音響信号処理　課題 2 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 28 日

1　実施内容

1.1　課題

WAV (MP3) ファイルに保存された楽曲の再生とマイクからの歌声の録音を同時に行いながら，楽曲のスペクトログラムと音声の音高（基本周波数）を同時に図示するカラオケシステムを制作する．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

楽曲のスペクトログラムと音声の音高，音量は常に表示されるようにし，その図に重なるように楽曲のガイドメロディの表示を行う．同時に再生時間や歌詞，入力音声の最低音・最高音も表示しカラオケシステムとしての機能の充実を目指す．余裕があれば，独自の採点機能の実装やスペクトル・スペクトル包絡の表示も行う．

1.3　実装結果

今回は米津玄師さんの「Lemon」を楽曲として選択し，実装を進めた．作成した簡易カラオケシステムの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．

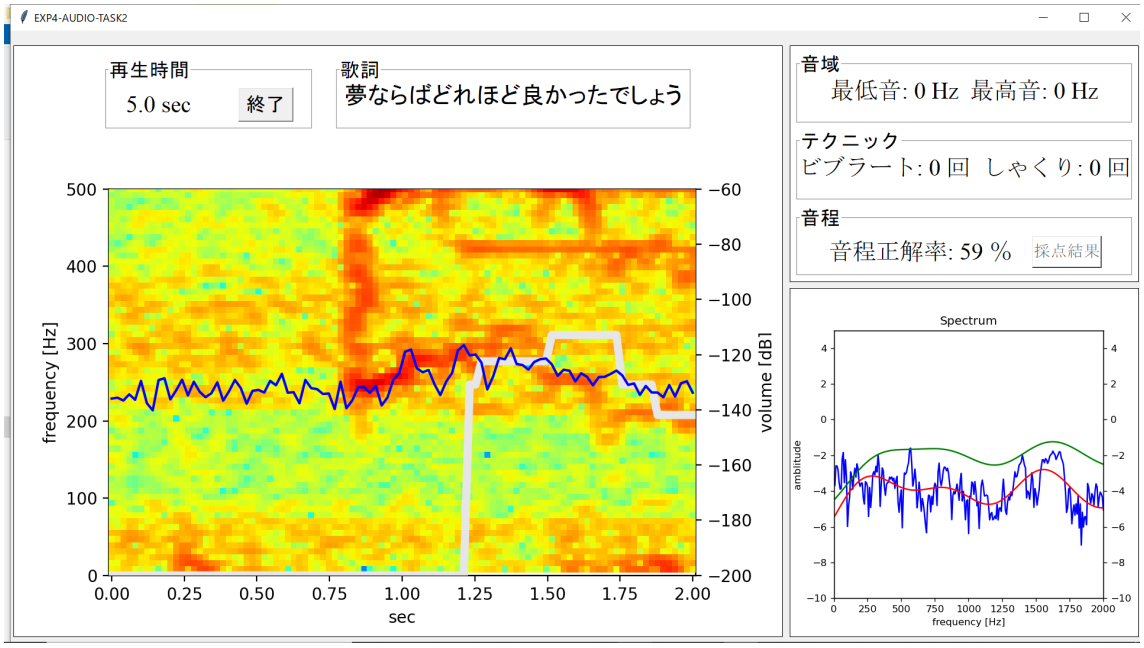

図1　作成したグラフィカルユーザーインターフェース

2　実装内容

2.1　全体の構成

作成した簡易カラオケシステムの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 2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．なお，図中の数字の単位は px となっている．以降，カラオケシステムの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．

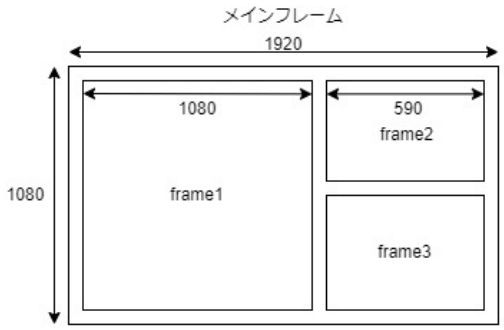

図2　簡易カラオケシステムの GUI のフレーム構成

2.2　カラオケシステムの処理部分の実装

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．マイクからの入力音声の処理と wav ファイル音楽再生の処理にあたっては pyaudio と pydub を用いた．

2.2.2　グローバル変数

以下の表 1に使用した主なグローバル変数の一覧を示す．なお，データを格納するためのリストなどは省略している．

表1　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
SAMPLING_RATE	int	16000	サンプリングレート
FRAME_SIZE	int	2048	フレームサイズ
SIZE_SHIFT	int	320	シフトサイズ
SPECTRUM_MIN	int	-5	スペクトルの最小値
SPECTRUM_MAX	int	1	スペクトルの最大値
VOLUME_MIN	int	-200	音量を表示する際の最小値
VOLUME_MAX	int	-60	音量を表示する際の最大値
EPSILON	float	1e-10	対数の真数が 0 にならないように加える微小値
hamming_window	numpy.ndarray	-	フレームサイズに合わせたハミング窓
MAX_NUM_SPECTROGRAM	int	1024	グラフに表示する縦軸方向のデータ数
NUM_DATA_SHOWN	int	100	グラフに表示する横軸方向のデータ数
SCALE_UP_RATE	int	16	周波数表示の拡大比率 (500Hz まで表示)
is_gui_running	boolean	False	GUI の開始フラグ
is_music_running	boolean	False	楽曲の開始フラグ
is_viv_state	boolean	False	直前フレームでのビブラートの有無のフラグ
is_shakuri_state	boolean	False	直前フレームでのしゃくりの有無のフラグ
vibrato_count	int	0	ビブラートの回数
shakuri_count	int	0	しゃくりの回数
interval_correct_rate	float	0	音程正解率
interval_correct_sum	float	0	各フレームの音程正解率の総和
frequency_zero_min	float	1e-10	音声の最低音高 (周波数)
frequency_zero_max	float	1e-10	音声の最高音高 (周波数)
interval_score	int	0	音程の得点
technique_score	int	0	技術の得点
final_score	int	0	最終スコア
x_stacked_data	numpy.ndarray	[]	入力音声データ
idx_guide	int	0	表示するガイドメロディのインデックス
is_guide_start	boolean	False	ガイド表示の開始フラグ
x_stacked_data_music	numpy.ndarray	[]	再生音源データ
now_playing_sec	float	0.0	楽曲の再生時間

2.2.3　関数

以下の表 2に作成・使用した主な関数の一覧を示す.

