

1 目的

今日では、あらゆるデバイスで信号の解析や処理を行っている。電子デバイスの設計や、機器の制御に対して、信号処理、信号解析は密接な関係にあり、開発者にとってその技術は必要不可欠である。本実験では、Scilab を使用して信号処理・解析の基礎を行う。今回は、信号の中でも代表的な音声信号を対象とし、Scilab での解析手法について学ぶのが実験の目的である。

2 実験環境

実験環境を以下の表 1 に示す。

表 1: 実験環境

デバイス	OS	ソフト
MacBookAir2019 13inch	MacOS BigSur 11.3.1	Scilab 6.1.0

3 実験方法

実験指導書 [1] に示される課題に対して、それを Scilab を用いて解く。

4 理論

4.1 標本化定理 (サンプリング定理)

標本化定理 (サンプリング定理)[2] とは、時間領域と周波数領域で相対性を持っており、 $f_m[\text{Hz}]$ 以上の周波数成分を持たない帯域制限信号 $f(t)$ は、 $\frac{1}{2}f_m[\text{sec}]$ より短い等しい間隔の標本 (サンプル) で完全に決定されることである。つまり、元の信号の周波数成分の 2 倍以上の高い周波数でサンプリングすることで標本から元の信号が復元できることになる。

4.2 窓関数

コンピュータ内で周期性が成り立たない信号に対して FFT を行うとき、周波数の先頭から N 個のサンプルを取ると、区間の両端の値が異なってしまう。このとき、フーリエ変換のグラフには本来の信号には関係のない余分なスペクトルが出力されてしまい、カクカクなグラフになる。そこで、周期性の無い信号で余分なスペクトルが出にくくなるようにうまく信号を切り取るのが窓関数である。窓関数の中で代表的なものは、信号を矩形波で切り取る「矩形窓」や、両端がなめらかに絞られている「ハミング窓」がある。

4.3 畳み込み

畳み込み (畳み込み積分) とは、関数 f を平行移動させながら関数 g を重ね足し合わせる二項演算のことであり、以下の式 1 ように定義できる。

$$(f * g)(t) = \int f(\tau)g(t - \tau)$$

5 実験結果

5.1 演習 4

さまざまな音を作ってみよう。

ここで作成したソースコードを以下の Code1 に示す。また、作成した音の波形を図 1 に示す。

```

1 t = 0: 1/44100: 2; //時間軸の作成
2 f = linspace(500, 300, length(t)); //周波数軸の作成
3 y = 0.9 * sin(2 * %pi * f.*t); //周波数(音)の作成
4 wavwrite(y, 44100, 16, 'enshu4.wav'); //音声データの書き出し
5 sound(y,44100); //音を鳴らす
6 scf(0);
7 xlabel('t');
8 ylabel('y');
9 plot(t,y);
10
11 xs2png(0, 'kadai4.png') //グラフをpngで保存

