音響信号処理　課題 1 レポート

工学部情報学科計算機科学コース 3 回生 1029-31-6722　末原　剛志

2021 年　1 月 10 日

1　実施内容

1.1　課題

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フェースを作成する．その際，音響信号のスペクトログラム，基本周波数，母音などの何らかの識別結果を最低限表示できるようにする．余力があれば，さらに使いやすいものになるような改良を行う．

1.2　実装目標

スペクトログラムは常に表示されるようにし，その図に重なるように基本周波数，音量，母音推定の結果をグラフで表示できるようにする．各グラフをボタン等によって切り替えることができるようにし，それに伴って縦軸のラベルも変化するように実装する．また，右下にはスライドバーによって選択された時間のスペクトルが表示されるようにし，同時にその時間における基本周波数・音量・母音推定結果も示す．余裕があれば音響信号の再生・停止・再生時間の表示なども目指す．

1.3　実装結果

作成したグラフィカルユーザーインターフェースを以下の図 1に示す．

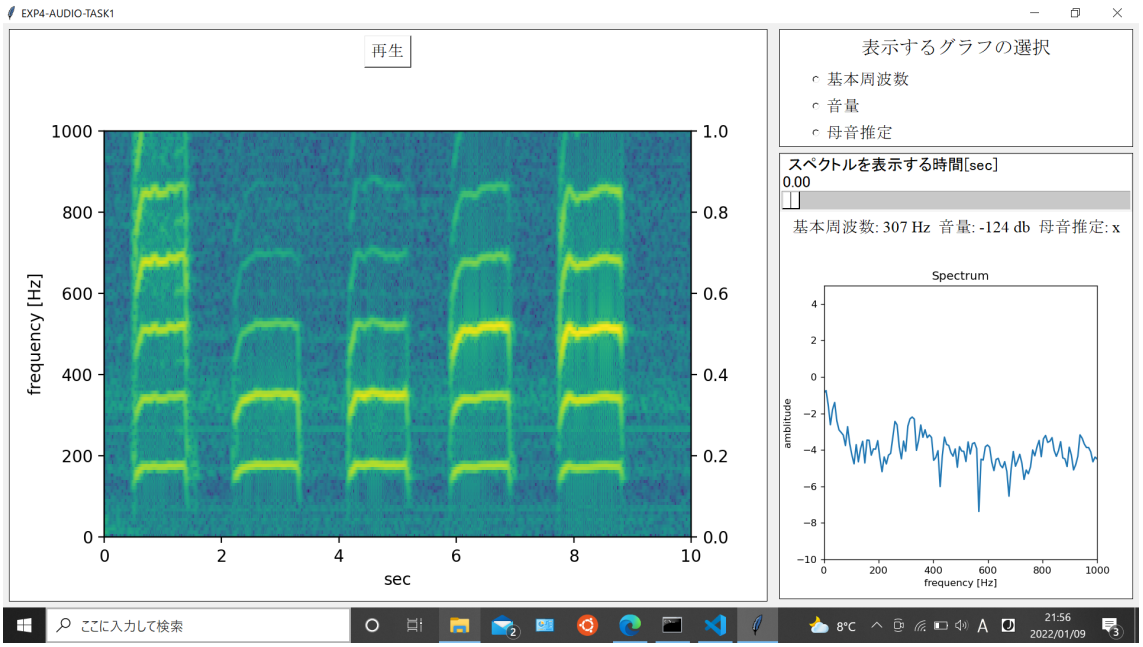

図1　作成したグラフィカルユーザーインターフェース

また，以下図 2・図 3・図 4には各種グラフを選択した際の表示の様子を示す．なおスライドバーの時間選択によるスペクトルの表示は 5.00sec に固定している．