表2　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
is_viv	boolean	ビブラートの有無を判定
is_shakuri	int	しゃくりの有無を判定
nn2hz	float	ノートナンバーから周波数への変換
hz2nn	int	周波数からノートナンバーへの変換
smoothing	list	基本周波数グラフのスムージング
calc_similarity	int	音程の正確性を計算
calc_score	int	採点結果の算出を行う関数
animate	-	グラフの再描画のために呼び出される関数
add_to_guide	-	ガイドメロディの各音階の配列を作成し連結する関数
_quit	-	GUI を終了する関数
show_score	-	最終的な採点結果を計算
draw_graph	-	スペクトルとスペクトル包絡を描画
input_callback	-	入力音声処理のフレーム毎に呼び出される関数
play_music	-	音楽ファイルから読み込むたびに呼び出される関数
update_gui_text	-	GUI 上に表示されるテキスト内容を更新

2.2.4　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．

2.2.5　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．

2.2.6 関数 is_viv

以下に関数 is_viv の実装内容を示す．

```
# 音高の最大値と最小値の差(レンジ)を計算
dif = np.amax(a) - np.amin(a)
# 音高の平均値を計算
f_mean = np.mean(a, dtype=float)
# 音高の平均値のラインと交差する回数を保存
mc = 0

# 配列内で連続する2つの値と音高の平均値との大小関係が異なれば交差しているとし，
# mcの値を 1 増やす
for i in range(len(a) - 1):
  if(
    (a[i] > f_mean and a[i+1] < f_mean) or
    (a[i] < f_mean and a[i+1] > f_mean)
  ):
    mc += 1

# 音高のレンジが閾値以下かつmcの値が閾値以上であればビブラートであると判定
if dif <= 50 and mc >= 5:
```

```
    # ただしビブラートの連続したカウントを防ぐためis_viv_stateを用いて直前のフレームで
    # ビブラートが検出されていなければTrueを返す
    if is_viv_state == False :
      is_viv_state = True
      return True
    else :
      return False
  # ビブラートでなければis_viv_stateの値を1にしてFalseを返す
  else :
    is_viv_state = False
    return False
```

この関数は周波数データの配列を引数にとり，その区間内のビブラートの有無を判定する．その際，以下の条件をすべて満たしている場合にビブラートが含まれていると判定し，True を返すように実装した．

(1) 音高のレンジ (最高周波数と最低周波数の差) が 50Hz 以下

(2) 音高の平均値のラインとの交差数が 5 回以上

(3) 直前のフレームでビブラートが検出されていない

ここではビブラートが見かけの音高を保ちながら高さを揺らすものであるという定義にしたがって (1) と (2) の条件を定めた．また同一区間内のビブラートの重複カウントを防ぐために is_viv_state というフラグを用いて (3) の条件も満たすことを確認している．図 3にはビブラート検出のイメージ図を示す．

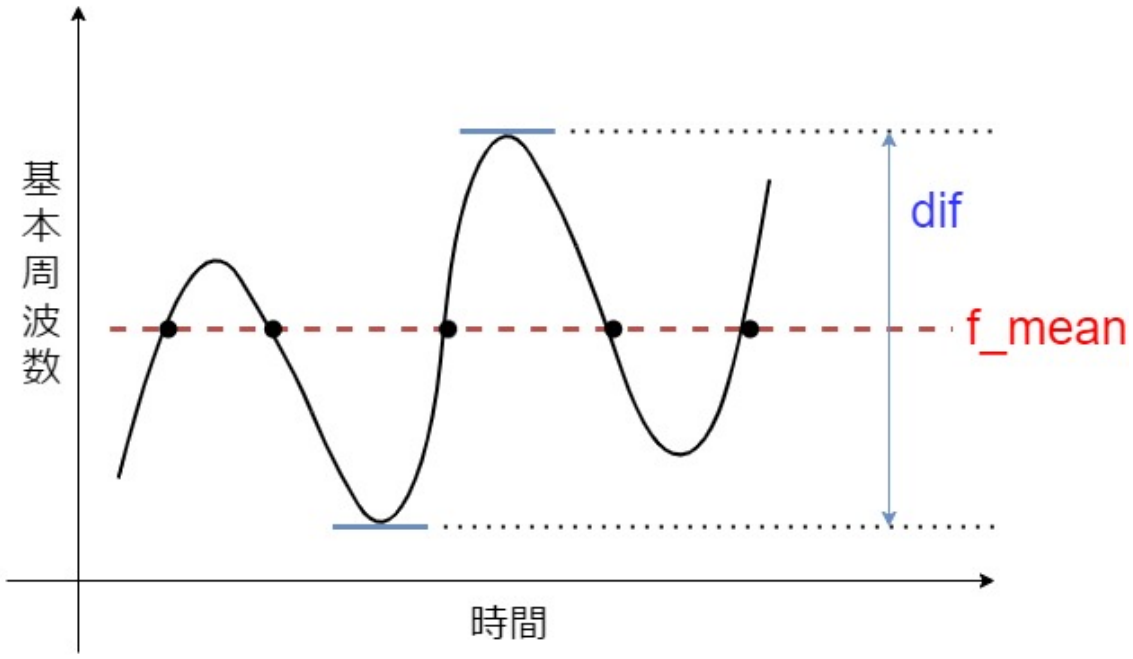

図3　ビブラート検出のイメージ

2.2.7　関数 is_shakuri

以下に関数 is_shakuri の実装内容を示す．

```
# フレーズの始まりであるかの判定
```

```
  if np.all(a[0:4]) or np.any(a[5:] == 0):
    is_shakuri_state = False
    return False
  # 音高の最小値と最大値，そのノートナンバーの差(レンジ)を計算
  f_min = np.amin(a[5:])
  f_max = np.amax(a[5:])
  dif = hz2nn(f_max) - hz2nn(f_min)

  # フレーズの最初の音高が最小値と等しいかを判定
  # 配列の最後の音高が最大値と等しいかを判定
  if f_max != a[-1] or f_min != a[5]:
    is_shakuri_state = False
    return False

  # ノートナンバーにして2以上4以下の音高の上昇がなければFalse
  if dif < 2 or dif > 4:
    is_shakuri_state = False
    return False
```

この関数は周波数データの配列を引数にとり，その区間内のしゃくりの有無を判定する．その際，以下の条件をすべて満たしている場合にしゃくりが含まれていると判定し，True を返すように実装した．

(1) 配列の始まりから 0.1 秒以上の無音区間が存在 (= フレーズ開始部分である)

(2) 無音区間を除いた最初の音高が最小値かつ配列の最後の音高が最大値

(3) 無音区間を除いた音高のレンジ (最高周波数と最低周波数の差) がノートナンバーにして 2 以上 4 以下

(4) 直前のフレームでしゃくりが検出されていない

ここではしゃくりがフレーズの始まりにおいて少し下の音程から本来の音程へ滑らかに上げる歌唱法であるという定義にしたがって (1) から (3) の条件を定めた．また同一区間内のしゃくりの重複カウントを防ぐために is_shakuri_state というフラグを用いて (4) の条件も満たすことを確認している．図 4にはしゃくり検出イメージ図を示す．