```

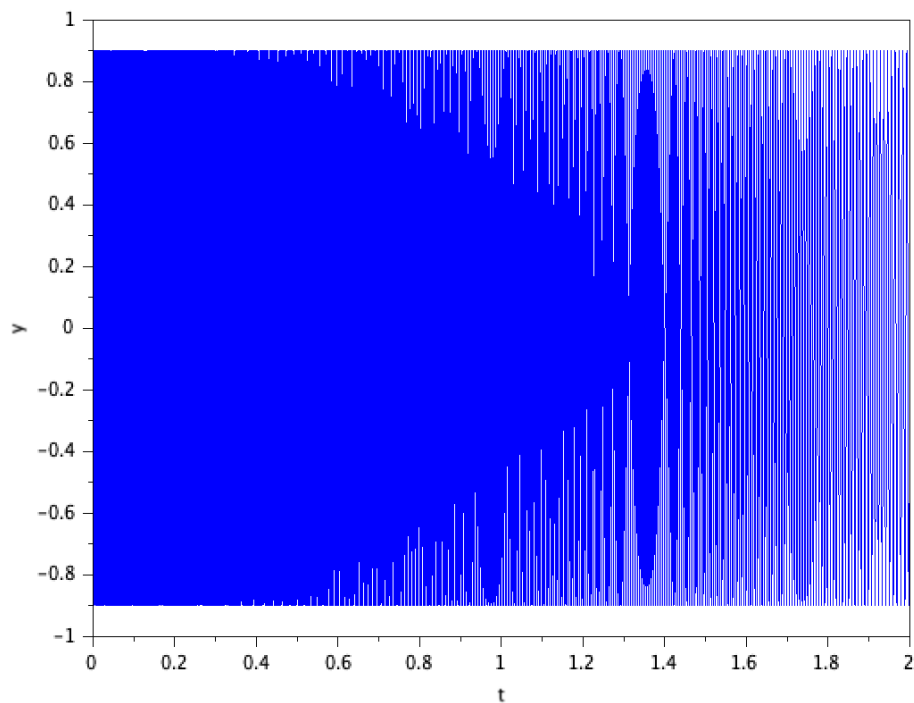


図 1: 演習 4 の出力波形

本実験では、周波数が 500 Hz ~ 300 Hz まで時間とともに低くなる音を作成した。図 1 から時間経過するごとにどんどん周波数が低くなっていることがわかる。

5.2 例題 6

ここで作成したソースコードを以下の Code2 に示す。また、作成した音をフーリエ変換したグラフを図 2 に示す。

```

1 t = 0 : 1/44100 : 2; //時間軸の作成
2 y = 0.9 * sin(2 * %pi * 440 * t); //440Hzの正弦波を作成
3 Y = fft(y); //フーリエ変換
4 Y = Y(1: length(y)/ 2 + 1); //フーリエ変換結果の後ろ半分を破棄
5 f = linspace(0, 22050, length(Y)); //周波数軸を作成
6 scf(0);
7 subplot(2,1,1);
8 xlabel('t');
9 ylabel('y');
10 plot(y1);

```

```

11
12 subplot(2,1,2);
13 xlabel('f');
14 ylabel('Y');
15 plot(f, abs(Y)); //フーリエ変換結果を図示
16 sound(y, 44100);
17 xs2png(0,'reidai6.png'); //グラフをpngで保存

