2024年度　信号処理特論1

36414067 飯田 諒
メディア情報プログラム
徳田・南角・橋本研究室

2024年6月29日

- 使用した音：自分自身の声
- 使用したソフト：WaveSufer
- 使用した機材：自身の計算機
- 使用した OS：Windows
- プログラミング言語：Python

課題のソースコードの詳細は資料の最後尾にまとめて載せる。

1　課題 1

まず、フーリエ級数展開の式は以下のとおりである。

$$A_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)\cos\left(\frac{2\pi}{T}nx\right)dx$$

$$B_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)\sin\left(\frac{2\pi}{T}nx\right)dx$$

この式を、離散データに対して使うために変換する。主に A_n について考える。
この式の意味は、正規直交基底を満たす、$\frac{2}{T}\cos\left(\frac{2\pi}{T}nx\right)(-\frac{T}{2} \leq x \leq \frac{T}{2}), (n = 1, 2, 3, ...)$ と、関数 $f(x)$ との内積を取って、$f(x)$ の成分分解をしている。(係数である $\frac{2}{T}$ は、$\left\langle \cos\left(\frac{2\pi}{T}nx\right), \cos\left(\frac{2\pi}{T}nx\right)\right\rangle = \int_{-\frac{T}{2}}^{\frac{T}{2}} \cos^2\left(\frac{2\pi}{T}nx\right)dx = \frac{T}{2}$ であるため、正規性を保つために割っている。) また、$\frac{2}{T}\cos\left(\frac{2\pi}{T}nx\right)$ は、$-\frac{T}{2} \leq x \leq \frac{T}{2}$ の間で、n 回振動しているから、$n[Hz]$ の信号である。これらの理由で、フーリエ級数展開は $0 \sim n[Hz]$ までの成分を分解している。この原理を利用して、今回の離散信号に対して A_n を求める。
$\cos\left(\frac{2\pi}{Fs}x\right)$ は、$0 \leq x \leq Fs$ で、$1[Hz]$ であるから、$f[Hz]$ は $\cos\left(\frac{2\pi}{Fs}fx\right)$ である。また、x は整数値しか取れないため、 積分を考えるとき区分求積法を考える。区分求積法の式は以下のとおりである。

$$\int_b^a f(x)dx = \lim_{n\to\infty}\frac{1}{n}\sum_{k=bn}^{an-1} f\left(\frac{k}{n}\right)$$

x が整数のみを考えたいため、$n = 1$ とすると、

$$\int_b^a f(x)dx = \sum_{k=bn}^{an-1} f(k)$$

となる。そして、$T[s]$ で周期性があると考えると、$N = Fs \cdot T$ 点で周期を持っている。これらの考え方から、$A(f)$ を定義すると、

$$A(f) = \int_0^N f(x)\frac{\cos\left(\frac{2\pi}{Fs}fx\right)}{\left\langle \cos\left(\frac{2\pi}{Fs}fx\right), \cos\left(\frac{2\pi}{Fs}fx\right)\right\rangle}dx \tag{1}$$

$$= \frac{1}{\left\langle \cos\left(\frac{2\pi}{Fs}fx\right), \cos\left(\frac{2\pi}{Fs}fx\right)\right\rangle}\int_0^N f(x)\cos\left(\frac{2\pi}{Fs}fx\right)dx \tag{2}$$

$$\approx \frac{1}{\left\langle \cos\left(\frac{2\pi}{Fs}fx\right), \cos\left(\frac{2\pi}{Fs}fx\right)\right\rangle}\sum_{k=0}^{N-1} f(k)\cos\left(\frac{2\pi}{Fs}fk\right) \tag{3}$$