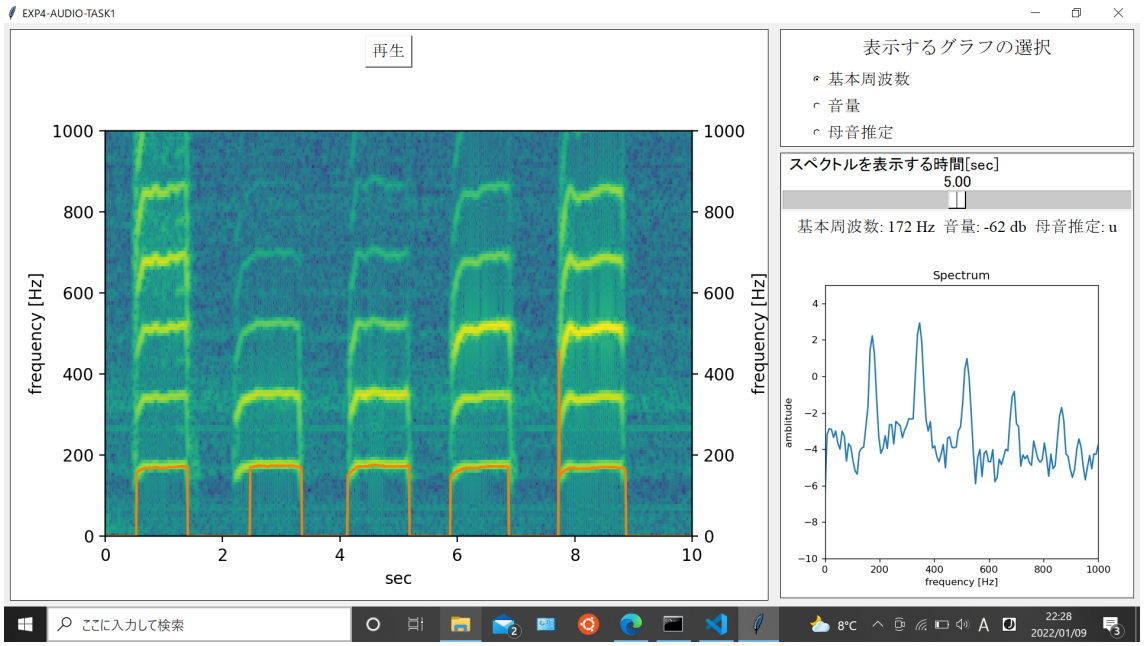

図2　基本周波数のグラフを選択

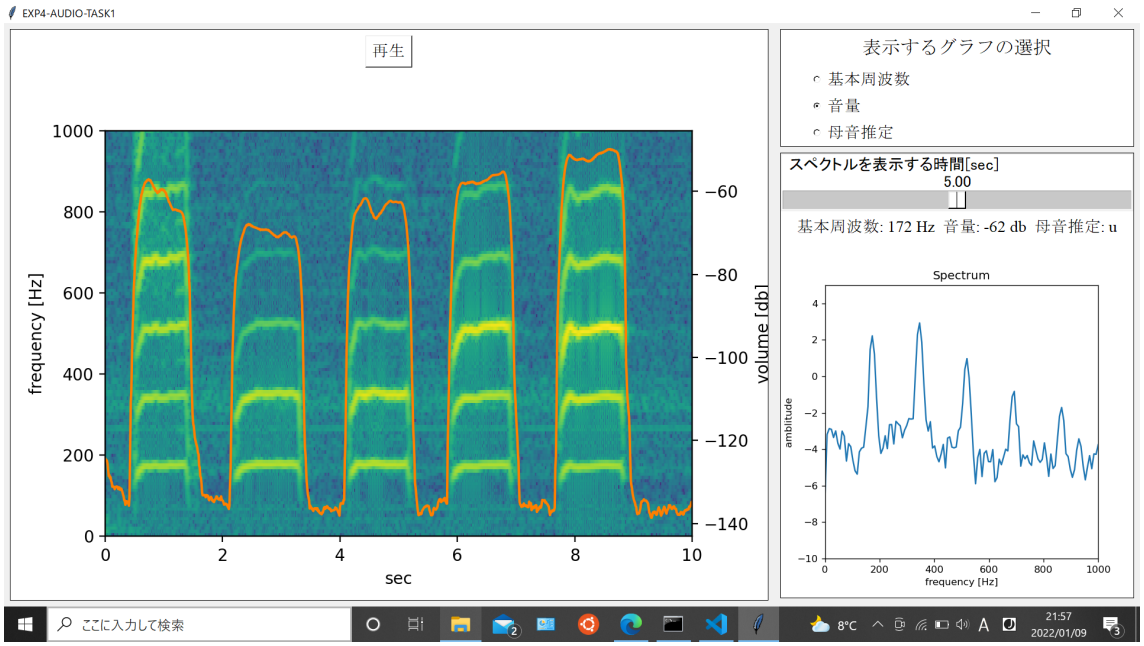

図3　音量のグラフを選択

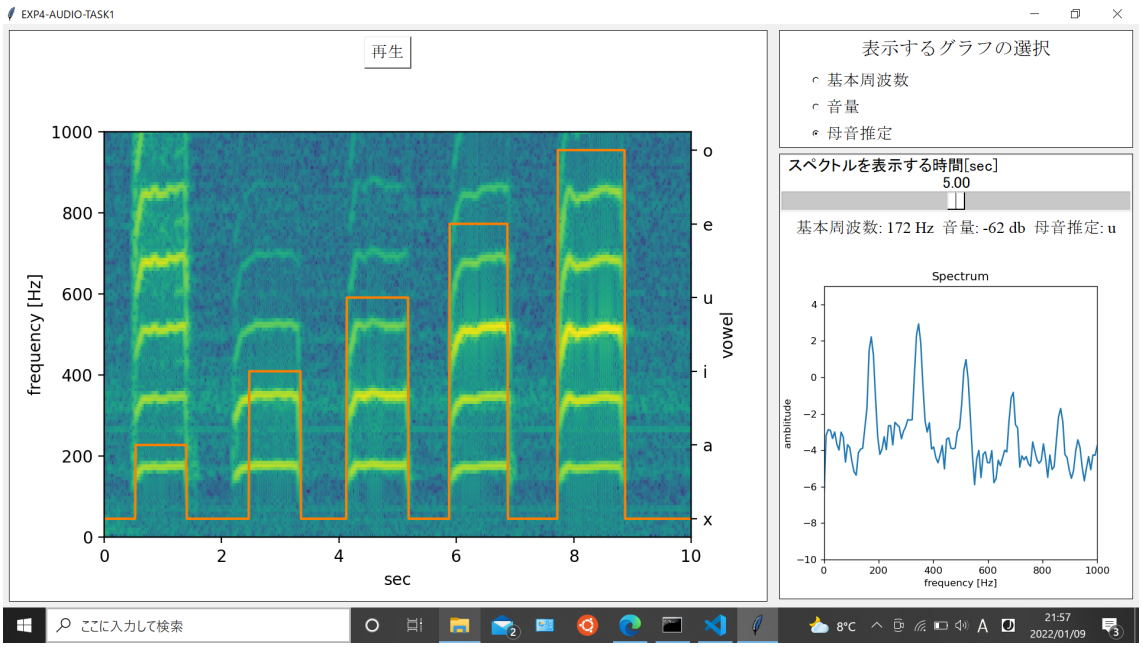

図4　母音推定のグラフを選択

2　実装内容

2.1　全体の構成

作成したグラフィカルユーザーインターフェース (1 の図 1) は，以下の図 5で示すようにメインフレーム上の 3 フレームから構成されている．以降，音響信号処理部分の実装とグラフィカルユーザーインターフェース部分の実装に分けて説明していく．

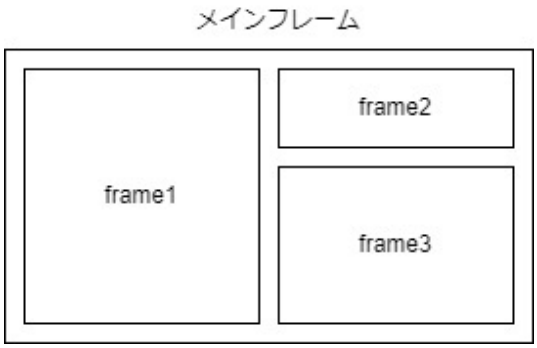

図5　グラフィカルユーザーインターフェースのフレーム構成

2.2 音響信号処理の実装

2.2.1 使用した言語とライブラリ

今回の演習では Python を使用した．また，実装にあたって音声処理ライブラリ librosa と演算用ライブラリ numpy を使用した．

2.2.2 関数

以下の表 1に作成・使用した主な関数の一覧を示す．

表1 作成・使用した主な関数

関数名	返り値の型	詳細
is_peak	boolean	配列のある要素がピークであるかを判定．関数 correlate で使用
correlate	float	自己相関を計算し，最大周波数を推定
zero_cross	int	ゼロ交差数を計算
cepstrum	list	ケプストラム係数を計算