```

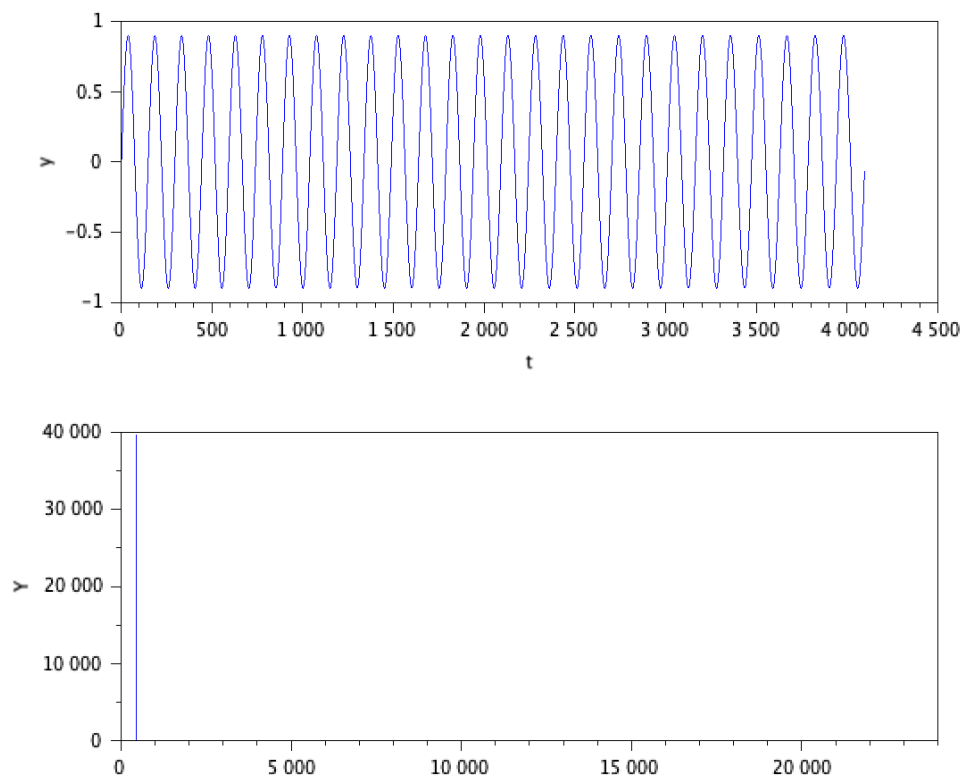


図 2: 例題 6 の出力波形

図 2 からわかるように、440 Hz の波形をフーリエ変換すると、周波数軸が 440 Hz のところに一本だけ線がでるグラフが出力される

5.3 例題 7

ここで作成したソースコードを以下の Code3 に示す。また、作成したグラフを図 3 に示す。

Code 3: 例題 7 のソースコード

```

1 [y, fs, bits] = wavread('enshu4.wav'); //enshu4.wavを読み込む
2 y1 = y(1 * fs + (0 : 4095)); //開始から 1 秒後の4096点を取り出す
3 Y1 = fft(y1); //フーリエ変換
4 Y1 = Y1(1 : length(y1)/ 2+1); //フーリエ変換結果の後ろ半分を破棄
5 f = linspace(0, fs/2, length(Y1)); //周波数軸を作成
6 scf(0);
7 subplot(2,1,1);
8 xlabel('t');
9 ylabel('y');
10 plot(y1);
11
12 subplot(2,1,2);
13 xlabel('f');
14 ylabel('Y');
15 plot(f, abs(Y1)); //フーリエ変換結果を図示
16 wavwrite(y, 44100, 16, 'test7.wav');
17 sound(y1, 44100); //音を出す

