音声情報処理演習まとめ

学籍番号: 1029332978 氏名: 上野山遼音

2023 年 12 月 8 日

目 次

1 演習 2 **2**

2 演習 3 **4**

3 演習 4 **5**

3.1 バタフライ演算 5

3.2 FFT の実践 (手計算) 5

4 演習 5 **5**

5 演習 6 **5**

6 演習 7 **5**

1　演習 2

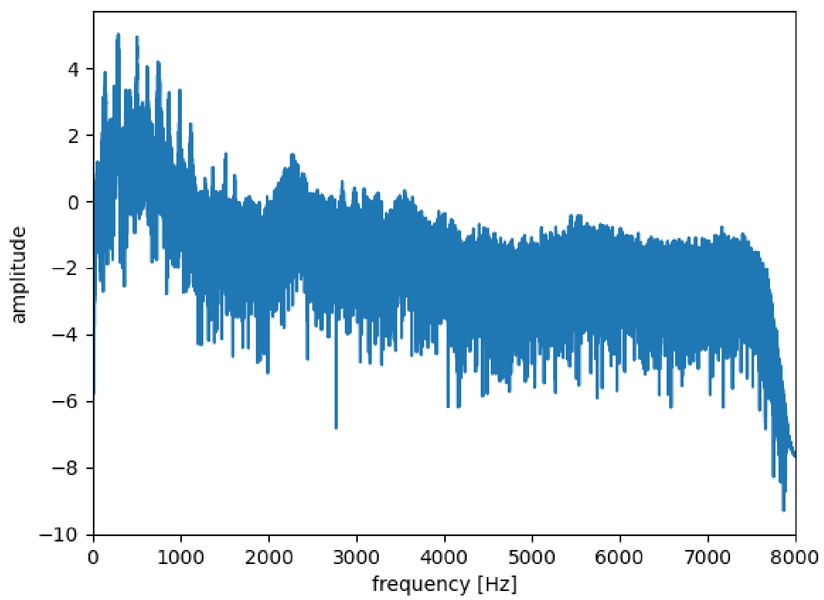

図 1: aiueo.wav の波形とスペクトル

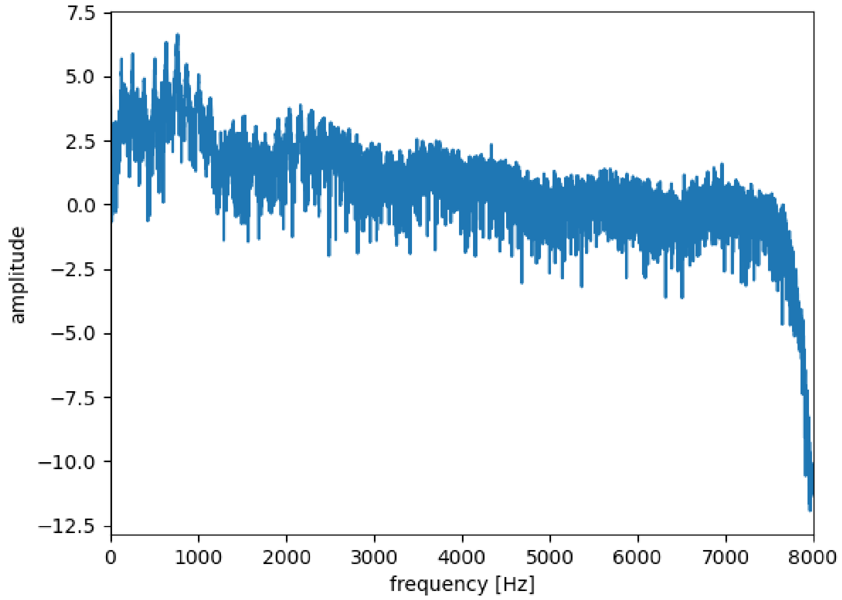

図 2: a.wav の波形とスペクトル

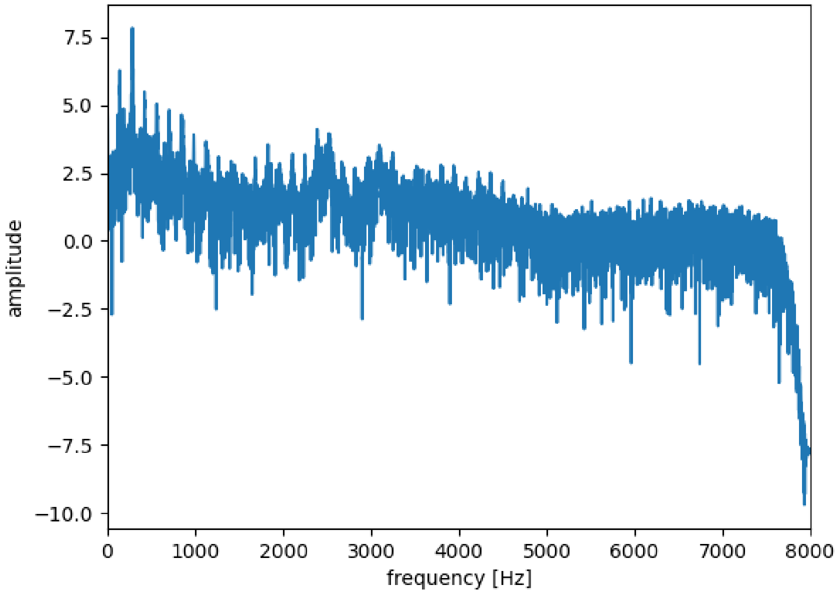

図 3: i.wav の波形とスペクトル

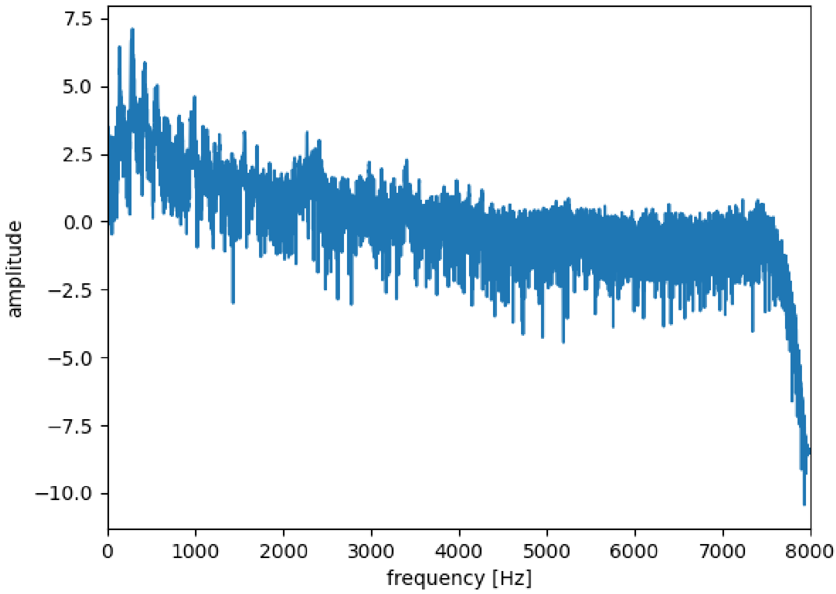

図 4: u.wav の波形とスペクトル

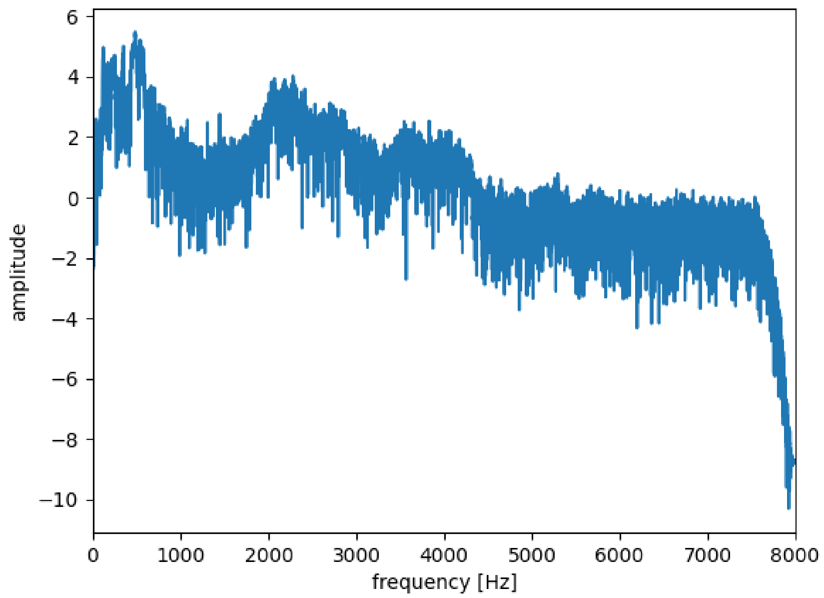

図 5: e.wav の波形とスペクトル

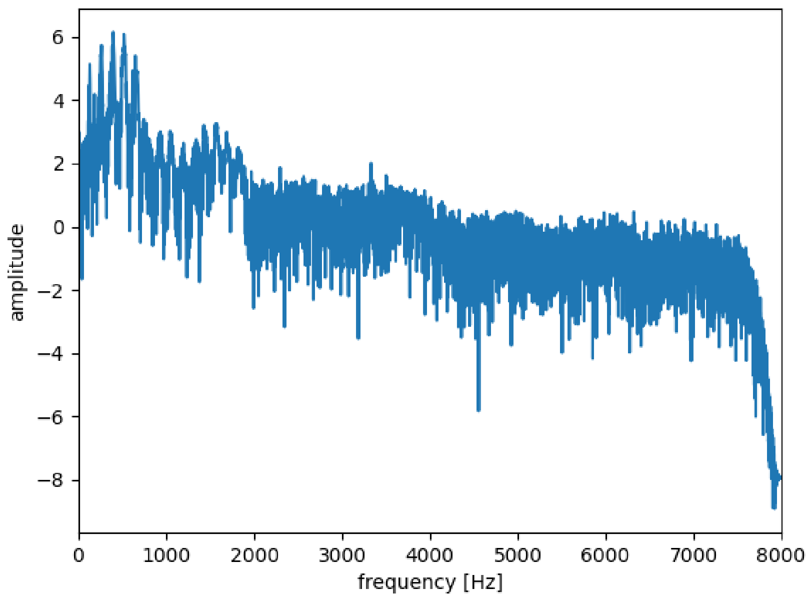

図 6: o.wav の波形とスペクトル

2　演習 3

実数入力に対する 1 次元離散フーリエ変換を計算するするライブラリ．高速フーリエ変換（FFT）によって、実数値配列の 1 次元 n 点離散フーリエ変換（DFT）を計算する．入力は実数値配列で、出力は複素数値配列である．入力には他にも n(使用する入力内の変換軸に沿ったポイントの数, optional), axis(変換を行う軸, optional) を指定できる．

※純粋な実数入力に対して DFT を計算すると、出力はエルミート対称 (その成分は任意の添字 i, j について (i, j) 成分は (j, i) 成分の複素共役と等しい.) になる．つまり、負の周波数項は対応する正の周波数項の複素共役にすぎない．したがって負の周波数項は冗長になるので RFFT では負の周波数項を計算

