: play 中のラベル



図 4: play 中のラベル (変更後)

2.3 Stop ボタン

再生を止め、表示を「Stopped」にする（図 5）。



図 5: stop 中のラベル

2.4 再生時間の表示ラベル

音声の再生とは干渉しないが、再生バーや音量レベルメーターなどと同時に動かすために、Thread のライブラリを用いている。再生時間の表示では、0.1 秒ごとにカウンタを増やし、分と秒に直している。実際のプログラムは以下の通り。

ソースコード 1: time counter

```
1 def start_time_counter(self):
2     def count_time():
3         self.elapsed_time = 0
4         while pygame.mixer.music.get_busy():
5             self.update_time_label()
6             time.sleep(0.1)
7             self.elapsed_time += 1
8             self.time_label.config(text=AudioPlayer.TIME_INIT_STR)
9
10    Thread(target=count_time, daemon=True).start()
11
12 def update_time_label(self):
13     minutes, seconds = divmod(self.elapsed_time, 600)
14     seconds, microseconds = divmod(seconds, 10)
15     time_str = f"{minutes:02}:{seconds:02}.{microseconds:01}"
16     self.time_label.config(text=time_str)
```

2.5 音量レベルメーター

再生時間の表示ラベルと同様に、スレッドを用いている。描画では、背景の黒い長方形に新しく長方形を重ねる形で表示している。スレッドの更新はプログラム上で変更でき、処理速度に応じて変える。音量のレベルメーターのみでは、0.03 秒間隔の更新が可能だった。ここでの音量レベルとは、音声データの波形の振幅を用いている。

2.6 再生バー

スレッドや描画に関して、音量レベルメーターと同様。pygame で再生している音声に対して、現在の再生時間を取得できる。そこから描画する長方形の大きさを決定している。

2.7 スペクトルメーター

スレッドに関して、音量レベルメーターと同様。再生直前にスペクトログラムの配列を作成し、現在の再生時間に応じたスペクトルを表示している。スペクトルは、棒グラフのような形で表示している。

3 エフェクタの実装

まず、エフェクタに関する GUI について説明する。次に、それぞれのエフェクタについて説明する。すべてのエフェクタは、エフェクタの適用ボタンとパラメータ用のバーからできている。フォーマットをそろえることで見た目を統一している。

3.1 effector ボタン

エフェクタは重ねて適用できる。ボタンを押すと、現在適用されているエフェクタが重ねた順番にパラメータとともに表示される。

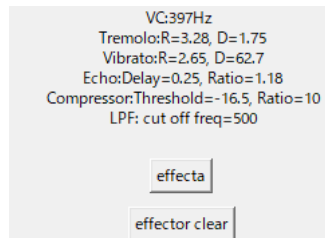


図 6: effector

3.2 effector clear ボタン

エフェクタの履歴をすべて消去し、もとの音源に戻す。

3.3 エフェクタの適用範囲の指定バー

左側の青丸が始点、右側の赤丸が終点となる。青丸と赤丸は交差しないようになっている。その下に見えるラベルに書かれた秒数に対してエフェクタが適用される。例として図 7 を示す。図 7 では、音源の中間のみ LPF を適用している。

