電子制御工学実験報告書

実験題目　：　**フーリエ解析**

報告者　：　4 年 42 番　鷲尾 優作

提出日　：　2023 年 1 月 31 日

実験日　：　2023 年 1 月 5 日，1 月 19 日

実験班　：

共同実験者　：

※ 指導教員記入欄

評価項目	配点	一次チェック ・　・	二次チェック ・　・
記載量	20		
図・表・グラフ	20		
見出し，ページ番号，その他体裁	10		
その他の減点	–		
合計	50		

コメント：

1　研究テーマ

電気自動車充電器・メガソーラ発電設備等の高周波騒音の軽減

2　背景・目的

再生可能エネルギーの普及に伴い，日本各地で大電力を扱う設備が日常に溶け込む形で普及しつつあるなか，近年問題となっているのが励磁音である．励磁音を発生させる代表的な装置であるパワコンユニットを備えた設備を図 1 に示す．

(a) メガソーラー発電所のパワコンユニット　　**(b)** 電気自動車急速充電器

図 1: 励磁音の発生源

パワコンユニットは、送電ネットワークと蓄電に向いた直流の変換を担う装置である．山間部のメガソーラー発電所のほか，都市部に設置された電気自動車急速充電器に使用され，動作中はキーンという高周波の音を発生させ，近隣住民とのトラブルの誘因に加え，生態系への影響も懸念されている．同様の家庭用設備では睡眠障害を引き起こすなど実生活に被害をもたらす問題であるといえる．

本実験では，生態系や人間の生活に対する発電設備の共存能力を向上させるため、励磁音の特性を解析し，騒音成分を軽減する方法を検討する．

3　解析手法の検討

3.1　励磁音の発生源特定

有効に励磁音を軽減するためには，励磁音の発生源を詳細に特定することが必要となる．しかしながら本実験では，パワコンユニットを分解することは不可能であるため，パワコンユニットの内部にあるコイルや電極の振動を直接測定することは困難である．

一般的な考え方として，励磁音がパワコンユニットの内部にある複数のコイルや電極の振動によって発生していると仮定する．この場合，パワコンユニット外部に流れる励磁音は，各発生源の振動によって発生する励磁音の和であ

ると推定される．今回はこの予測に基づき，パワコンユニット外部に流れる励磁音を測定し，フーリエ変換により励磁音の周波数特性を解析，発生源を特定することを目指す．

3.2 離散フーリエ変換（DFT）

コンピュータ上でフーリエ変換による周波数解析を実現する方法として，離散フーリエ変換（DFT）がある．DFTは，本来連続関数に適用するフーリエ変換を，マイクロフォン等から取得した離散的な時系列データの解析に用いるためのアルゴリズムである．実装が比較的容易であること，入力するデータの特性を考慮し，本実験ではDFTを用いて励磁音の周波数特性を解析する．

■離散フーリエ変換の理論

離散フーリエ変換の理論について述べる．前提として連続関数に適用するフーリエ変換は，有限区間 $[-T/2, T/2]$ において式1で表される．

$$F(\omega) = \int_{-T/2}^{T/2} f(t)e^{-i\omega t}dt \tag{1}$$

しかしながら，離散的な時系列データに対して式1を適用すると積分計算が実施できず，0が出力されてしまう．離散フーリエ変換は，この問題を解決するため入力信号のサンプリング周期に合わせ，複素フーリエ級数を用いて式1を近似するものである．式1を複素フーリエ級数で近似すると，各周波数 ω におけるフーリエ成分 $F(\omega)$ は式2で表される．

$$F(\omega) = \sum_{n=0}^{N-1} f(n)e^{-i\omega n} \tag{2}$$

■アルゴリズムの実装

上記アルゴリズムを実装するために，Rustによる DFTの関数を作成した．Rustは，C/C++と同様に高速な実行速度を実現することができることに加え，メモリの安全性を保証することができる言語である．将来的なFFTへの改造を考慮し，Rustを用いて実装した．入力する時系列データは64bitの浮動小数点数を想定している．

