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[Hz] 以上の周波数成分を持たない帯域制限信号 $f(t)$ は，$\frac{1}{2}f_m$[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 [3] 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)[4] 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．

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

畳み込みを用いることにより，システムの時間経過による状態の変化を導き出す事ができるようになる．

4.4　データの円滑化

入力信号の多くはきれいな波形ではなくノイズが加わっている場合が多い．このとき，ノイズなどの影響を除去の作業が必要となる．ノイズを除去するために用いられる円滑化手法はいくつか存在するが，ここでは単純移動平均について述べる．単純移動平均とは，系列データを，ある一定区画ごとの平均値を区間をずらしながら求

める手法である．式 1 は 3 点移動平均，式 2 は 5 点移動平均の定義式である．式中の DS_t は円滑化後のデータ，t は区画を示している．また，これらの式を使って円滑化した場合の様子を図 1,2 に示す．

$$DS_t = \frac{1}{4}(D_{t-1} + 2D_t + D_{t+1}) \tag{1}$$

$$DS_t = \frac{1}{16}(D_{t-2} + 4D_{t-1} + 6D_t + 4D_{t+1} + D_{t+2}) \tag{2}$$

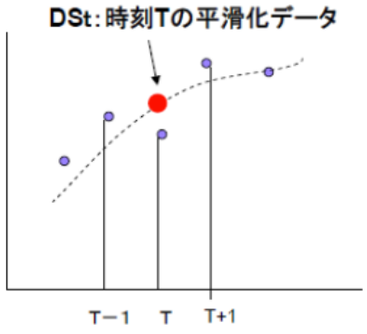

図 1: 3 点平均移動

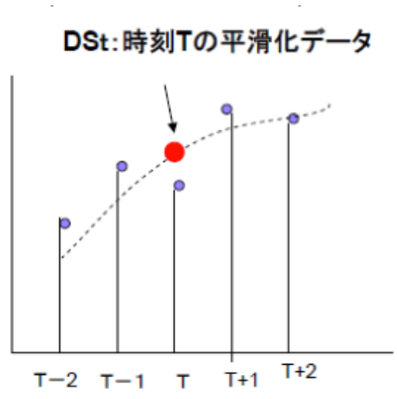

図 2: 5 点平均移動

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 3 に示す．

Code 1: 演習 4 のソースコード

```
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
```

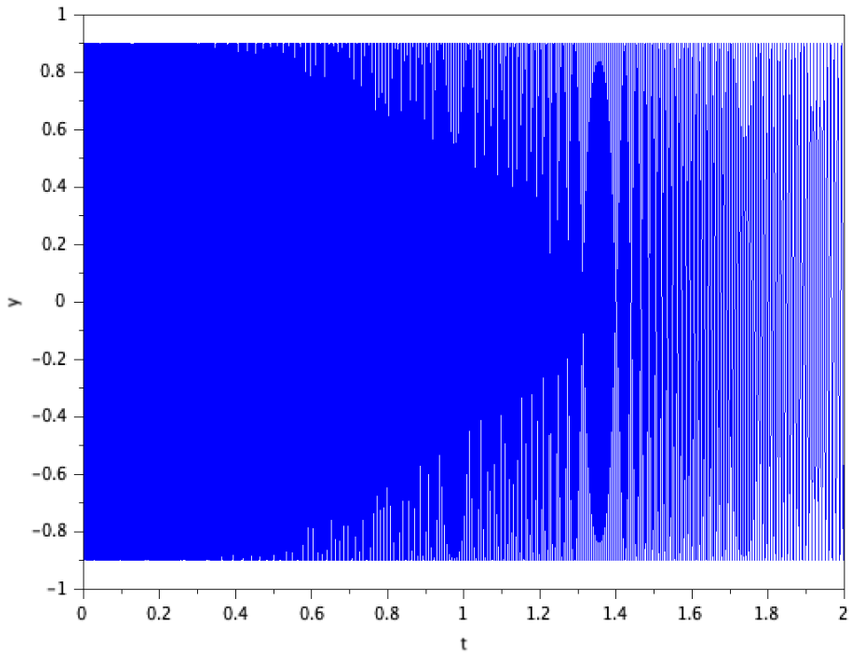

図 3: 演習 4 の出力波形

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 3 からも時間が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 4 に示す．