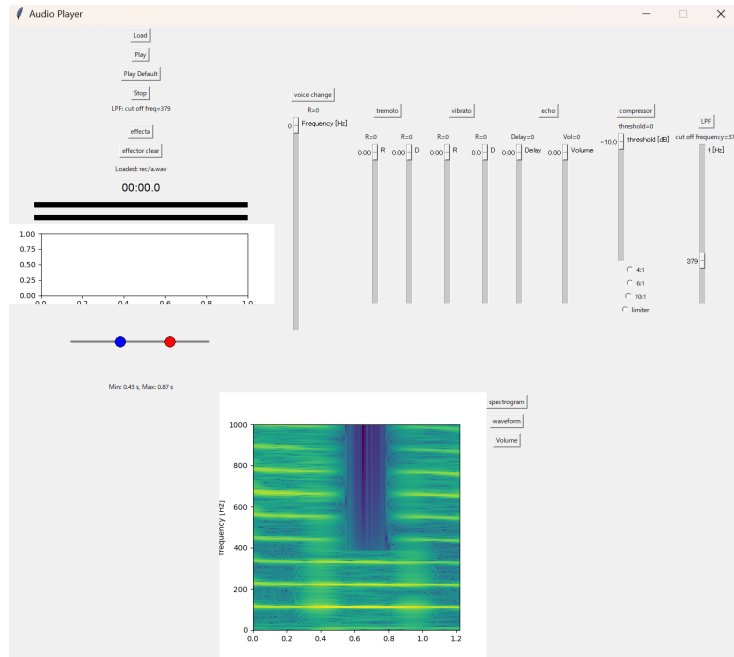


図 7: エフェクタの範囲指定

3.4 ボイスチェンジ

スケールでは、0Hz から 2000Hz までの周波数帯から加える信号の周波数を決定する。

3.5 トレモロ

スケールでは、R は 0.00 から 10.00、D は、0.00 から 5.00 まで変えられる。

3.6 ビブラート

スケールでは、R は 0.00 から 10.00、D は、0.0 から 300.0 まで変えられる。

3.7 エコー

エコーでは、以下のように出力信号を定義した。

$$y(t) = x(t) + r * x(t + \tau)$$

スケールでは、Delay は 0.00s から 1.00s、Volume は、0.00 から 2.00 まで変えられる。

3.8 コンプレッサー

コンプレッサーでは、以下のように出力を定義した。
まず、音量 (dB) を計算する。

$$\text{vol} = 20 \cdot \log_{10}(|x(t)|).$$

次に、threshold に基づき、大きな音量を ratio を用いて小さくする。vol > threshold の場合、

$$y(x) = \begin{cases} \text{threshold} + \frac{x(t) - \text{threshold}}{\text{ratio}} & (x(t) > 0) \\ -\text{threshold} + \frac{x(t) + \text{threshold}}{\text{ratio}} & (x(t) < 0) \end{cases}$$

スケールでは、threshold は -10.0dB から -40.0dB まで変えられる。また、ラジオボタンで、レシオを 1:4、1:6、1:10、limiter のように表している。それぞれ ratio を 4、6、10、1000 のように定義している。

3.9 LPF

LPF では、以下のように信号を定義した。

$$\mathcal{F}[y(t)](w) = \begin{cases} \mathcal{F}[x(t)](w) & (\mathcal{F}[x(t)](w) < f_c) \\ 0 & \text{otherwise.} \end{cases}$$

スケールでは、カットオフ周波数を 0Hz から 500Hz まで変えられる。カットオフした際、音量が極端に小さくなったため、信号全体の振幅の最大値を 0.9 とするように波形を定数倍している。

3.10 エフェクタの適用後の処理

エフェクタを適用した際、まず、音声信号を更新している。この更新時に、始点と終点だけが更新されるようにしている。次に、音声の再生では、保存してある wav ファイルのパスが必要になるので、毎回一時ファイルを作成し、エフェクタに通した音声を wav ファイルとして保存し、ファイル名のみをプログラム上で保持している。最後に、適用したエフェクタの情報をテキストとして保持し、effector ボタンで表示できるようにしている。

ソースコード 2: エフェクタの適用後

```
1 self.current_audio_data_changed[self.min_index:self.max_index] = audio_data_changed[self.  
  min_index:self.max_index]  
2  
3 audio_data_changed = (audio_data_changed * 32768.0). astype('int16') # 値の範囲を [-1.0 ~  
  +1.0] から [-32768 ~ +32767] へ変換する  
4 # 音声ファイルとして出力する  
5 with tempfile.NamedTemporaryFile(delete=True, suffix='.wav', dir='rec/temp') as temp_file:  
6     self.temporary_file = temp_file.name  
7     scipy.io.wavfile.write(self.temporary_file, int(self.audio_sr), audio_data_changed)  
8  
9 self.effector_now_label = self.effector_now_label + "VC:" + str(frequency) + "Hz_\\n_"
```

4 グラフの実装

エフェクタを適用した後の音声信号に対して、それぞれグラフを描画している。スペクトログラムと音声の波形は、エフェクタを通した音声のみを表示し、音量では、変更前と後の2つをプロットしている。それぞれボタンが押されてから演算、描画している。