リスト1に作成したRustによるDFTの関数実装を示す．

リスト 1: fourier.rs

```
pub mod transform {
    use num::complex::Complex;
    use std::f64::consts::PI;

    pub fn dft(frames: Vec<f64>, sampling_freq: i64) -> Vec<Complex<f64>> {
        let mut spectrum: Vec<Complex<f64>> = Vec::new();
        for i in 1..=(sampling_freq / 2) {
            let mut sigma = Complex::new(0.0, 0.0);
            for (j, frame) in frames.iter().enumerate() {
                let delta_t = (j as f64) / (sampling_freq as f64);
                let omega = 2.0 * PI * (i as f64);
                let exponent = Complex::new(0.0, -omega * delta_t);
```

```
13                 sigma += frame * exponent.exp();
14             }
15             spectrum.push(sigma);
16         }
17         return spectrum;
18     }
19 }
```

関数の入力と出力は，それぞれ入力信号 $f(n)$ と各周波数 ω におけるフーリエ成分 $F(\omega)$ とする．関数の入力として，可変長の入力信号 $f(n)$ の時系列ベクタ frames と入力信号のサンプリング周期 sampling_freq を与える．関数の出力としては，可変長の Complexf64 型のフーリエ成分を格納するベクタ spectrum を返すものとする．

本関数は，リスト 2 に示すように実行する可能である．実験中においては振幅スペクトルの大きさを補正するためにデータサイズの半分で割る処理が可能であるが，こちらは関数実装に含める必要性が薄いと判断し，関数の返り値をここで補正する処理を実施している．

リスト 2: main.rs

```
39     let spectrum = fourier::transform::dft(frames, sampling_freq);
40 
41     for (i, s) in spectrum.iter().enumerate() {
42         let amp = s.norm() * 2.0 / matrix.shape()[0] as f64;
43         println!("{},␣{}", i, amp);
44     }
```

4 環境

本実験は次の環境で実施した．

- PC: MacBook Pro Apple M1 16GB (MacOS 13.1)
- Rust: 1.52.1
- Python: 3.9.13 with anaconda

Rust 依存環境

- csv: 1.1.6
- ndarray: 0.15.6
- num: 0.4.0

5 解析正当性の確認

作成した Rust による DFT の関数が正しく動作しているかを確認するため，既知の正弦波の合成波形を入力し，応答を確認する．ここでは，振幅 5 の波形を入力するものとし 200Hz と 1000Hz の正弦波の合成波を用いた評価を case1，100Hz，500Hz，2000Hz の正弦波の合成波を用いた評価を case2 とする．

解析データの評価を簡単とするため，同関数を呼び出しグラフ化する Python のスクリプトを作成した．スクリプトは，入力波形と計算後の振幅スペクトルを matplotlib を用いて可視化するものである．スクリプトを，リスト 3, リスト 4 に示す．本実験では，解析はこのスクリプトを通して実施する．

リスト 3: main.py

```
8  def transform(input_file_name, output_file_name):
9      start = time.perf_counter()
10     with open(output_file_name, 'w') as f:
11         subprocess.run(['./target/debug/rust_fourier_transform',
12                         input_file_name, '8000'], stdout=f, stderr=subprocess.DEVNULL)
13     return (time.perf_counter() - start)
```

リスト 4: main.py

```
40 if __name__ == '__main__':
41     input_file_name = "sin_100_500_2000.txt" # 入力ファイル名
42     output_file_name = "out.csv" # 出力ファイル名
43     max_freq = 200 # 予想される最大周波数
44
45     print("--␣関数読み込み␣--")
46     input_wave = pd.read_csv(input_file_name, delimiter=",",
47                              header=None, names=("time", "amp"))
48
49     print("--␣解析開始␣--")
50     elapsed_time = transform(input_file_name, output_file_name)
51
52     print("--␣解析終了␣--␣time:␣{:.10f}␣sec".format(elapsed_time))
53
54     output_wave = pd.read_csv(output_file_name, delimiter=",",
55                               header=None, names=("freq", "amp"))
56
57     print("--␣グラフ描画␣--")
58     # matplotlib
59     graph(input_wave, output_wave, max_freq)
```