Code 2: 例題 6 のソースコード

```
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
```

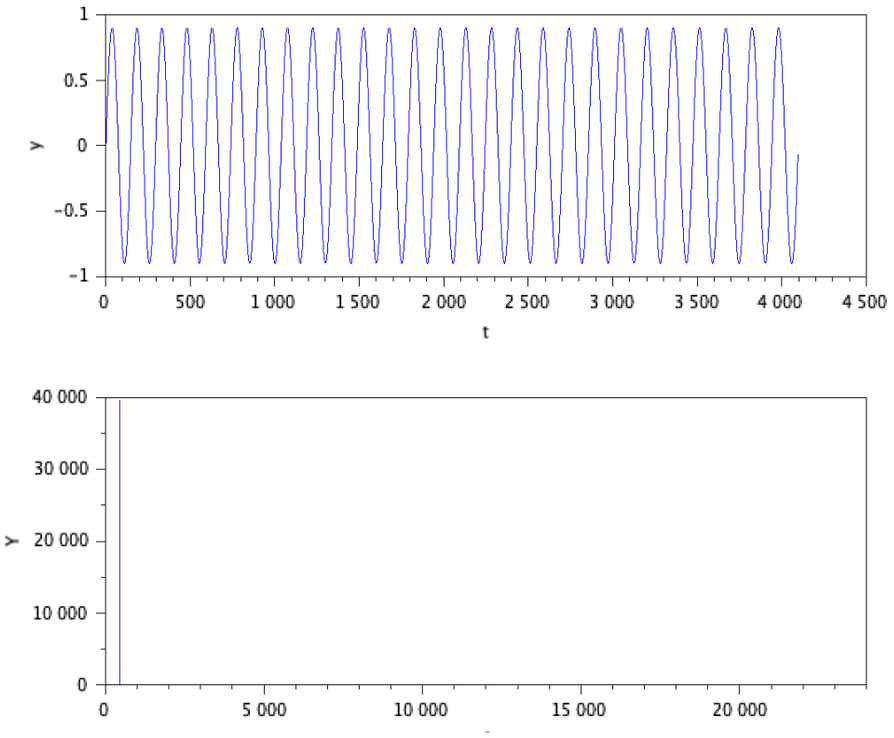

図 4: 例題 6 の出力波形

図 4 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 5 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');
```

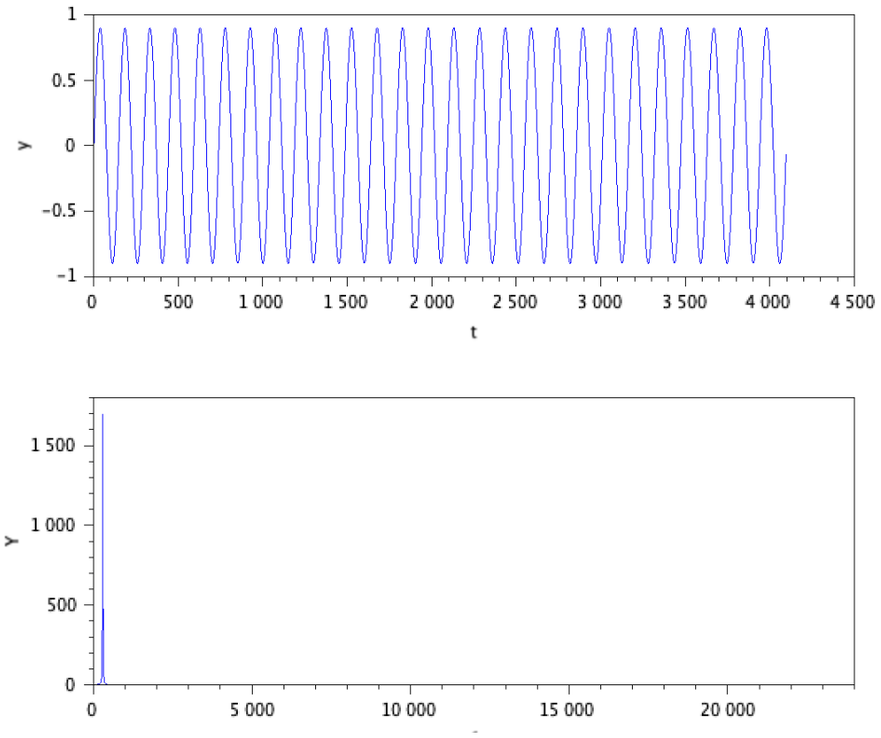

図 5: 例題 7 の出力波形

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 6 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');
```