log_likelihood	list	最尤推定で母音を推定
vowel_smoothing	list	母音推定グラフのスムージング
f_zero_smoothing	list	基本周波数グラフのスムージング

2.2.3 グローバル変数

以下の表 2に使用した主なグローバル変数の一覧を示す．

表2 使用した主なグローバル変数

変数名	変数の型	初期値	詳細
SR	int	16000	サンプリングレート
dimension	int	13	抽出するケプストラム係数の次元
model	numpy.ndarray	-	母音推定に用いる学習モデルの音響データ
x	numpy.ndarray	-	推定対象となる音響データ
model_vowel	numpy.ndarray	-	学習モデルのデータを母音ごとに分割したもの
duration	float	-	音響データのサイズ (x の長さを SR で割ることで計算)
size_frame	int	2048	フレームサイズ (2 のべき乗)
hamming_window	numpy.ndarray	-	フレームサイズに合わせたハミング窓
size_shift	float	160	シフトサイズ (0.01 秒ごと)
frequency_zero	list	空リスト	基本周波数を保存
spectrogram	list	空リスト	スペクトログラムを保存
volume	list	空リスト	音量を保存
vowel	list	[x,a,i,u,e,o]	母音のリスト (x は無音であることを意味する)
vowel_presumption	list	空リスト	母音推定の結果を保存