しない．その結果、出力の軸の長さは $\lfloor n/2 \rfloor + 1$ になる．

3 演習 4

3.1 バタフライ演算

DFT について　そもそも、DFT では N 点の実変数 $f(0), f(1), ..., f(N-1)$ を離散フーリエ変数 $F(0), F(1), ..., F(N-1)$ に変換するために N 次正方行列である DFT 行列を掛け合わせているのだった．この計算では，$O(N^2)$ となる．この計算量を軽減するために、高速フーリエ変換（FFT）が考案された．

FFT の計算量　上述の通り DFT では計算量が多いので、バタフライ演算を用いて計算量を減らしている．

3.2 FFT の実践 (手計算)

input $= (1, 0, 3, 2, 4, 0, 2, 0)^T$

$$(\text{略}) \tag{1}$$

以下で計算が正しいことを検証する．

Listing 1: FFT の実践 (numpy 篇)

```
>>> import numpy as np
>>> np.fft.fft([1,0,3,2,4,0,2,0])
array([12. +0.j , -4.41421356-2.41421356j,
      0. +2.j , -1.58578644-0.41421356j,
      8. +0.j , -1.58578644+0.41421356j,
      0. -2.j , -4.41421356+2.41421356j])
```

4 演習 5

使用した窓関数はハミング窓である．フレームサイズは 512，シフト長は SR$(= 16000)/100 = 160$ とした．スペクトログラムの縦軸は周波数であり，標本化定理に基づきプロット区間は 0 から 8000Hz までとした．また，横軸は時間である．濃淡が x 軸上の時間における y 軸上の周波数の強さを表す。

5 演習 6

np.fft.rfft と np.fft.fft の違いについて　どちらも高速フーリエ変換 (FFT) を行うものではあるが，np.fft.rfft は実数の入力に対して FFT を行い，np.fft.fft は複素数の入力に対して FFT を行う．

rfft が存在する理由は、先述のエルミート対称性を利用し、必要な計算数を削減することができる点にある．

6 演習 7

DFT は (2) 式で表される．

$$F(t) = \sum_{x=0}^{N-1} f(x) e^{-2\pi i x t/N} dx \tag{2}$$

ただし，$N = 2^M (M \in Z^+)$ である．

Listing 2: FFT 実装

```
import numpy as np
import time


def FFT(f: np.ndarray) -> np.ndarray:
    n = len(f)
    w = np.exp(-2j * np.pi / n)
    w_N = w ** np.arange(n//2)
    if n == 1:
        return f[0]
    F_even = FFT(f[::2])
    F_odd = FFT(f[1::2])
    F = np.zeros(n, dtype=np.complex128)
    F[0:n//2] = F_even + w_N * F_odd
    F[n//2:] = F_even - w_N * F_odd

    return F


input_array = np.arange(2**14, dtype=int)
start = time.perf_counter()
FFT(input_array)
end = time.perf_counter()
print(end - start)
start = time.perf_counter()
np.fft.rfft(input_array)
end = time.perf_counter()
print(end - start)

print(np.allclose(FFT(input_array), np.fft.fft(input_array), atol=1e-10))
```