この時、$\left\langle \cos\left(\frac{2\pi}{Fs}fx\right), \cos\left(\frac{2\pi}{Fs}fx\right)\right\rangle$ を計算すると、

$$\left\langle \cos\left(\frac{2\pi}{Fs}fx\right), \cos\left(\frac{2\pi}{Fs}fx\right)\right\rangle = \int_0^N \cos^2\left(\frac{2\pi}{Fs}fx\right)dx \tag{4}$$

$$= \cdots \tag{5}$$

$$= \frac{1}{2}(N + \frac{Fs}{4\pi fN} sin(\frac{4\pi}{Fs}fN)) \tag{6}$$

$$\approx \frac{2}{N} \tag{7}$$

最後の近似は、$\left|\frac{\sin x}{x}\right| \leq 1 \ll N$ を用いた。以上より、B_n も同様の議論をすると、

$$A(f) = \frac{N}{2}\sum_{k=0}^{N-1} f(k)\cos\left(\frac{2\pi}{Fs}fk\right)$$

$$B(f) = \frac{N}{2}\sum_{k=0}^{N-1} f(k)\sin\left(\frac{2\pi}{Fs}fk\right)$$

この時、f は連続値を取ってよいが、今回求めたい $H_n(n$ は整数) は、$n[Hz]$ の周波数の係数を表すため、$H_n = \sqrt{A_n^2 + B_n^2} = \sqrt{A(n)^2 + B(n)^2}$ である。
本問題では H_n を求めるという問題であるが、連続値にできるため線グラフで表す。

サンプリング周波数は Fs = 16000 とした。
N をどんな値にするかに関して、N が大きくなると高い周波数分解能が得られるが、時間分解能 (より局所的な時間を見る能力) が減り、逆に N を小さくすると周波数分解能が低くなり時間分解能が高くなる。したがって、これら二つの能力はトレードオフの関係にある。今回は、倍音について深く考察したいと考えたため、N を大きくし周波数分解能を高くした。具体的には N=1024 にした。

プログラムについて述べる。最初に音声データの最大値を取り、時系列すべてに対して割ることで正規化をする。次に、適当に音声が出ているところを選び、N 点とるために、全データ区間から、start_iter šstart_iter +N を切り出す。次に、ブラックマン窓を掛ける。次に、A(f) を求めるため、Af[] という配列を作り、for 文で足し合わせ、最後に $\frac{2}{N}$ を掛けた。また今回は f を滑らかに描画したいと考えたため、f を計算する間隔を delta_f という変数で定めた。しかし、delta_f が 0.5 の時は A[0.5],A[3.5] といった配列は用意できないため、$Af[1] = A(0.5), Af[2] = A(1), Af[3] = A(1.5)$ つまり、$Af[f] = A(f \cdot delta\_f)$ と定めた。最後に Hn を計算して描画するという流れである。ここでは主要なコードのみ載せる。

Listing 1: fourier.py

```
1 data_from_window = np.zeros(len(norm_data))
2 for i in range(N) :
3     data_from_window[i] = norm_data[i+start_iter]*brackman_window(i,N)
4
5 Af = np.zeros(int(max_f/delta_f + 1))
6 Bf = np.zeros(int(max_f/delta_f + 1))
7
8 for f in range(len(Af)) :
9     for j in range(N) :
```