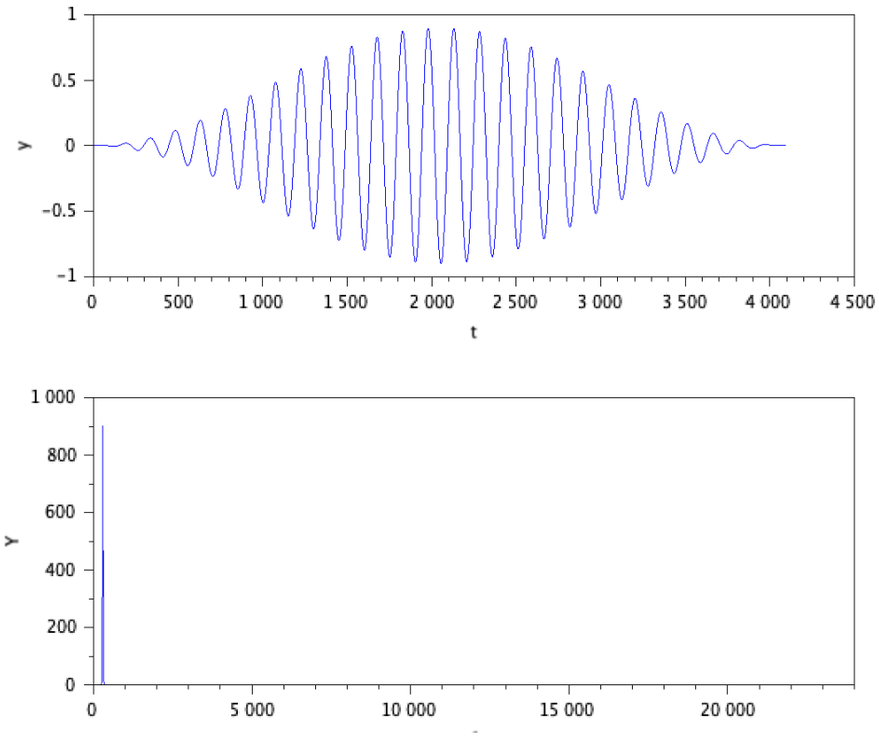

図 6: 例題 8 の出力波形

図 6 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こ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，fsn 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 7,8 に示す．