2.2.4 フレームごとの処理

以下に各フレームごとの処理の実装内容を示す.

```
# size_shift 分ずらしながら size_frame 分のデータを取得
for i in np.arange(0, len(x)-size_frame, size_shift):

  # 該当フレームのデータを取得
  idx = int(i)  # arange のインデクスは float なので int に変換
  x_frame = x[idx : idx+size_frame]

  #
  # 基本周波数の推定
  #
  # 自己相関から基本周波数を推定
  x_f0 = correlate(x_frame)

  # ゼロ交差数の計算
  x_zero_cross = zero_cross(x_frame)

  # ゼロ交差数が閾値より大きければ無声音とみなし f0 を 0 とする
  if x_zero_cross > 300:
    frequency_zero.append(0)
  else:
    frequency_zero.append(x_f0)
  #
  # 対数振幅スペクトログラムの計算
  #
  # 窓掛けしたデータを FFT
  fft_spec = np.fft.rfft(x_frame * hamming_window)

  # 複素スペクトログラムを対数振幅スペクトログラムに
  fft_log_abs_spec = np.log(np.abs(fft_spec))

  # 計算した対数振幅スペクトログラムを配列に保存
  spectrogram.append(fft_log_abs_spec[0:128])

  # 音量(デシベルに変換)
  vol = 20 * np.log10(np.mean(x_frame ** 2))
  volume.append(vol)

  # 対数振幅スペクトルをケプストラム係数に
  fft_log_abs_ceps = np.fft.fft(fft_log_abs_spec)

  # 母音推定
  recognition_id = log_likelihood(model_vowel, (np.real(fft_log_abs_ceps[0:dimension])))
  vowel_presumption.append(recognition_id)
```

ここで，特に注意したポイントは

- 有声音と無声音を区別するためのゼロ交差数の閾値を 300 に設定した点
- 対数振幅スペクトログラムの配列から 128 番目までを取り出すことで 1000Hz までの周波数領域に限定している点
- $RMS = \sqrt{\frac{1}{N}\sum_{t=0}^{N-1} x_t^2}$ と $Vol_{dB} = 20log_{10}RMS$ の式から音量を計算した点

- 対数振幅スペクトルからケプストラム係数を抽出する際，フーリエ変換の返り値が複素数のリストであるため，その実部のみを取り出すようにした点

である．

2.2.5 関数 is_peak

以下に関数 is_peak の実装内容を示す．

```
# 配列 a の index 番目の要素がピーク（両隣よりも大きい）であれば True を返す関数
# 自己相関を求める際に使用
def is_peak(a, index):
        # 配列の端は例外的に除く
        if index == 0 or index == len(a) - 1:
                return False
  # 左右の大きいほうの値より大きければ True
        else:
                return  a[index] >= max(a[index-1], a[index+1])
```

この関数は与えられた配列の特定の要素がピークであるかを判定する関数である．ピークであるかどうかはその要素が両隣の要素よりも大きいかどうかで決まるので，両隣の要素の最大値よりも大きければ True を返すようにした．ただし，配列の両端については例外とし必ず False を返すようにしている．この関数は自己相関から基本周波数を推定する関数 correlate 内で使用されている．

2.2.6 関数 correlate

以下に関数 correlate の実装内容を示す．

```
# 基本周波数を求める関数
def correlate(a):
  # 自己相関が格納された，長さが len(x)*2-1 の対称な配列を得る
  autocorr = np.correlate(a, a, 'full')

  # 不要な前半を捨てる
  autocorr = autocorr [len (autocorr ) // 2 : ]

  # ピークのインデックスを抽出する
  peakindices = [i for i in range (len (autocorr )) if is_peak (autocorr, i)]

  # インデックス0 がピークに含まれていれば捨てる
  # インデックス0が 通常最大のピークになる
  peakindices = [i for i in peakindices if i != 0]

  # 返り値の初期化
  max_peak_frequency = 8000.0

  # ピークがなければ最大周波数を返す
  # ピークがあれば自己相関が最大となる(0を含めると2番目に大きい)周期を得る
  if len(peakindices) != 0:
    # 自己相関が最大となるインデックスを得る
    max_peak_index = max (peakindices , key=lambda index: autocorr [index])
    # インデックスに対応する周波数を得る
    max_peak_frequency = SR / max_peak_index

  return max_peak_frequency
```