```

```
18 xs2png(0, 'reidai7.png');
```

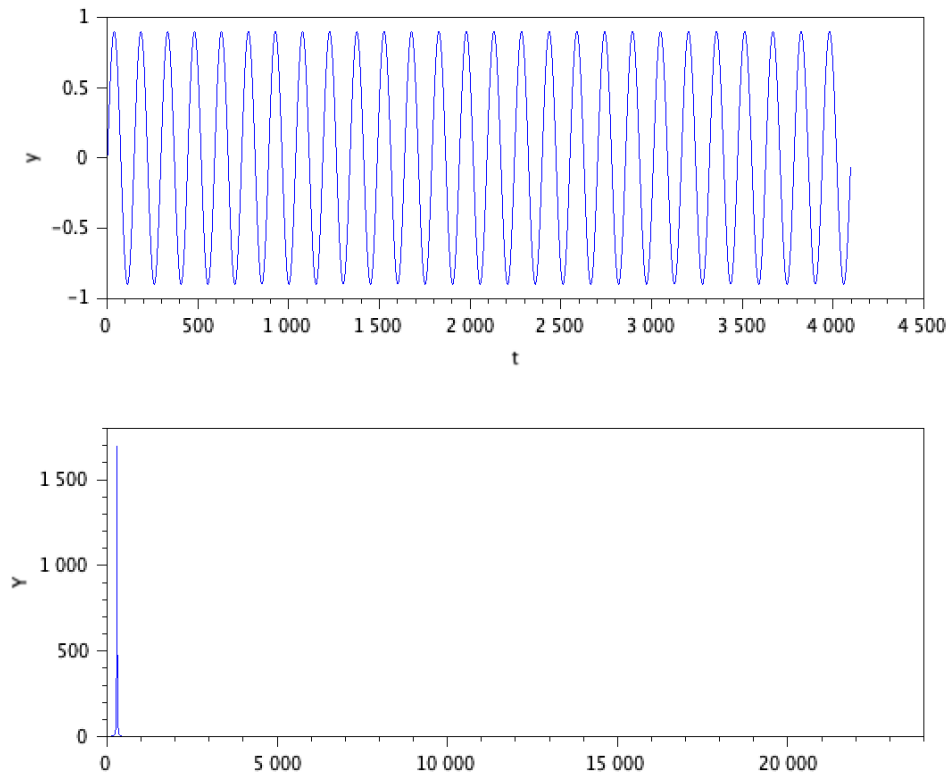


図 3: 例題 7 の出力波形

例題 7 ではこれまでの信号とは違い、非定常信号に適用できる方法でフーリエ変換を行っている。これを短時間フーリエ変換といい、非定常信号から取り出した一部が無限に繰り返されていると仮定してフーリエ変換を行っている。

5.4 例題 8

ここで作成したソースコードを以下の Code4 に示す。また、作成したグラフを図 4 に示す。

Code 4: 例題 8 のソースコード

```
1 [y, fs, bits] = wavread('enshu4.wav'); //enshu4.wavを読み込む
2 y1 = y(1 * fs + (0 : 4095)); //開始から1秒後の4096点を取り出す
3 w = window('hn', size(y1,2)); //ハニング窓の作成
4 y1 = w.*y1; //作成したハニング窓をかける
5 Y1 = fft(y1); //フーリエ変換
6 Y1 = Y1(1 : length(y1)/ 2+1); //フーリエ変換結果の後半分を破棄
7 f = linspace(0, fs/2, length(Y1)); //周波数軸を作成
8 scf(0);
9 subplot(2,1,1);
10 xlabel('t');
11 ylabel('y');
12 plot(y1);
13
14 subplot(2,1,2);
15 xlabel('f');
16 ylabel('Y');
17 plot(f, abs(Y1)); //フーリエ変換結果を図示
18 xs2png(0, 'reidai8.png');
```