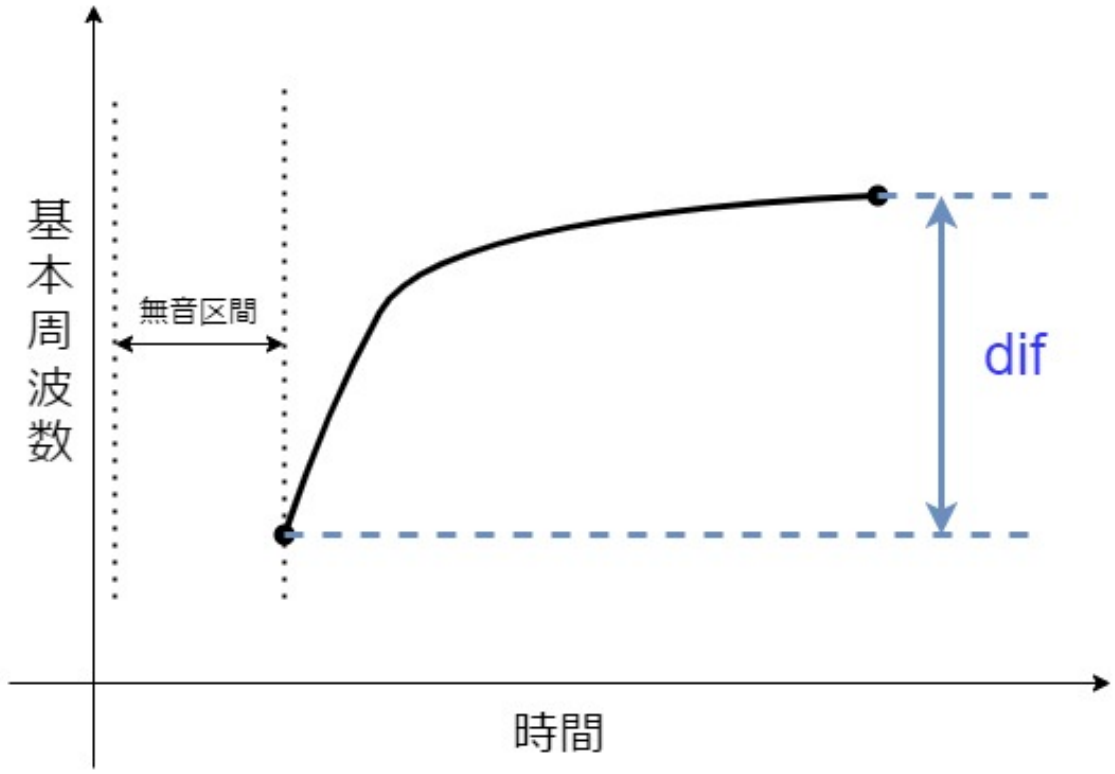

図4　しゃくり検出のイメージ

2.2.8　関数 nn2hz

以下に関数 nn2hz の実装内容を示す.

```
# ノートナンバーがある範囲に入っていなければ0を返す
if notenum <= 40 or notenum >= 75:
  return 0.0
# ノートナンバーを対応する周波数に変換して返す
else :
  return 440.0 * (2.0 ** ((notenum - 69) / 12.0))
```

この関数はノートナンバー nn を受け取ってそれに対応した基本周波数 f を返す関数である．$f = \frac{440 \times 2^{nn-69}}{12}$ という式にしたがって変換を行っている．ただし，ノートナンバーが想定している範囲 (ここでは 40 以上 75 以下) 外であれば 0 を返すようにした.

2.2.9　関数 hz2nn

以下に関数 hz2nn の実装内容を示す.

```
return int (round (12.0 * ((math.log((frequency + 0.001) / 440.0)) / math.log(2.0)))) + 69
```

この関数は周波数 f を受け取ってそれに対応したノートナンバー nn を返す関数である．$\mathrm{nn} = 12log_2(\frac{f}{440})$ という式にしたがって変換している．また真数が 0 になるのを防ぐため f に微小値 EPSILON を加えている.

2.2.10 関数 smoothing

以下に関数 smoothing の実装内容を示す.

```
# XYXならXXXに変換
for i in range(len(a)-2):
  if a[i+1] > 0 and 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]

# a[i]が0または500より大きければ欠損値に変換
for i in range(len(a)):
  if a[i] == 0 or a[i] > 500:
    a[i]=np.nan

return a
```

この関数はグラフの平滑化を行うための関数である．基本周波数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．また想定している範囲 (0 より大きく 500 以下) 外の値であれば欠損値に変換してグラフに表示しないようにすることで，グラフがより見やすいものになるよう工夫している．

2.2.11 関数 calc_similarity

以下に関数 calc_similarity の実装内容を示す.

```
# ガイドメロディの音程が不定の部分は現在の正解率をそのまま返す
if y == 0:
  return interval_correct_rate
# 入力音声の音高が明らかに異常であれば現在の正解率をそのまま返す
if x < 50 or x > 500:
  return interval_correct_rate
# ガイドメロディの音程のノートナンバーと入力音声の音高のノートナンバーを比較
# 絶対値が1違う毎に10%ずつ正解率を引いた値を返す
else:
  x_nn = hz2nn(x)
  y_nn = hz2nn(y)
  res = max(0, 100 - abs(x_nn-y_nn)*10)
  return res
```

この関数は音声の周波数を受け取り，ガイドメロディと比較してその瞬間の音程正解率を計算する関数である．基本的には音声の音高とガイドメロディの周波数をそれぞれノートナンバーに変換し，それらの差に応じ

て正解率を計算している．具体的にはノートナンバーの差の絶対値が 1 異なるごとに 100% から 10% ずつ減らすようにしている．また，ガイドメロディが不定の場合や入力音声が想定範囲 (ここでは 50Hz 以上 500Hz 以下) 外であれば現在の音程正解率をそのまま返すようにしている．

2.2.12 関数 calc_score

以下に関数 calc_score の実装内容を示す．

```
# 音程正解率を0.9倍した値を音程の得点とする(90点満点)
interval_score = int(interval_correct_rate * 0.9)

# 5秒に1回ビブラートあるいはしゃくりがあれば満点となる基準で技術の得点を設定
# ただし，音程正解率を乗算しており音程の正確さも考慮に入れるようにした
technique_score = min(10, int(interval_correct_rate*(shakuri_count+vibrato_count)/
(2*now_playing_sec)))

# 音程の得点と技術の得点の合計が最終スコア
final_score =  min(100, interval_score+technique_score)
```

この関数は最終的なスコアを計算する関するである．音程の得点 (90 点満点) と技術点 (10 点満点) によって簡易的な採点システムを作成した．音程の得点は音程正解率を 0.9 倍することで算出し，技術点はビブラートとしゃくりの回数，音程正解率を総合的に加味して算出している．そしてそれらの合計点を最終スコアとして算出するようにした．

2.2.13 関数 add_to_guide

以下に関数 add_to_guide の実装内容を示す．

```
# 第1引数のノートナンバーを周波数に変換し第2引数の数だけガイドメロディとして加える
new = np.full(count, nn2hz(num))
if num == 0 : new = np.full(count, 0)
lemon_data = np.concatenate([lemon_data, new])
```

この関数は音階を打ち込んでガイドメロディを作成する関数である．音階のノートナンバーをそのフレーム数を引数にとり，それを連結していくことで楽曲のガイドメロディを作成している．