この関数は与えられた配列とそれを時間方向にシフトしたものとの自己相関を計算し，2 番目に大きいピークが得られるときの周期から基本周波数を推定する関数である．numpy の関数によって配列の自己相関を求めた後，関数 is_peak によってピークとなっているインデックスを取得している．このとき，インデックス 0 は通常最大のピークとなるため含まれていれば削除するようにしている．また，配列によってはピークが存在しない場合もありうるので返り値の初期値を 8000[Hz] とした．ピークが存在すれば自己相関が最大となるインデックスの逆数によって基本周波数を推定している．

2.2.7 関数 zero_cross

以下に関数 zero_cross の実装内容を示す．

```
# 音声波形データを受け取り，ゼロ交差数を計算する関数
def zero_cross(waveform):
  # ゼロ交差数を格納する変数
        zc = 0

  # 配列内で連続する2つの値の正負が異なれば交差しているとし，zcの値を 1 増やす
        for i in range(len(waveform) - 1):
                if(
                        (waveform[i] > 0.0 and waveform[i+1] < 0.0) or
                        (waveform[i] < 0.0 and waveform[i+1] > 0.0)
                ):
                        zc += 1

        return zc
```

この関数は与えられた配列からゼロ交差数を計算する関数である．配列の連続する要素の正負が異なっている組を数え上げるようにしている．有声音であればゼロ交差数は基本周波数の 2 倍に近い値になるので，この値がある閾値より大きければ無声音と判定できる．

2.2.8 関数 cepstrum

以下に関数 cepstrum の実装内容を示す．

```
# ケプストラム係数を計算する関数
def cepstrum(a):
  # フレームサイズ
  size_frame = 2048 # 2のべき乗

  # フレームサイズに合わせてハミング窓を作成
  hamming_window = np.hamming(size_frame)

  # シフトサイズ
  size_shift = 16000 / 100      # 0.01 秒 (10 msec)

  # ケプストラム係数を保存するlist
  cepstrogram = []

  # size_shift分ずらしながらsize_frame分のデータを取得
  for i in np.arange(0, len(a)-size_frame, size_shift):

    # 該当フレームのデータを取得
    idx = int(i)        # arangeのインデクスはfloatなのでintに変換
    a_frame = a[idx : idx+size_frame]
```

```
    fft_spec = np.fft.rfft(a_frame * hamming_window) # 窓関数をかけた後にフーリエ変換

    # 複素スペクトログラムを対数振幅スペクトログラムに
    fft_log_abs_spec = np.log(np.abs(fft_spec))

    # 対数振幅スペクトルをケプストラム係数に(フーリエ変換により抽出)
    fft_log_abs_ceps = np.fft.fft(fft_log_abs_spec)

    # ケプストラム係数をリストに追加
    # dimension は抽出するケプストラム係数の次元(今回は13)
    cepstrogram.append(np.real(fft_log_abs_ceps[0:dimension]))

  return cepstrogram
```

この関数は与えられた音響データからケプストラム係数を計算する関数である．2048 に設定したフレームサイズと 0.01 秒ごとのシフトサイズによってフレームごとに短時間フーリエ変換を行っている．その後対数振幅スペクトルに変換し，そのスペクトルをとる (再びフーリエ変換を行う) ことでケプストラムを抽出している．そして最終的に低次 (ここでは 13 次) のケプストラム係数の実部をリストに追加して返すようになっている．この関数は最尤推定を行う関数 log_likelihood 内で使用されている．

2.2.9 関数 log_likelihood

以下に関数 log_likelihood の実装内容を示す．

```
# 各母音の対数尤度を計算する関数
# x_vowelには各母音に対応する学習データが格納
def log_likelihood(x_vowel, a):
  # 各母音に対応するケプストラム係数、その平均、その共分散(と対数をとったもの)を格納するlist
  ceps_vowel = []
  mean_vowel = []
  cov_vowel = []
  log_cov_vowel = []

  # 計算して格納
  for i in range(6):
    # 各母音のケプストラム係数
    ceps_vowel.append(cepstrum(x_vowel[i]))
    # 各母音のケプストラム係数の平均
    mean_vowel.append(np.mean(np.array(ceps_vowel[i]), axis = 0))
    # 各母音のケプストラム係数の共分散
    cov_vowel.append(np.var(np.array(ceps_vowel[i]), axis = 0))
    # 各母音のケプストラム係数の共分散の対数をとる
    log_cov_vowel.append(np.log(cov_vowel[i]) // 2)

  # 対数尤度のリスト
  L = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

  # 「無音』の対数尤度と「あ」〜「お」の対数尤度を計算
  # 確率分布は正規分布を仮定
  for d in range(dimension):
    for i in range(6):
      L[i] -= log_cov_vowel[i][d]
      L[i] -= ((a[d]-mean_vowel[i][d]) ** 2) / (2 * cov_vowel[i][d])

  # 対数尤度が最大となるインデックスを返す(0:x 1:a 2:i 3:u 4:e 5:o)
```

```
  return L.index(max(L))
```