Code 5: 課題 9 のソースコード

```
n = 1 : 100;
Fs = 8000;
t = 0 : 1/Fs : 1-1/Fs;
y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成

s = sin(2 * %pi * 440 * t);
y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
scf(0);

h = ones(1, 3)/3 //フィルタの生成
y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形

subplot(2,1,1);
xlabel('n');
ylabel('y_nz,y_sn,y_fsn');
plot(n, y_nz(n),'-r');
plot(n, y_sn(n),'-g');
plot(n, y_fsn(n),'-b');


w_nz = window('hn', size(y_nz,2)); //窓関数の作成
w_sn = window('hn', size(y_sn,2)); //窓関数の作成
w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
```

```
yw_nz = w_nz.*y_nz;
yw_sn = w_sn.*y_sn;
yw_fsn = w_fsn.*y_fsn; //窓関数を適用

Y_nz = fft(yw_nz);
Y_sn = fft(yw_sn);
Y_fsn = fft(yw_fsn); //フーリエ変換

Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除

f_nz = linspace(0, 22050, length(Y_nz));
f_sn = linspace(0, 22050, length(Y_sn));
f_fsn = linspace(0, 22050, length(Y_fsn)); //周波数軸の作成

subplot(2,1,2);
xlabel('f');
ylabel('Y_nz,Y_sn,Y_fsn');
plot(f_nz, abs(Y_nz),'-r'); //ノイズのフーリエ変換
plot(f_sn, abs(Y_sn),'-g'); //ノイズ付きsin波形のフーリエ変換
plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換



scf(1);
h = ones(1, 3)/5 //フィルタの生成
y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形

subplot(2,1,1);
xlabel('n');
ylabel('y_nz,y_sn,y_fsn');
plot(n, y_nz(n),'-r');
plot(n, y_sn(n),'-g');
plot(n, y_fsn(n),'-b');


w_nz = window('hn', size(y_nz,2)); //窓関数の作成
w_sn = window('hn', size(y_sn,2)); //窓関数の作成
w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
yw_nz = w_nz.*y_nz;
yw_sn = w_sn.*y_sn;
yw_fsn = w_fsn.*y_fsn; //窓関数を適用

Y_nz = fft(yw_nz);
Y_sn = fft(yw_sn);
Y_fsn = fft(yw_fsn); //フーリエ変換

Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除

f_nz = linspace(0, 22050, length(Y_nz));
f_sn = linspace(0, 22050, length(Y_sn));
f_fsn = linspace(0, 22050, length(Y_fsn)); //周波数軸の作成

subplot(2,1,2);
```

```
xlabel('f');
ylabel('Y_nz,Y_sn,Y_fsn');
plot(f_nz, abs(Y_nz),'-r'); //ノイズのフーリエ変換
plot(f_sn, abs(Y_sn),'-g'); //ノイズ付きsin波形のフーリエ変換
plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換

xs2png(0, 'kadai9_h3.png');
xs2png(1, 'kadai9_h5.png');
```

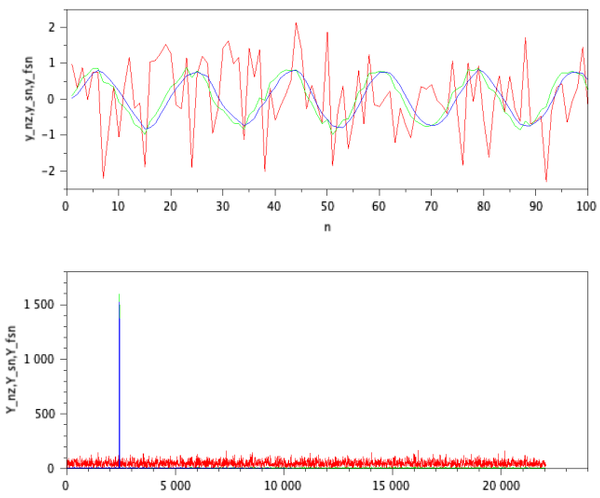

図 7: $h = 0.3333$ の場合

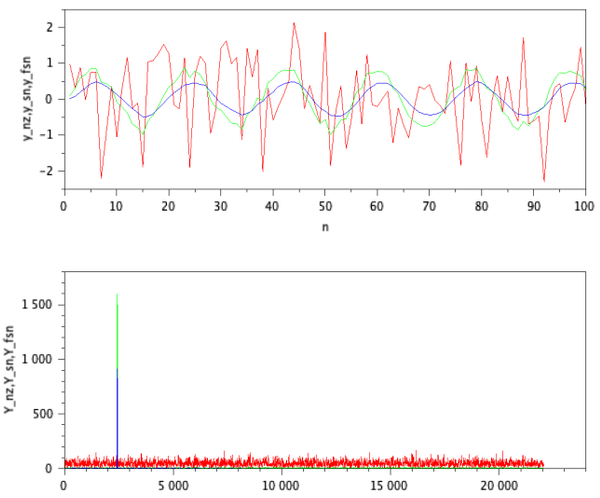

図 8: $h = 0.2$ の場合

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．$h = 0.33\cdots$ のときと $h = 0.2$ のときを比べると，わずかに $h = 0.33\cdots$ 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 9 に示す．