2.2.14 関数 input_callback

以下に関数 input_callback の実装内容を示す．

```
# 現在のフレームの音声データをnumpy arrayに変換
x_current_frame = np.frombuffer(in_data, dtype=np.float32)

# 現在のフレームとこれまでに入力されたフレームを連結
x_stacked_data = np.concatenate([x_stacked_data, x_current_frame])

# 抽出するケプストラム係数の次元
dimension = 13


# フレームサイズ分のデータがあれば処理を行う
if len(x_stacked_data) >= FRAME_SIZE:

  # フレームサイズからはみ出した過去のデータは捨てる
```

```
x_stacked_data = x_stacked_data[len(x_stacked_data)-FRAME_SIZE:]

# スペクトルを計算
fft_spec = np.fft.rfft(x_stacked_data * hamming_window)
fft_log_abs_spec = np.log10(np.abs(fft_spec) + EPSILON)[:-1]

# スペクトル包絡を抽出
spectrum_data = np.log(np.abs(fft_spec) + EPSILON)[:-1]
fft_log_abs_ceps = np.fft.fft(spectrum_data)
fft_log_abs_ceps[dimension:len(fft_log_abs_ceps)-dimension] = 0
spectrum_envelope = (np.fft.ifft(fft_log_abs_ceps)).real


# 2次元配列上で列方向（時間軸方向）に1つずらし（戻し）
# 最後の列（＝最後の時刻のスペクトルがあった位置）に最新のスペクトルデータを挿入
spectrogram_data = np.roll(spectrogram_data, -1, axis=1)
spectrogram_data[:, -1] = fft_log_abs_spec[0:(FRAME_SIZE//2)//SCALE_UP_RATE]

# 音量も同様の処理
vol = 20 * np.log10(np.mean(x_current_frame ** 2) + EPSILON)
volume_data = np.roll(volume_data, -1)
volume_data[-1] = vol

# 基本周波数も同様の処理
# x_f0は自己相関によって推定された基本周波数，x_f0_for_graphはそれを正規化したもの
x_f0 = correlate(x_stacked_data)
nn = hz2nn(x_f0)
x_f0_for_graph = nn2hz(nn)
frequency_zero_data = np.roll(frequency_zero_data, -1)
frequency_zero_data_for_graph = np.roll(frequency_zero_data_for_graph, -1)

# 音量がある閾値より小さければ無音であるとみなす
if vol < -80 :
  frequency_zero_data[-1] = 0
  frequency_zero_data_for_graph[-1] = 0
# 発声があった際に行う処理
else :
  frequency_zero_data[-1] = x_f0
  frequency_zero_data_for_graph[-1] = x_f0_for_graph

  # 最低音高と最高音高を更新
  # 音高がある範囲に入っていれば一番最初は無条件でその値にし，それ以降は比較により
  更新するかどうかを決定
  if frequency_zero_min == 1e-10 and x_f0_for_graph > 50 and x_f0_for_graph < 500:
    frequency_zero_min = x_f0_for_graph
    frequency_zero_max = x_f0_for_graph
  elif x_f0_for_graph > 50 and x_f0_for_graph < 500:
    frequency_zero_min = min(frequency_zero_min, x_f0_for_graph)
    frequency_zero_max = max(frequency_zero_max, x_f0_for_graph)

  # ビブラートの判定
  # 直近0.5秒間の入力音声に対して判定
  if is_viv(frequency_zero_data[75:]) :
    vibrato_count += 1

  # しゃくりの判定
  # 直近0.5秒間の入力音声に対して判定
```

```
      if is_shakuri(frequency_zero_data[75:]):
        shakuri_count += 1

    # 再生時間が3.95秒になったらガイドメロディを開始
    if now_playing_sec >= 3.95 : is_guide_start = True

    # ガイドメロディが始まっていれば処理を行う
    if is_guide_start:
    # 配列上で 1 つずらし（戻し）
    # 最後の列（= 最後の時刻のスペクトルがあった位置）に最新のガイドメロディデータを挿入
      guide_melody = np.roll(guide_melody, -1)
      guide_melody[-1] = lemon_data[idx_guide]
      # ガイドメロディの値が0であればグラフには表示しないようにする
      if guide_melody[-1] == 0 : guide_melody[-1] = np.nan
      idx_guide += 1
      # ガイドメロディと音声の音高から音程正解率を算出
      # interval_correct_sumに得られた値を加える
      interval_correct_sum += calc_similarity(x_f0, lemon_data[idx_guide])
      # interval_correct_sumをidx_guideで割り，音程正解率の平均値を算出
      interval_correct_rate = int(interval_correct_sum/idx_guide)

  # グラフのスムージング処理
  frequency_zero_data_for_graph = smoothing(frequency_zero_data_for_graph)

  # 戻り値は pyaudio の仕様に従うこと
  return None, pyaudio.paContinue
```

この関数はマイク入力の音声処理において，フレーム毎に呼び出される関数である．pyaudio で作成した再生ストリームからシフトサイズの単位で実行されていく．全体としては，入力音声のスペクトル・基本周波数・音量の計算やスペクトル包絡の抽出等を行っている．

手順としては最初に音声データを numpy array に変換した後，そのフレームと既存のフレームを連結する処理を行っている．連結後の配列が設定したフレームサイズ 2048 よりも大きければはみだした過去のデータを捨てた上でフーリエ変換によってスペクトルの計算を行う．そしてその対数をとった値をスペクトログラムとして表示するようにしている．その後，対数振幅スペクトルをさらにフーリエ変換することによってスペクトル包絡を抽出している．なお，ここでは抽出するケプストラム係数の次元は 13 とし，それらの実部をとることで抽出を実現した．また，スペクトログラムの表示は最大値を 500Hz にするために対数振幅スペクトル全体の $\frac{1}{16}$ を使用している．この値はグローバル変数 SCALE_UP_RATE で指定した．

次に音量 vol は $RMS = \sqrt{\frac{1}{N}\sum_{t=0}^{N-1} x_t^2}$ と $Vol_{dB} = 20log_{10}RMS$ の式から計算した．その後，基本周波数 x_f0 を関数 correlate によって求めており，グラフに表示する値は x_f0 に関数 hz2nn と nn2hz を順に適用することで正規化を行っている．そしてデータとしてリストに追加する際は音量が −80db より小さければ無音区間と判定し，一律で 0 を追加するように実装した．逆に −80db 以上であれば x_f0 とそれを正規化した値をそれぞれ追加し，入力音声の最低音と最高音の更新を行うようにした．また，発音区間に限り関数 is_viv と is_shakuri によってビブラートとしゃくりの判定を行い，それらのカウントを実行した．

最後にガイドメロディの処理もこの関数で行っている．具体的には，楽曲の最初のフレーズが始まる 3.95 秒後から表示を開始するようにし，そこからはフレーム毎に新しいガイドを追加するようにした．グラフに表示するリストへの追加を行うガイドのインデックスは変数 idx_guide で管理している．さらにそのガイドの音階と既に求めた基本周波数 x_f0 によって音程正解率を算出している．関数 calc_similarity によってそのフレーム

の音程正解率を計算した上で，そのフレームまでの音程正解率の総和を格納した変数 interval_correct_sum を idx_guide で割った平均値を音程正解率として表示するようにしている．