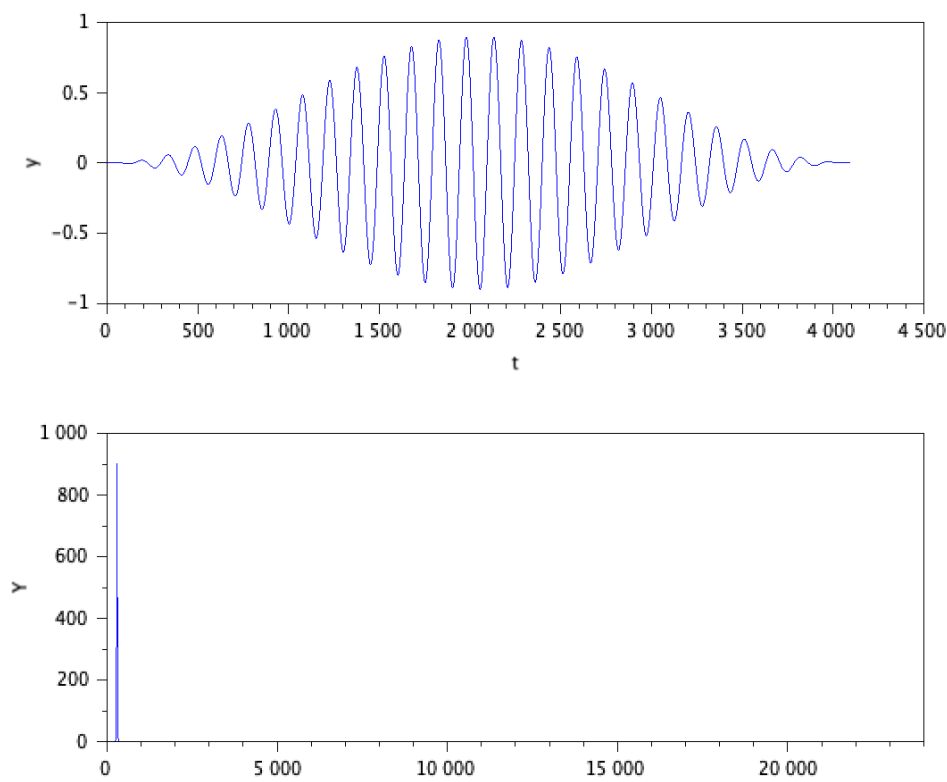


図 4: 例題 8 の出力波形

図 4 のフーリエ変換後のグラフより、窓関数をかけることによりその範囲だけ切り取ることができていることがわかる。

5.5 演習 9

h の長さを長くして信号 sn に畳み込むと、 f_{sn} はどうなるかを試せ。

ここで作成したソースコードを以下の Code5 に示す。また、作成したグラフを図 5,6 に示す。

Code 5: 課題 9 のソースコード

```

1 n = 1 : 100;
2 Fs = 8000;
3 t = 0 : 1/Fs : 1-1/Fs;
4 y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成
5
6 s = sin(2 * %pi * 440 * t);
7 y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
8 scf(0);
9
10 h = ones(1, 3)/3 //フィルタの生成
11 y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形
12
13 subplot(2,1,1);
14 xlabel('n');
15 ylabel('y_nz,y_sn,y_fsn');
16 plot(n, y_nz(n),'-r');
17 plot(n, y_sn(n),'-g');
18 plot(n, y_fsn(n),'-b');
19
20
21 w_nz = window('hn', size(y_nz,2)); //窓関数の作成
22 w_sn = window('hn', size(y_sn,2)); //窓関数の作成
23 w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成

```

```

24 yw_nz = w_nz.*y_nz;
25 yw_sn = w_sn.*y_sn;
26 yw_fsn = w_fsn.*y_fsn; //窓関数を適用
27
28 Y_nz = fft(yw_nz);
29 Y_sn = fft(yw_sn);
30 Y_fsn = fft(yw_fsn); //フーリエ変換
31
32 Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
33 Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
34 Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除
35
36 f_nz = linspace(0, 22050, length(Y_nz));
37 f_sn = linspace(0, 22050, length(Y_sn));
38 f_fsn = linspace(0, 22050, length(Y_fsn)); //周波数軸の作成
39
40 subplot(2,1,2);
41 xlabel('f');
42 ylabel('Y_nz,Y_sn,Y_fsn');
43 plot(f_nz, abs(Y_nz),'-r'); //ノイズのフーリエ変換
44 plot(f_sn, abs(Y_sn),'-g'); //ノイズ付きsin波形のフーリエ変換
45 plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換
46
47
48
49 scf(1);
50 h = ones(1, 3)/5 //フィルタの生成
51 y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形
52
53 subplot(2,1,1);
54 xlabel('n');
55 ylabel('y_nz,y_sn,y_fsn');
56 plot(n, y_nz(n),'-r');
57 plot(n, y_sn(n),'-g');
58 plot(n, y_fsn(n),'-b');
59
60
61 w_nz = window('hn', size(y_nz,2)); //窓関数の作成
62 w_sn = window('hn', size(y_sn,2)); //窓関数の作成
63 w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
64 yw_nz = w_nz.*y_nz;
65 yw_sn = w_sn.*y_sn;
66 yw_fsn = w_fsn.*y_fsn; //窓関数を適用
67
68 Y_nz = fft(yw_nz);
69 Y_sn = fft(yw_sn);
70 Y_fsn = fft(yw_fsn); //フーリエ変換
71
72 Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
73 Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
74 Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除
75
76 f_nz = linspace(0, 22050, length(Y_nz));
77 f_sn = linspace(0, 22050, length(Y_sn));
78 f_fsn = linspace(0, 22050, length(Y_fsn)); //周波数軸の作成
79
80 subplot(2,1,2);