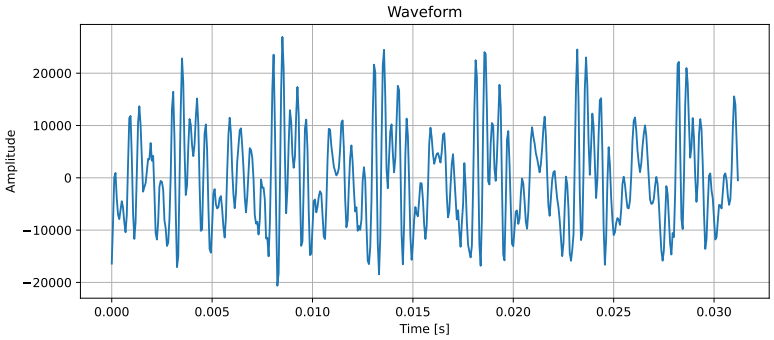

図 1: 入力音声波形

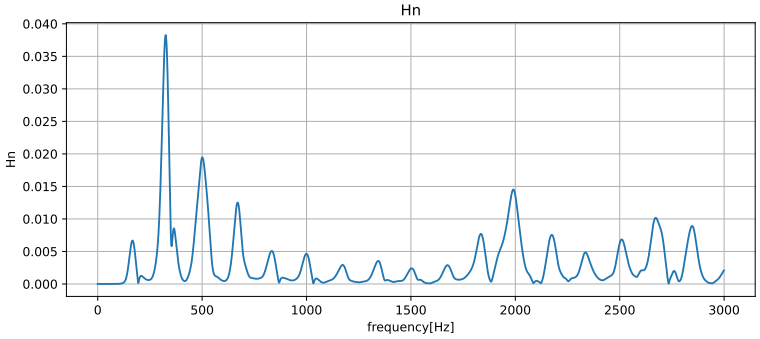

図 2: フーリエ級数展開グラフ,N = 1024,Fs = 16000,delta_f = 0.5

```
10          Af[f] += data_from_window[j] * math.cos(2*math.pi/Fs*f*delta_f*
                j)
11          Bf[f] += data_from_window[j] * math.sin(2*math.pi/Fs*f*delta_f*
                j)
12      Af[f] *= 2/N
13      Bf[f] *= 2/N
```

入力音声波形、フーリエ級数展開のグラフは図 1、図 2 のとおりである。 これを観察すると、167[Hz] 程にフォルマントがあり、倍音として 167 の整数倍のところにもフォルマントがあることが分かる。

2　課題 2

LPF に通した音声波形、フィルタに通した後のフーリエ級数展開のグラフは図 3、図 4 のとおりである。 まず図 1 と図 3 の音声波形を観察する。LPF の関数は時間方向に幅を持った 10 点に関する平均を取っている（時間移動平均）ため、高周波成分に対して鈍感にする性能を持っているフィルタであり、LPF という名の通り、低周波成分を通すフィルタである。図を見比べると、確かに細かな振動、つまり高周波成分がなくなり、荒い波の信号になっていることが読み取れる。

また図 2 と図 4 を見比べると、これも高周波成分である 1000[Hz] 以降からフォルマントが小さくなっていることが読み取れる。また逆に低周波成分では振幅スペクトルが大き

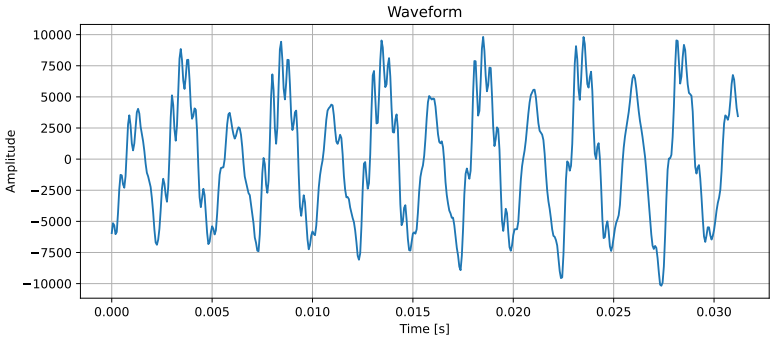

図 3: LPF 通過後音声波形

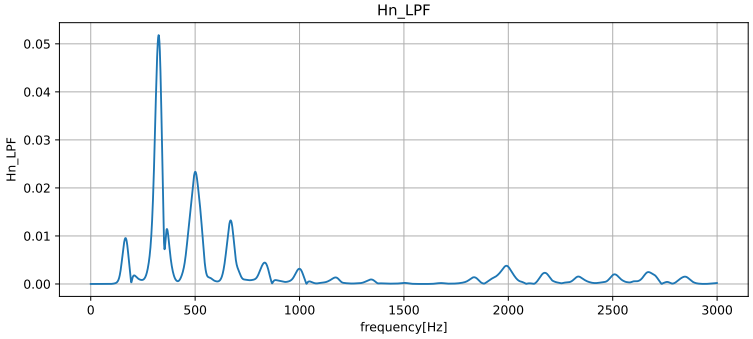

図 4: LPF 通過後フーリエ級数展開グラフ,N = 1024,Fs = 16000 , delta_f = 0.5

くなっていることも読み取れる。これは高周波成分の移動平均が倍音成分と一致する要素があるため、その分大きくなっていると考察する。

3　課題 3

試聴した感想は、LPF を通した後では、音がこもったように聞こえた。これは高周波成分が聞こえなくなったことによって相対的に低周波成分、つまりこもった音の成分が際立って聞こえたためであると考察した。