2.2.15 関数 play_music

以下に関数 play_music の実装内容を示す．

```
# pydubのmake_chunksを用いて音楽ファイルのデータを切り出しながら読み込む
# 第二引数には何ミリ秒毎に読み込むかを指定
# ここでは20ミリ秒ごとに読み込む
size_frame_music = 20 # 20ミリ秒毎に読み込む

idx = 0

# make_chunks関数を使用して一定のフレーム毎に音楽ファイルを読み込む
for chunk in make_chunks(audio_data, size_frame_music):

  # GUIが終了してれば，この関数の処理も終了する
  if is_gui_running == False:
    break

  # pyaudioの再生ストリームに切り出した音楽データを流し込む
  # 再生が完了するまで処理はここでブロックされる
  stream_play.write(chunk._data)

  # 現在の再生位置を計算（単位は秒）
  now_playing_sec = (idx * size_frame_music) / 1000.

  idx += 1

  # データの取得
  data_music = np.array(chunk.get_array_of_samples())

  # 正規化
  data_music = data_music / np.iinfo(np.int16).max

  #
  # 以下はマイク入力のときと同様
  #

  # 現在のフレームとこれまでに入力されたフレームを連結
  x_stacked_data_music = np.concatenate([x_stacked_data_music, data_music])

  # 抽出するケプストラム係数の次元
  dimension = 13

  # フレームサイズ分のデータがあれば処理を行う
  if len(x_stacked_data_music) >= FRAME_SIZE:
  # フレームサイズからはみ出した過去のデータは捨てる
    x_stacked_data_music = x_stacked_data_music[len(x_stacked_data_music)-FRAME_SIZE:]

    # 音量も同様の処理
    vol = 20 * np.log10(np.mean(data_music ** 2) + EPSILON)

    #
    # 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_stacked_data_music)
      nn = hz2nn(x_f0)

      # スペクトルを計算
      fft_spec = np.fft.rfft(x_stacked_data_music * hamming_window)
      fft_log_abs_spec = np.log10(np.abs(fft_spec) + EPSILON)[:-1]

      # スペクトル包絡を抽出
      spectrum_data_music = np.log(np.abs(fft_spec) + EPSILON)[:-1]
      fft_log_abs_ceps = np.fft.fft(spectrum_data_music)
      fft_log_abs_ceps[dimension:len(fft_log_abs_ceps)-dimension] = 0
      spectrum_envelope_music = (np.fft.ifft(fft_log_abs_ceps)).real

      # 2次元配列上で列方向（時間軸方向）に1つずらし（戻し）
      # 最後の列（＝最後の時刻のスペクトルがあった位置）に最新のスペクトルデータを挿入
      spectrogram_data_music = np.roll(spectrogram_data_music, -1, axis=1)
      spectrogram_data_music[:, -1] = fft_log_abs_spec[0:(FRAME_SIZE//2)//SCALE_UP_RATE]

  # 楽曲が終了したらフラグをFalseにし，「採点結果を表示」のボタンをアクティブにする
  is_music_running = False
  bt_score["state"] = tkinter.NORMAL
  ani.event_source.stop()
  stream.stop_stream()
```

この関数は pyaudio で作成した再生ストリームによって再生する wav ファイル音楽の処理を行う関数である．ここでは pydub で読み込んだ音楽ファイルを再生する部分のみ関数化しており，別スレッド t_play_music で実行している．今回の実装では，pydub の make_chunks 関数を用いて音楽ファイルのデータを切り出しながら読み込んでおり，読み込む間隔は音声処理のシフトサイズと同じく 20ms とした．音楽ファイルを読み込んだ回数を変数 idx で管理し，これにより現在の再生位置 (時間)now_playing_sec を計算している．また，楽曲の処理にあたって正規化とフレームの連結を行っている．処理内容は関数 play_music 内で述べた音声処理の実装とほぼ同じであるため，ここでは割愛する．異なる点は，楽曲の再生が終了した後にフラグ is_music_running を False にしていることである．

2.2.16 関数 update_gui_text

以下に関数 calc_similarity の実装内容を示す．

```
while True:
    # 歌詞の更新
    if now_playing_sec >= 20.0:
      lyrics_label["text"] = "古びた思い出の埃を払う"
    elif now_playing_sec >= 14.0:
      lyrics_label["text"] = "忘れたものをとりに帰るように"
    elif now_playing_sec >= 8.0:
      lyrics_label["text"] = "未だにあなたのことを夢に見る"
    elif now_playing_sec >=0.0:
      lyrics_label["text"] = "夢ならばどれほど良かったでしょう"

    # GUIが表示されているかつ音楽が再生されていれば再生位置（秒），音域，
    各種歌唱テクニックの回数，音程正解率をテキストとしてGUI上に表示
    if is_gui_running and is_music_running:
      playing_sec_label["text"] = '%.1f sec' % now_playing_sec
      range_label["text"] = "最低音: %s Hz  最高音: %s Hz" %
```

```
        (int(frequency_zero_min), int(frequency_zero_max))
        technique_label["text"] = "ビブラート: %s 回　しゃくり: %s 回" %
        (vibrato_count, shakuri_count)
        interval_label["text"] = "音程正解率: %s %" % int(interval_correct_rate)

    # 0.1秒ごとに更新
    time.sleep(0.1)
```

この関数は GUI 上に表示するテキストの内容を更新する関数である．具体的には，歌詞の更新，再生時間の更新，音域の更新，ビブラート・しゃくりの回数の更新，音程正解率の更新が該当する．0.1 秒ごとに表示を更新するようにしており，歌詞の内容は再生時間に応じて変更するように実装した．

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

図 5の赤枠で示すようにフレーム 1 は再生時間を表示する部分，歌詞を表示する部分，楽曲のスペクトログラムと音声の音高・音量，ガイドメロディの表示を行う部分の 3 つに分かれている．