この関数は正規分布を仮定したうえで学習データから各母音のケプストラム係数の確率密度関数を推定する．具体的にはここで各母音に対応するケプストラム係数の平均 μ と共分散 $\sum$ を算出している．その上で推定対象のデータ X に対し，ケプストラム集合の対数尤度 $L = log_p(X|\mu, \sum)$ の計算を行っている．最後に対数尤度が最大となるインデックスを返すようになっており，これが各母音に対応している．補足になるが，ここでインデックス 0 に対応するのは無音と推定したものである．確率分布として正規分布を仮定しているため対数尤度 L は $L = -\sum_{n=1}^{N}\sum_{d=1}^{D}(log\ \sigma_d\ +\ \frac{(x_{n,d}\ -\ \mu_d)^2}{2\sigma_d^2})$ で計算できる．

2.2.10 関数 vowel_smoothing

以下に関数 vowel_smoothing の実装内容を示す．

```
# 母音推定のスムージングを行う関数
def vowel_smoothing(a):
  # XYXならXXXに変換
  for i in range(len(a)-2):
    if a[i]!=a[i+1] and a[i+1]!=a[i+2]:
      a[i+1]=a[i]

  # XYYXならXXXXに変換
  for i in range(len(a)-3):
    if a[i]!=a[i+1] and a[i+1]==a[i+2] and a[i+2]!=a[i+3]:
      a[i+1]=a[i]
      a[i+2]=a[i]

  # XYYYXならXXXXXに変換
  for i in range(len(a)-4):
    if a[i]!=a[i+1] and a[i+1]==a[i+2] and a[i+2]==a[i+3] and a[i+3]!=a[i+4]:
      a[i+1]=a[i]
      a[i+2]=a[i]
      a[i+3]=a[i]

  return a
```

この関数は母音推定結果の平滑化を行うための関数である．母音の推定においては推定値として同じものが連続するはずという前提に基づいてスムージングを行っている．これにより，途中推定がうまく行われていない箇所があってもその結果をある程度正しく推測できるようにしている．

2.2.11 関数 f_zero_smoothing

以下に関数 f_zero_smoothing の実装内容を示す．

```
# 基本周波数のスムージングを行う関数
def f_zero_smoothing(a):
  global vowel_presumption

  # 無音なら値を0に変換
  for i in range(len(a)):
    if vowel_presumption[i] == 0:
      a[i] = 0

  return a
```

この関数は基本周波数の推定結果の平滑化を行うための関数である．無音区間の基本周波数は無意味であるという前提に基づいてスムージングを行っている．具体的には，母音推定において無音であると推定されていた箇所での基本周波数を 0 とすることでグラフがより見やすいものになるよう工夫している．

2.3 グラフィカルユーザーインターフェースの実装

2.3.1 使用したライブラリ

グラフィカルユーザーインターフェースの構築には Tkinter を，グラフの描画には matplotlib を使用した．

2.3.2 フレーム 1

フレーム 1 はスペクトログラムの表示とフレーム 2 のラジオボタンで選択されたグラフが描画されるフレームである．加えて，再生ボタンを作成し押された場合に音響データが再生されるようにしている．

```
# 再生ボタンの描画
bt_pb = tkinter.Button(frame1, text="再生", font=("Times New Roman", 20), background="white",
command = play_back)
bt_pb.pack(side=TOP)

# スペクトログラムの描画
fig, ax1 = plt.subplots()
# 2つ目の縦軸を作成
ax2 = ax1.twinx()
# masterに対象とするframeを指定
canvas = FigureCanvasTkAgg(fig, master=frame1)
# ラベル設定
ax1.set_xlabel('sec')
ax1.set_ylabel('frequency [Hz]')
# 表示の設定
ax1.imshow(
        np.flipud(np.array(spectrogram).T),
        extent=[0, duration, 0, 1000],
        aspect='auto',
        interpolation='nearest'
)
# 最後にFrameに追加する処理
canvas.get_tk_widget().pack(side="left")
```