5.1 入力想定データ

入力想定データとして，サンプリング周波数 8000Hz，データ数 8000 個のデータを 2 つ用意し Rust プログラムの正当性を評価した．データは，1 列目に時刻，2 列目に振幅を格納したテキストファイルとして用意した．

case1 で用いたデータの一部を参考として，リスト 5 に示す．

リスト 5: sin-200-1000.txt

```
0.000000,0.000000
0.000125,4.004837
0.000250,5.927051
0.000375,4.897505
0.000500,1.763356
```

5.2 case1 200Hz と 1000Hz の正弦波の合成波

case1 で用いたデータを入力として，Python スクリプトを実行した応答を図 2 に示す．処理は 2.81 秒で正常に終了した．

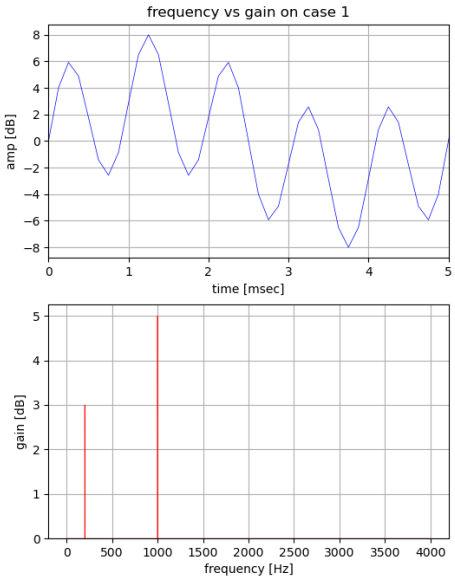

図 2: case1 における DFT 結果

図 2 からは，200Hz と 1000Hz にピークを持つ振幅スペクトルが得られていることが確認できる．

5.3 case2 100Hz，500Hz，2000Hz の正弦波の合成波

case2 で用いたデータを入力として，Python スクリプトを実行した応答を図 3 に示す．処理は 2.80 秒で正常に終了した．

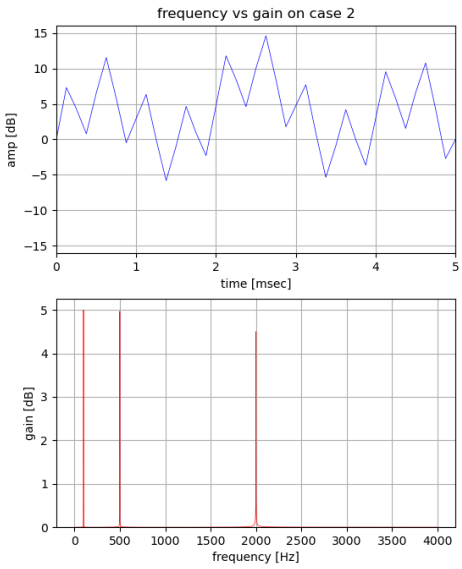

図 3: case2 における DFT 結果

図 3 からは，100Hz，500Hz，2000Hz にピークを持つ振幅スペクトルが得られていることが確認できる．

5.4 結言

case1，case2 ともに入力に対し意図した変換を実施でき，作成した Rust 関数および可視化用 Python スクリプトを用いて，正弦波の合成波の振幅スペクトルを正しく計算することができたといえる．

参考文献

[1] 外山 茂浩、実験テキスト「フーリエ解析」、(2022 年)

[2] 日経 BP、パワーコンディショナー（PCS）とは、
https://project.nikkeibp.co.jp/ms/article/WORD/20130925/305286、(2013 年 10 月)

[3] エネチェンジ、EV(電気自動車) の気になる充電！時間や場所、料金は？、
https://enechange.jp/articles/ev-charging、(2021 年 12 月 19 日)