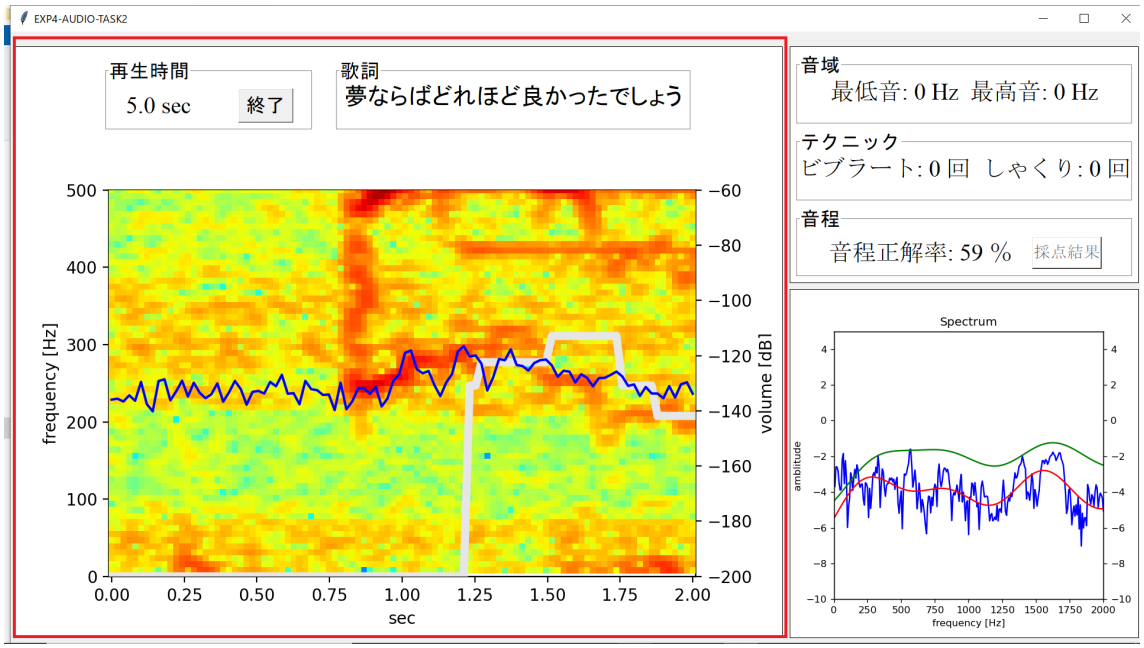

図5　フレーム 1 の内容

まずは，以下に再生時間と歌詞を表示する部分の実装を示す．

```
# 終了ボタンが押されたときに呼び出される関数
# ここではGUIを終了する
def _quit():
        root.quit()
```

```
        root.destroy()

# 再生時間と歌詞を表示する領域のフレームをフレーム1内に作成
frame1_sub = tkinter.Frame(master=frame1, width=1050, height=120, background="white")
frame1_sub.propagate(False)

# 再生時間を表示するラベルフレーム
playing_sec_frame = tkinter.LabelFrame(master=frame1_sub, text="再生時間",
font=("Arial", 25), width=350, height=120, background="white")
playing_sec_frame.propagate(False)
playing_sec_label = tkinter.Label(master=playing_sec_frame, text="0.0 sec",
font=("Times New Roman", 30), background="white")

# 終了ボタンを作成
bt_fin = tkinter.Button(master=playing_sec_frame, text="終了", command=_quit, font=("", 25))

# 歌詞を表示するラベルフレーム
lyrics_frame = tkinter.LabelFrame(master=frame1_sub, text="歌詞",
font=("Arial", 25), width=600, height=120, background="white")
lyrics_frame.propagate(False)
lyrics_label = tkinter.Label(master=lyrics_frame, text="", font=("", 30), background="white")
```

ここでは，終了ボタンを押されたときに呼び出されるコールバック関数として _quit を定義している．これにより，終了ボタンが押されると強制的に GUI が終了するようになっている．また，各種ラベルフレームを作成し再生時間や歌詞を表示する領域を確保した．再生時間の表示は小数第 1 位までに制限している．
次にグラフの描画部分の実装を以下に示す．

```
# スペクトログラムを描画
fig, ax1 = plt.subplots(1, 1)
canvas = FigureCanvasTkAgg(fig, master=frame1)

# pcolormeshを用いてスペクトログラムを描画
# 戻り値はデータの更新 & 再描画のために必要
ax1_sub_1 = ax1.pcolormesh(
        X,
        Y,
        spectrogram_data,
        shading='nearest',      # 描画スタイル
        cmap='jet',                     # カラーマップ
        norm=Normalize(SPECTRUM_MIN, SPECTRUM_MAX)
        # 値の最小値と最大値を指定して，それに色を合わせる
)

# 音量を表示するために反転した軸を作成
ax2 = ax1.twinx()

# ガイドメロディ，音声の音高，音量をプロットする
# 戻り値はデータの更新 & 再描画のために必要
ax1_sub_2, = ax1.plot(time_x_data, guide_melody, c= '0.9', linewidth=5)
ax1_sub_3, = ax1.plot(time_x_data, frequency_zero_data_for_graph, c='y', linewidth=5)
ax2_sub, = ax2.plot(time_x_data, volume_data, c='b')

# ラベルの設定
ax1.set_xlabel('sec')                   # x軸のラベルを設定
ax1.set_ylabel('frequency [Hz]')        # y軸のラベルを設定
ax2.set_ylabel('volume [dB]')           # 反対側のy軸のラベルを設定
```

```
# 音高と音量を表示する際の値の範囲を設定
ax1.set_ylim([0, 500])
ax2.set_ylim([VOLUME_MIN, VOLUME_MAX])
plt.grid(linewidth=0.5)

# maplotlib animationを設定
ani = animation.FuncAnimation(
        fig,
        animate,                    # 再描画のために呼び出される関数
        interval=400,               # 400ミリ秒間隔で再描画を行う
        blit=True                   # blitting処理を行うため描画処理が速くなる
)
```

ここでは周波数を縦軸とする ax1 とそれを反転した軸 ax2 を作成し楽曲のスペクトログラム，音声の音高・音量を同時に表示するようにした．ここで ax1 の縦軸は 0 から 500Hz までとなっておりスペクトログラムの拡大表示を行っている．描画は matplotlib の animation を使用し，リアルタイムで描画処理がされるようになっている．

また，この領域では灰色の太線でガイドメロディの表示も行っている．マイクの入力音声の音高は黄色の太線で表示するようにしているため，ガイドとのずれを視覚的にわかりやすく確認できるようになっている．以下の図 6では音声が入っていないとき (左) と入っているとき (右) で表示がどのように変わるかを示した．図中の青線が音量を表すグラフである．