```

```

81 xlabel('f');
82 ylabel('Y_nz,Y_sn,Y_fsn');
83 plot(f_nz, abs(Y_nz),'-r'); //ノイズのフーリエ変換
84 plot(f_sn, abs(Y_sn),'-g'); //ノイズ付きsin波形のフーリエ変換
85 plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換
86
87 xs2png(0, 'kadai9_h3.png');
88 xs2png(1, 'kadai9_h5.png');

```

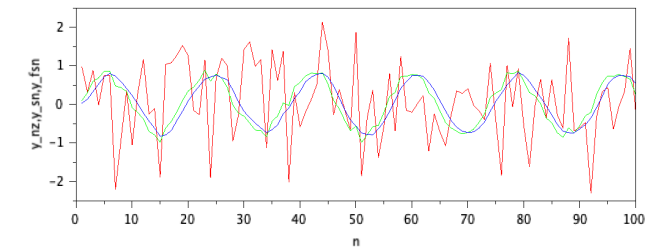


図 5: $h = 0.3333$ の場合

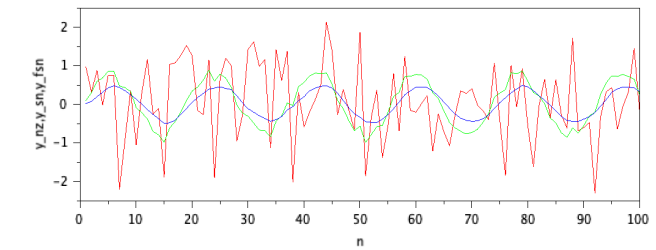


図 6: $h = 0.2$ の場合

ここでは、 h の値を変化させ、フィルターによる波形への影響を確認した。 $h = 0.33 \dots$ のときと $h = 0.2$ のときを比べると、わずかに $h = 0.33 \dots$ のほうが sin 波形がなめらかになった。その代わり、振幅が多少小さくなることがわかった。

5.6 演習 13

wfir を用いて、550 Hz 以下の低減だけを通過させる **LPF** を設計し、その **LPF** を s_n に適用せよ。

ここで作成したソースコードを以下の Code6 に示す。また、作成したグラフを図 7 に示す。

Code 6: 演習 13 のソースコード

```

1 n = 1 : 100;
2 Fs = 8000;
3 t = 0 : 1/Fs : 1-1/Fs;
4 y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成
5
6 s = sin(2 * %pi * 440 * t);
7 y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
8 scf(0);
9
10 h = wfir('lp', 40, 0.1375, 'hm', [0 0]); //フィルタの生成
11 y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形
12
13 subplot(2,1,1);
14 xlabel('n');
15 ylabel('y_nz,y_sn,y_fsn');
16 plot(n, y_fsn(n),'-b');
17
18

```

```

19 w_nz = window('hn', size(y_nz,2)); //窓関数の作成
20 w_sn = window('hn', size(y_sn,2)); //窓関数の作成
21 w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
22 yw_nz = w_nz.*y_nz;
23 yw_sn = w_sn.*y_sn;
24 yw_fsn = w_fsn.*y_fsn; //窓関数を適用
25
26 Y_nz = fft(yw_nz);
27 Y_sn = fft(yw_sn);
28 Y_fsn = fft(yw_fsn); //フーリエ変換
29
30 Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
31 Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
32 Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除
33
34 f_nz = linspace(0, 4000, length(Y_nz));
35 f_sn = linspace(0, 4000, length(Y_sn));
36 f_fsn = linspace(0, 4000, length(Y_fsn)); //周波数軸の作成
37
38 subplot(2,1,2);
39 xlabel('f');
40 ylabel('Y_nz,Y_sn,Y_fsn');
41 plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換
42
43 xs2png(0, 'kadai13.png')