Code 6: 演習 13 のソースコード

```
n = 1 : 100;
Fs = 8000;
t = 0 : 1/Fs : 1-1/Fs;
y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成

s = sin(2 * %pi * 440 * t);
y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
scf(0);

h = wfir('lp', 40, 0.1375, 'hm', [0 0]); //フィルタの生成
y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形

subplot(2,1,1);
xlabel('n');
ylabel('y_nz,y_sn,y_fsn');
plot(n, y_fsn(n),'-b');


```

```
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')
```

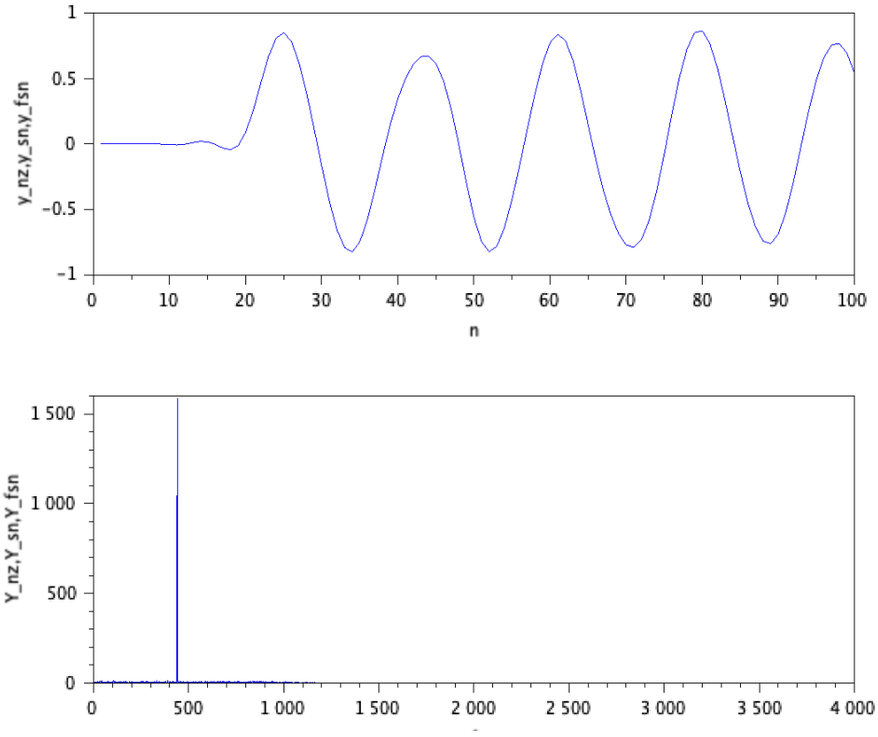

図 9: 演習 13 の出力波形

図 9 からもわかるように，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 10 に示す．

Code 7: 演習 14 のソースコード

```
n = 1 : 100;
Fs = 8000;
t = 0 : 1/Fs : 1-1/Fs;
y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成

s = sin(2 * %pi * 440 * t);
y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
scf(0);

h = wfir('hp', 43, 0.1125, 'hm', [0 0]); //フィルタの生成
y_fsn = convol(y_sn, h); //フィルタを畳み込んだsin波形

subplot(2,1,1);
xlabel('n');
ylabel('y_nz,y_sn,y_fsn');

plot(n, y_fsn(n),'-b');