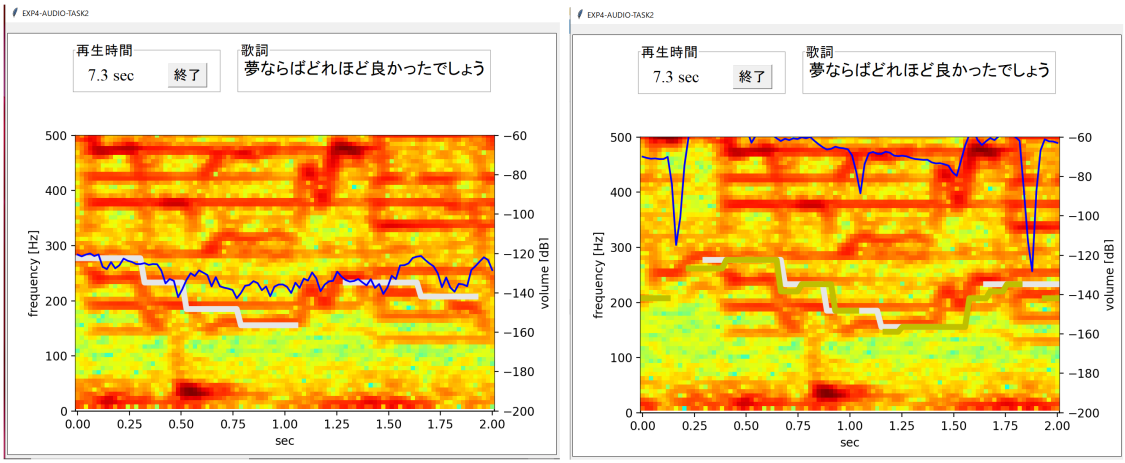

図6　入力音声の有無によるグラフ表示の違い

2.3.3　フレーム 2

図 7の赤枠で示すようにフレーム 2 は音域を表示する部分，歌唱テクニックの回数を表示する部分，音程正解率と採点結果の表示を行う部分の 3 つに分かれている．

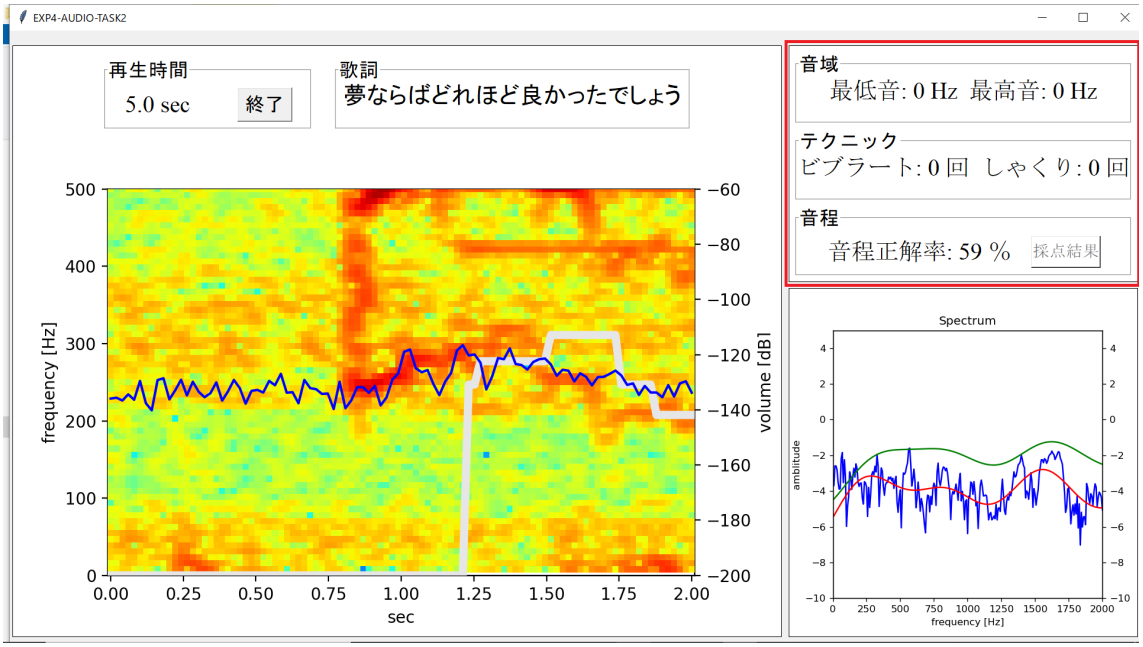

図7　フレーム 2 の内容

以下にフレーム 2 の各部分の実装を示す.

```
# 音域を表示するラベルフレーム
range_frame = tkinter.LabelFrame(master=frame2, text="音域", font=("Arial", 25),
width=640, height=120, background="white")
range_frame.propagate(False)
range_label = tkinter.Label(master=range_frame, text="最低音:   Hz   最高音:   Hz",
font=("Times New Roman", 30), background="white")

# テクニック(ビブラートとしゃくりの回数)を表示するラベルフレーム
technique_frame = tkinter.LabelFrame(master=frame2, text="テクニック", font=("Arial", 25),
width=640, height=120, background="white")
technique_frame.propagate(False)
technique_label = tkinter.Label(master=technique_frame, text="ビブラート:   回   しゃくり:
回", font=("Times New Roman", 30), background="white")

# 音程の正解率と採点結果を表示するラベルフレーム
interval_frame = tkinter.LabelFrame(master=frame2, text="音程", font=("Arial", 25),
width=640, height=120, background="white")
interval_frame.propagate(False)
interval_label = tkinter.Label(master=interval_frame, text="音程正解率:   %",
font=("Times New Roman", 30), background="white")

# 「採点結果を表示」のボタンが押されたときに呼び出される関数
# 最終スコアを表示
def show_score():
  global interval_correct_rate, interval_score, technique_score, final_score
  bt_score.pack_forget()
  interval_label.pack_forget()
  calc_score()
  interval_frame["text"] = "採点結果"
  interval_label["padx"] = 5
```

```
  interval_label["text"] = "音程: %s 点 +  技術: %s 点 = %s 点" %
  (interval_score, technique_score, final_score)
  interval_label.pack()
  return

# 採点結果を表示するボタン
bt_score = tkinter.Button(interval_frame, text="採点結果", font=("Times New Roman", 20),
background="white", command = show_score, padx=10, state=tkinter.DISABLED)
```

各領域はラベルフレームによって作成しており，関数 update_gui_text によってテキストの内容が随時更新されるようになっている．また採点結果を表示させるためのボタンとコールバック関数 show_score を作成し，楽曲終了後に押すと図 8で示すように最終的なスコアが表示されるようになっている．このボタンは初期ステータスを DISABLED にすることで楽曲が終了した後にしか押せないようにし，押した後には GUI 上から消えるように工夫した．