上に示したのはフレーム 1 のグラフィカルユーザーインターフェースに関するプログラムである．ボタンウィジェットの生成とスペクトログラムを描画するキャンバスを追加している．スペクトログラムの縦軸は 0 から 1000[Hz] までとなっており拡大表示するように設定した．

```
# 再生ボタンが押された際に呼び出されるコールバック関数
def play_back():
  cmd = "sox aiueo.wav -d"
  p = subprocess.Popen(cmd, shell=True)
```

次に示したのは再生ボタンが押されたときに呼び出されるコールバック関数であり，これが押されると sox の再生コマンドが呼び出され音響データの再生が行われるようになっている．

2.3.3 フレーム 2

フレーム 2 はラジオボタンによって基本周波数，音量，母音推定の結果のいずれかをスペクトログラム上に描画させるフレームである．

```
# ラベルの生成
label = tkinter.Label(frame2, text="表示するグラフの選択", font=("Times New Roman", 24),
anchor="w", background="white")

# ラジオボタンで使用する値の変数を生成
radiovalue = tkinter.IntVar()

# ラジオボタン「基本周波数」の描画
bt_f0 = tkinter.Radiobutton(frame2, variable=radiovalue, value=1, text="基本周波数",
font=("Times New Roman", 20), width=50, anchor="w", background="white", command = draw_frequency)
# ラジオボタン「音量」の描画
bt_power = tkinter.Radiobutton(frame2, variable=radiovalue, value=2, text="音量",
font=("Times New Roman", 20), width = 50, anchor="w", background="white", command = draw_volume)
# ラジオボタン「母音推定」の描画
bt_vowel = tkinter.Radiobutton(frame2, variable=radiovalue, value=3, text="母音推定",
font=("Times New Roman", 20), width = 50, anchor="w", background="white", command = draw_vowel)

label.pack(pady = 10)
bt_f0.pack()
bt_power.pack()
bt_vowel.pack()
```

上に示したのはフレーム 2 のグラフィカルユーザーインターフェースに関するプログラムである．ラベルとラジオボタンを生成し，描画するグラフを選択できるようにした．

```
# ラジオボタン「基本周波数」が押された際に呼び出されるコールバック関数
def draw_frequency():
  ax2.cla()
  ax2.set_ylabel('frequency [Hz]')
  x_data = np.linspace(0, duration, len(frequency_zero))
  ax2.plot(x_data, frequency_zero, c = '#ff7f00')
  ax2.set_ylim([0, 1000])
  canvas.draw()

# ラジオボタン「音量」が押された際に呼び出されるコールバック関数
def draw_volume():
  ax2.cla()
  ax2.set_ylabel('volume [db]')
  x_data = np.linspace(0, duration, len(volume))
  ax2.plot(x_data, volume, c = '#ff7f00')
  canvas.draw()

# ラジオボタン「母音推定」が押された際に呼び出されるコールバック関数
def draw_vowel():
  ax2.cla()
  ax2.set_ylabel('vowel')
  x_data = np.linspace(0, duration, len(vowel_presumption))
  ax2.plot(x_data, vowel_presumption, c = '#ff7f00')
  ax2.set_yticks([0, 1, 2, 3, 4, 5])
  ax2.set_yticklabels(['x', 'a', 'i', 'u', 'e', 'o'])
  canvas.draw()
```

次に示したのは各ボタンが押されたときに呼び出されるコールバック関数であり，押されたボタンの種類に応じて縦軸のラベルと描画されるグラフが変わるようになっている．具体的には，ax2.cla() によって元のグラフを消したのちに新しいグラフのプロットを行っている．vowel_presumption では推定された母音に対応するインデックスが格納されているため，set_yticks と set_yticklabels によってラベルを設定している．繰り返しになるが，x はその推定結果が無音であることを示している．