4　ソースコードの主要部

4.1　課題 1

Listing 2: fourier.py

```
1
2 def brackman_window(t,T) :
3     return 0.42 - 0.5*math.cos(2*math.pi*t/T) + 0.08*math.cos(4*math.
          pi*t/T)
4
```

```
5  def my_fourier(norm_data, sampling_rate, start_iter, N, max_f, delta_f)
       :
6
7      Fs = sampling_rate
8
9      data_from_window = np.zeros(len(norm_data))
10     for i in range(N) :
11         data_from_window[i] = norm_data[i+start_iter]*brackman_window(i,
               N)
12
13     Af = np.zeros(int(max_f/delta_f + 1))
14     Bf = np.zeros(int(max_f/delta_f + 1))
15
16     for f in range(len(Af)) :
17         for j in range(N) :
18             Af[f] += data_from_window[j] * math.cos(2*math.pi/Fs*f*
                   delta_f*j)
19             Bf[f] += data_from_window[j] * math.sin(2*math.pi/Fs*f*
                   delta_f*j)
20         Af[f] *= 2/N
21         Bf[f] *= 2/N
22     return Af, Bf
23
24 def plot(Xn, Xn_name, fname, max_f, delta_f) :
25     frequency = np.arange(0,max_f,delta_f)
26     Xn = Xn[:len(frequency)]
27     plt.figure(figsize=(10,4))
28     plt.plot(frequency,Xn)
29     plt.title("{}".format(Xn_name))
30     plt.xlabel("Hz")
31     plt.ylabel("{}".format(Xn_name))
32     plt.grid()
33     plt.savefig("../figure/{}_{}.pdf".format(fname,Xn_name))
34     plt.figure()
35
36
37 if __name__ == "__main__" :
38     input_wav_file = fpath + ".wav"
39     sampling_rate, data = wavfile.read(input_wav_file)
40     norm_data = data/np.max(data)
41     Af, Bf = my_fourier(norm_data, sampling_rate, start_iter, N, max_f,
           delta_f)
42     Hf = np.sqrt(Af*Af + Bf*Bf)
43     plot(Af, "An", fname, max_f, delta_f)
44     plot(Bf, "Bn", fname, max_f, delta_f)
45     plot(Hf, "Hn", fname, max_f, delta_f)
```

4.2　課題 2

Listing 3: LPF.py

```
1
2 def my_LPF(data, lag) :
3
4     sumdata = np.zeros(len(data)+1)
5     for i in range(len(data)) :
6         sumdata[i+1] = sumdata[i] + data[i]
7
8     y = np.zeros(len(data)-lag+1)
9     for i in range(len(y)) :
10        y[i] = sumdata[i+lag] - sumdata[i]
11    y/=lag
12
13    return y
14
15
16 if __name__ == "__main__" :
17    input_wav_file = fpath + ".wav"
18    sampling_rate, data = wavfile.read(input_wav_file)
19    y = my_LPF(data,lag)
20    norm_y = y/np.max(y)
21    Af, Bf = my_fourier(norm_y, sampling_rate, start_iter, N, max_f,
          delta_f)
22    Hf = np.sqrt(Af*Af + Bf*Bf)
23    plot(Af, "An_LPF", fname, max_f, delta_f)
24    plot(Bf, "Bn_LPF", fname, max_f, delta_f)
25    plot(Hf, "Hn_LPF", fname, max_f, delta_f)
```

4.3　課題 3

Listing 4: LPF_reconst.py

```
1
2 if __name__ == "__main__" :
3     input_wav_file = fpath + ".wav"
4     output_wav_file = "{}_LPF_reconst.wav".format(fname)
5     sampling_rate, data = wavfile.read(input_wav_file)
6     y = my_LPF(data,lag)
7     y_ = np.zeros(lag-1)
8     y_con = np.concatenate((y,y_)).astype(np.int16)
9     wavfile.write(output_wav_file, sampling_rate, y_con)
```