```

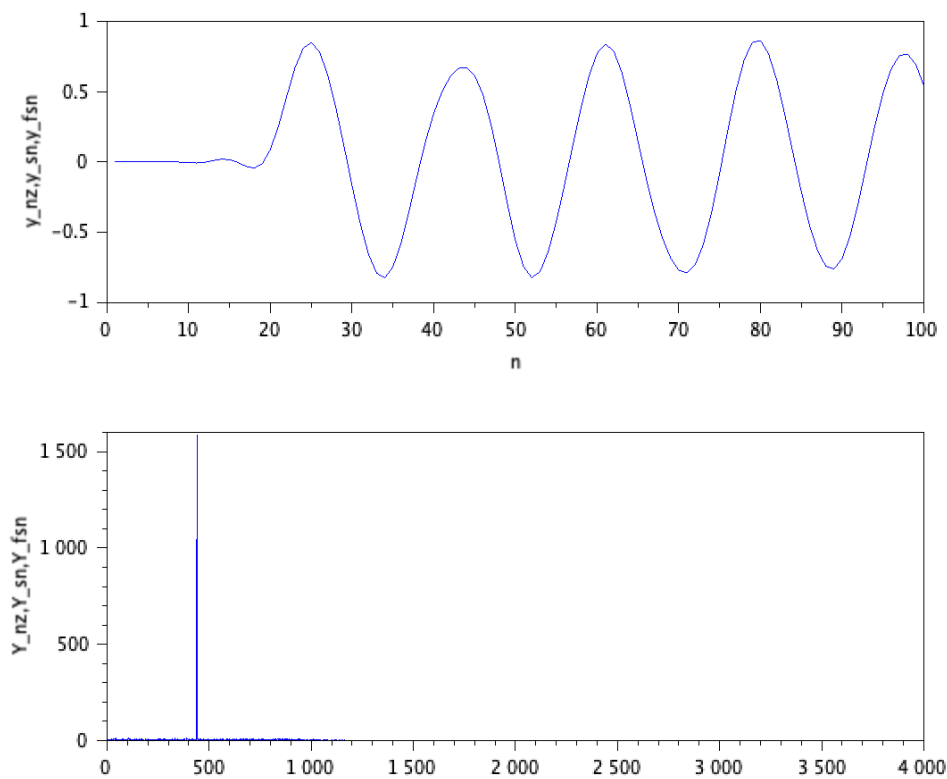


図 7: 演習 13 の出力波形

図 7 からわかるように、550 Hz 以下のみが出力されており、それ以上はカットされている。よってローパスフィルタが正常に動作していることがわかる。

5.7 演習 14

wfir を用いて、ある周波数以上の広域を通過させるフィルタ (ハイパスフィルタ, **HPF**) も設計できる. 450 Hz 以上を通過させる **HPF** を設計し、その **HPF** を s_n に適用せよ.

ここで作成したソースコードを以下の Code7 に示す. また、作成したグラフを図 8 に示す.

Code 7: 演習 14 のソースコード

```
1 n = 1 : 100;
2 Fs = 8000;
3 t = 0 : 1/Fs : 1-1/Fs;
4 y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成
5
6 s = sin(2 * %pi * 440 * t);
7 y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
8 scf(0);
9
10 h = wfir('hp', 43, 0.1125, 'hm', [0 0]); //フィルタの生成
11 y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形
12
13 subplot(2,1,1);
14 xlabel('n');
15 ylabel('y_nz,y_sn,y_fsn');
16
17 plot(n, y_fsn(n), '-b');
18
19
20 w_nz = window('hn', size(y_nz,2)); //窓関数の作成
21 w_sn = window('hn', size(y_sn,2)); //窓関数の作成
22 w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
23 yw_nz = w_nz.*y_nz;
24 yw_sn = w_sn.*y_sn;
25 yw_fsn = w_fsn.*y_fsn; //窓関数を適用
26
27 Y_nz = fft(yw_nz);
28 Y_sn = fft(yw_sn);
29 Y_fsn = fft(yw_fsn); //フーリエ変換
30
31 Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
32 Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
33 Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除
34
35 f_nz = linspace(0, 4000, length(Y_nz));
36 f_sn = linspace(0, 4000, length(Y_sn));
37 f_fsn = linspace(0, 4000, length(Y_fsn)); //周波数軸の作成
38
39 subplot(2,1,2);
40 xlabel('f');
41 ylabel('Y_fsn');
42
43 plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換
44
45 xs2png(0, 'kadai14.png')
```