2.3.4 フレーム 3

フレーム 3 はスライドバーによる特定の時間の選択とその時間の基本周波数，音量，母音推定の結果の表示，その時間のスペクトルの描画が行われるフレームである．

```
# スペクトルを表示する領域を確保
# ax2, canvs2 を使って上記のコールバック関数でグラフを描画する
fig2 = plt.figure(figsize=(5, 5), dpi = 60)
ax3 = fig2.add_subplot(111)
ax3.set_ylabel('amblitude')
ax3.set_xlabel('frequency [Hz]')
ax3.set_title("Spectrum")
canvas2 = FigureCanvasTkAgg(fig2, master=frame3)

# スライドバーで選択された時間の基本周波数、音量、母音推定の結果を表示するラベルを作成
label2 = tkinter.Label(frame3, width=100, text=u"基本周波数:   Hz   音量:   db   母音推定:
" , font=("Times New Roman", 20), background="white")

# スライドバーで選択された時間に対応する配列のインデックスを格納する変数
selected_time = 0
# スライドバーを作成
scale = tkinter.Scale(
        command=_draw_graph,
        # ここにコールバック関数を指定
        master=frame3,
        # 表示するフレーム
    background="white",
        from_=0,
        # 最小値
        to=duration,
        # 最大値
        resolution=size_shift/SR,
        # 刻み幅
        label=u'スペクトルを表示する時間[sec]',
        orient=tkinter.HORIZONTAL,
        # 横方向にスライド
        length=600,
        # 横サイズ
        width=30,
        # 縦サイズ
        font=("", 20)
        # フォントサイズは20pxに設定
)
scale.pack(side="top")

label2.pack(side=TOP, pady=8)

# "top"は上部方向にウィジェットを積むことを意味する
canvas2.get_tk_widget().pack(side="right")
```

上に示したのはフレーム 3 のグラフィカルユーザーインターフェースに関するプログラムである．ラベルとスライドバーを生成し，時間の選択とその時間に対応する各推定値の表示，スペクトルの描画を行えるようにした．

```
# スライドバーの値が変更されたときに呼び出されるコールバック関数
# vはスライドバーの値
def _draw_graph(v):
  # スライドバーの値が変更されたらselected_timeの値を更新
  update_selected_time(v)
  # スライドバーの値が変更されたらラベルの内容を更新
  label2["text"] = "基本周波数: %s Hz  音量: %s db  母音推定: %s"
  % (int(frequency_zero[selected_time]), int(volume[selected_time]),

  vowel[vowel_presumption[selected_time]])
  # スライドバーの値からスペクトルのインデクスおよびそのスペクトルを取得
  index = int((len(spectrogram)-1) * (float(v) / duration))

  # 直前のスペクトル描画を削除し，新たなスペクトルを描画
  ax3.cla()
  ax3.set_ylabel('amblitude')
  ax3.set_xlabel('frequency [Hz]')
  x_data = np.linspace(0, SR/2, len(spectrogram[index]))
  ax3.plot(x_data, spectrogram[index])
  ax3.set_ylim(-10, 5)
  ax3.set_xlim(0, 1000)
  ax3.set_title("Spectrum")
  canvas2.draw()
```

次に示したのはスライドバーの値が更新されたときに呼び出されるコールバック関数である．スライドバーの値に応じてラベルの内容の変更とスペクトルの再描画を行うようになっており，ある瞬間の時間の情報を取り出せるようになっている．また，ここでは表示するスペクトルは 1000Hz までとなっている．

3 振り返り

3.1 工夫した点

グラフィカルユーザーインターフェースの作成にあたって特に工夫した点はユーザー視点の使いやすさを重視したことである．シンプルなフレーム構成と視覚的にわかりやすいウィジェットの配置によりユーザーが一目で各機能の意味を理解できるようにしている．また音響データの基本的な情報が伝わりやすいスペクトログラムをベースに、その他の情報をグラフ化して重ねて表示できるようにした．その際，全てのグラフをまとめて表示するのではなくラジオボタンによって 1 種類ずつ切り替えることができるようにし，それに応じて右側の縦軸ラベルも変化させることで煩雑な見た目にならないよう注意している．そしてスライドバーによる特定の時間選択を可能にしたことで，スペクトルの可視化と各瞬間における基本周波数や音量，母音推定結果の取得を実現した．このように音響データの大域的な変化と局所的な値の観測を同時に行えるようにした上で，音響データの再生ボタンも追加し更なる利便性の向上を目指した．加えてデータ処理における工夫は母音推定において無音に対応した学習モデルを用意したことである．これにより適切な推定結果が得られるようにし，不自然なグラフとならないよう改良を図った．また，表示する周波数領域を 1000Hz までに限定したり各グラフのスムージングを行ったりしたことでユーザーにとって見やすいグラフが表示されるようにした．

3.2 改善点

当初の目標としていた音響データの再生を実装することはできたが，停止機能や再生時間の表示を行うまでには至らなかった．スペクトログラムの横軸で時間をとっているため，現在どの部分が再生されているのかをスペクトログラム上に表示できればもっとわかりやすいグラフィカルユーザーインターフェースになったと思う．また，対象となる音響ファイルの選択ができるようにしたり，グラフの概観 (表示する範囲や線の色・太さなど) をユーザー側が変更できるようにしたりとユーザーが使いやすくなる工夫をさらに導入しても良いと思う．