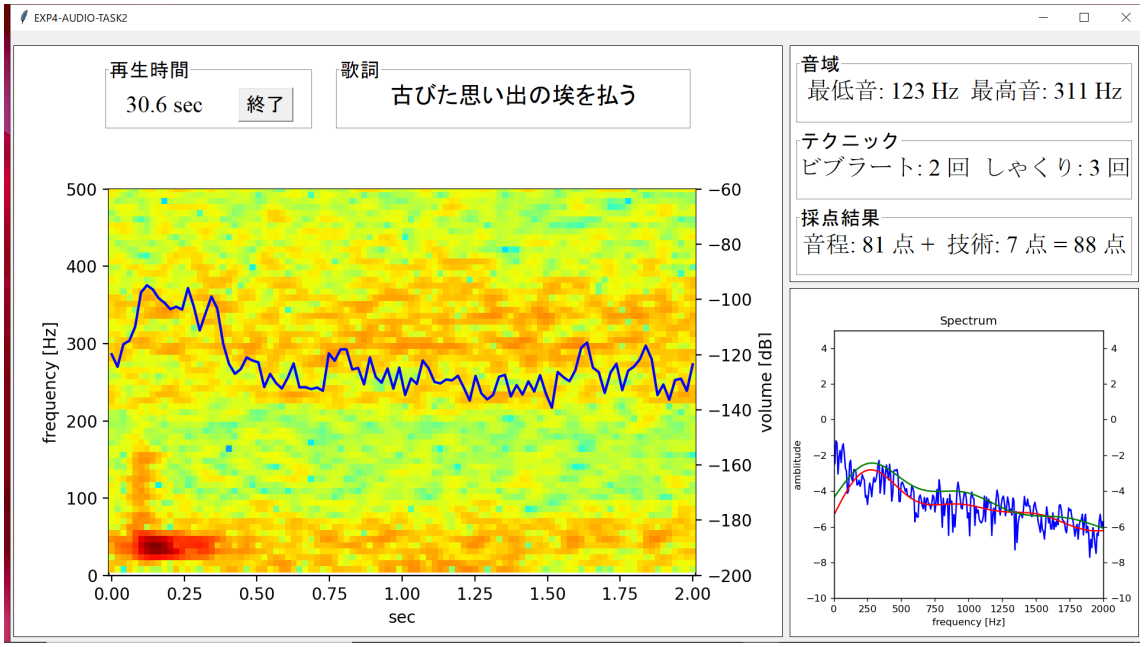

図8　採点結果の表示

2.3.4　フレーム 3

図 9の赤枠で示すようにフレーム 3 は音声のスペクトルとスペクトル包絡，楽曲のスペクトル包絡を同時に表示する部分である．

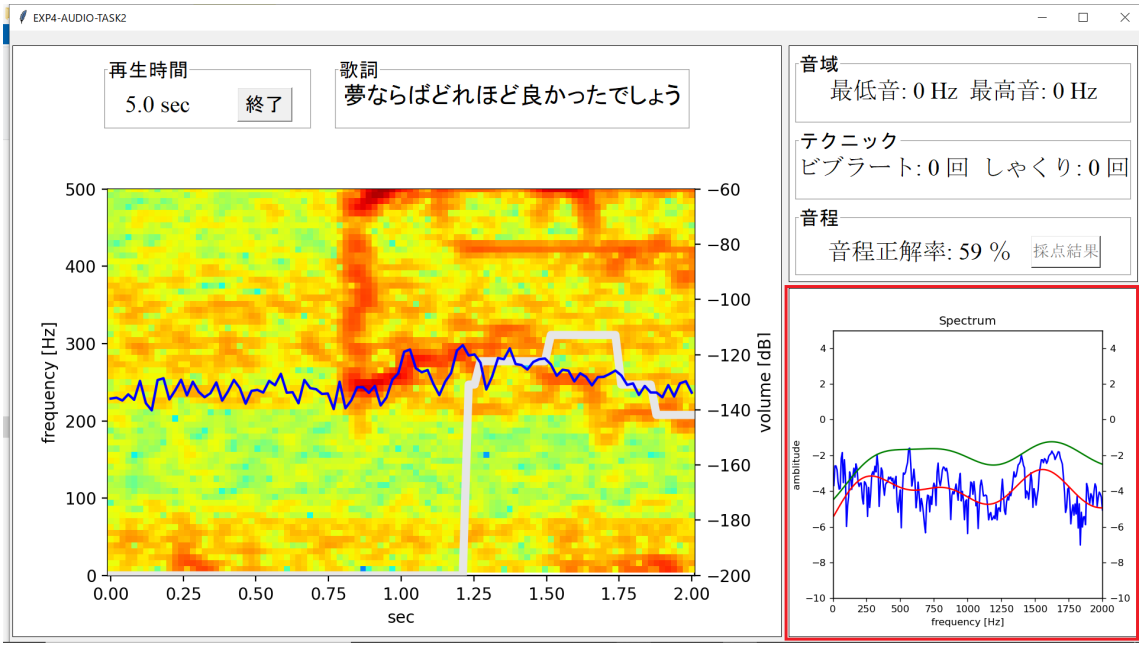

図9　フレーム 3 の内容

以下にフレーム 3 のグラフ描画の実装を示す.

```
# 音声のスペクトル，音声と楽曲(ボーカル)のスペクトル包絡を描画する関数
def draw_graph():
  global is_gui_running, is_music_running
  while is_gui_running and is_music_running:
    ax3.cla()
    ax4.cla()

    ax3.plot(freq_y_data_full, spectrum_data, c='b')
    ax3.plot(freq_y_data_full, spectrum_envelope, c='r')
    ax3.set_ylabel('amblitude')
    ax3.set_xlabel('frequency [Hz]')
    ax3.set_title("Spectrum")
    ax3.set_ylim(-10, 5)
    ax3.set_xlim(0, 2000)

    ax4.plot(freq_y_data_full, spectrum_envelope_music, c='green')
    ax4.set_ylim(-10, 5)
    ax4.set_xlim(0, 2000)
    canvas2.draw()
    time.sleep(1.0)

# スペクトルを表示する領域を確保
# ax3, canvs2 を使って上記の関数でグラフを描画する
fig2 = plt.figure(figsize=(5, 5), dpi = 60)
canvas2 = FigureCanvasTkAgg(fig2, master=frame3)
canvas2.get_tk_widget().pack()
ax3 = fig2.add_subplot(111)
ax3.plot(freq_y_data_full, spectrum_data, c='b')
ax3.plot(freq_y_data_full, spectrum_envelope, c='r')
ax3.set_ylabel('amblitude')
```

```
ax3.set_xlabel('frequency [Hz]')
ax3.set_title("Spectrum")
ax3.set_ylim(-10, 5)
ax3.set_xlim(0, 2000)

ax4 = ax3.twinx()
ax4.plot(freq_y_data_full, spectrum_envelope_music, c='green')
ax4.set_ylim(-10, 5)
ax4.set_xlim(0, 2000)
canvas2.draw()
```

ここではパワーを縦軸とする ax3 とそれを反転した軸 ax4 を作成し音声のスペクトルとスペクトル包絡，楽曲のスペクトル包絡を同時に表示するようにした．上の図 9で示すように，音声のスペクトルは青色，音声のスペクトル包絡は赤色，楽曲のスペクトル包絡は緑色で表現している．また，再描画のための関数 draw_graph を作成し，GUI が終了していないかつ楽曲が終わっていない限り 1 秒ごとに繰り返し描画されるように実装した．この関数は t_update_spectrum_graph という別スレッドで実行されている．

3　振り返り

3.1　工夫した点

簡易カラオケシステムの作成にあたって特に工夫した点はユーザーにとって視覚的にわかりやすいシステムの実現を目指したことである．実際のカラオケシステムを参考に，ガイドメロディと音声の音高の効果的な表示を考え実装を行った．また，歌詞や音量，音域，音程正解率等をリアルタイムで更新しながら表示し，マイク入力を行いながら音声情報を可視化するようにした．同時に，ビブラートやしゃくりといった歌唱テクニックの判定を定義から独自に実装しそれを踏まえた採点機能の導入も実現した．さらに，音声のスペクトル包絡と楽曲のスペクトル包絡を同時にグラフで表示するようにした点も工夫の 1 つである．今回は楽器音に比べてボーカルの声量が大きい楽曲を選択しており，2 つのスペクトル包絡を比較することでユーザーは声道フィルタの振幅特性をそのアーティストと比べることができる．このように様々な要素をシステムに組み込み，カラオケシステムの改良を図った．

視覚的なわかりやすさの向上を目指す上でグラフの表示についてもいくつか工夫を行った．例えばグラフの平滑化により飛び値や不具合の影響を受けにくくしたり，音量のグラフと音高のグラフが被りにくくなるように縦軸の表示範囲を設定したりした．また，音高やガイドメロディの表示において値が 0 の点のプロットは見にくいと感じたため，欠損値 NaN に変換してグラフ上には表示されないようにもした．

3.2　改善点

当初の実装目標はほとんどクリアできたが，音源分離がうまくできず楽曲からのボーカル抽出は断念した．ボーカル部分を綺麗に抽出できればスペクトル包絡を比べることで，アーティストとの声道特徴の類似性をより正確に測れるため機会があればチャレンジしてみたいと思う．また今回のガイドメロディは手動による打ち込みであり，やや恣意的な部分があるためもう少し工夫の余地はあるように思える．歌唱テクニックや採点機能についてもより多様なテクニックに注目したり判定方法を改善したりできると思う．