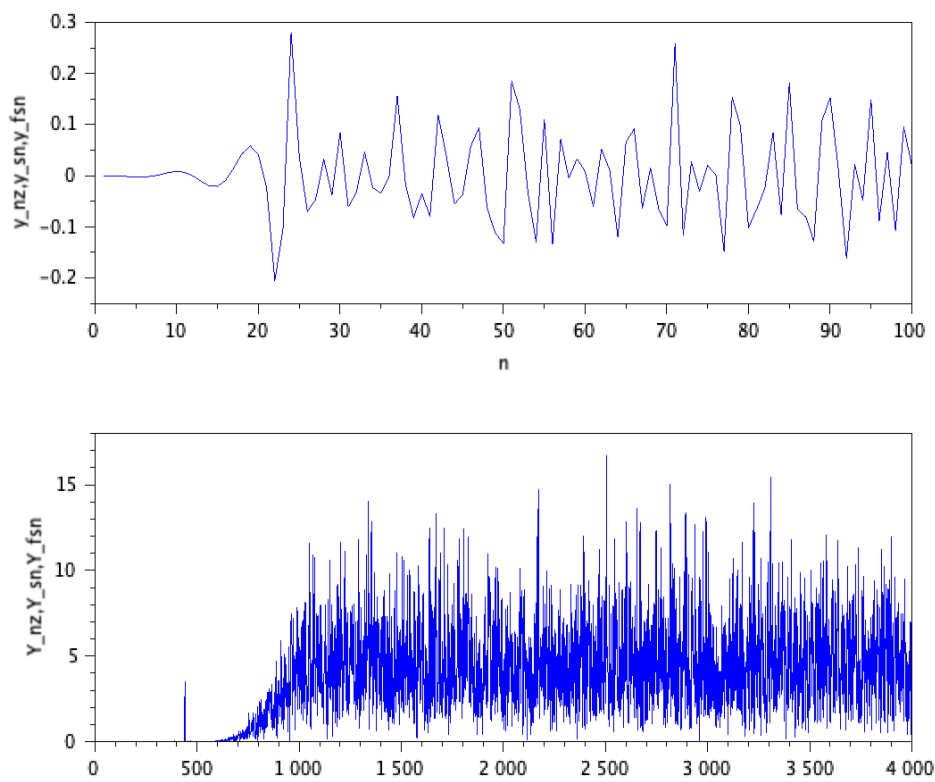


図 8: 演習 14 の出力波形

図 8 からわかるように、450 Hz 以上の周波数だけが出力されていることがわかる。よってハイパスフィルタが正常に動作していることがわかる。

6 考察

参考文献

- [1] 野尻 紘聖: 信号処理, 制御情報システム工学実験 2 実験指導書 (最終閲覧日 2021 年 5 月 25 日)
- [2] 嵯峨山 茂樹: 音声解析 (1) サンプリングと量子化, 東京大学 工学部 計数工学科/物理工学科 (最終閲覧日 2021 年 5 月 25 日)
https://ocw.u-tokyo.ac.jp/lecture_files/engin_01/2/notes/ja/B1-Sampling.pdf
- [3] ロジカルアーツ研究所: 窓関数を用いる理由 (最終閲覧日 2021 年 5 月 26 日)
<https://www.logical-arts.jp/archives/124>