w_nz = window('hn', size(y_nz,2)); //窓関数の作成
w_sn = window('hn', size(y_sn,2)); //窓関数の作成
w_fsn = window('hn', size(y_fsn,2)); //窓関数の作成
yw_nz = w_nz.*y_nz;
yw_sn = w_sn.*y_sn;
yw_fsn = w_fsn.*y_fsn; //窓関数を適用

Y_nz = fft(yw_nz);
Y_sn = fft(yw_sn);
Y_fsn = fft(yw_fsn); //フーリエ変換

Y_nz = Y_nz(1 : length(yw_nz) / 2 + 1);
Y_sn = Y_sn(1 : length(yw_sn) / 2 + 1);
Y_fsn = Y_fsn(1 : length(yw_fsn) / 2 + 1); //フーリエ変換結果の後半を削除

f_nz = linspace(0, 4000, length(Y_nz));
f_sn = linspace(0, 4000, length(Y_sn));
f_fsn = linspace(0, 4000, length(Y_fsn)); //周波数軸の作成

subplot(2,1,2);
xlabel('f');
ylabel('Y_fsn');

plot(f_fsn,abs(Y_fsn),'-b'); //フィルター付きsin波形のフーリエ変換

xs2png(0, 'kadai14.png')
```

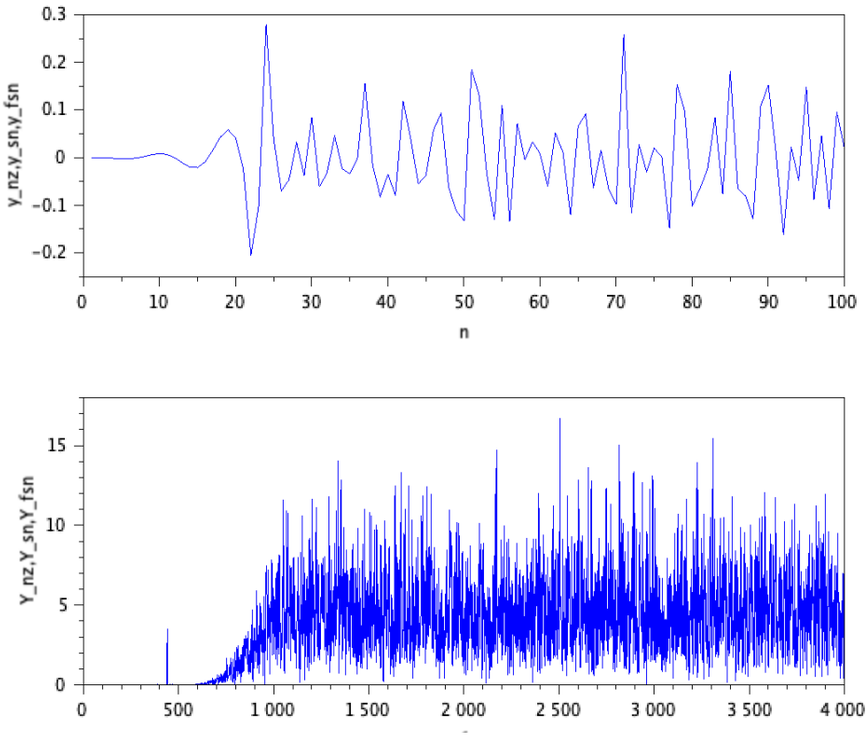

図 10: 演習 14 の出力波形

図 10 からもわかるように，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．

5.8 研究課題

3 点および 5 点の移動平均を行うプログラムを Scilab の sce-ファイルで作成し，フィルタの前後の信号を同時にプロットしてそれらの効果を確認せよ．ただし，読込む信号は雑音交じりの信号 *sn* とする．

以下に作成したソースコードを以下の Code8,9 に示す．また，その実行結果を以下の図 12,13 に示す．ただし，ノイズの作成はこれまでの実験で使用したノイズを使用し，ノイズ混じりの円滑化前のグラフは図**??**に示す．

Code 8: 3 点平均移動のソースコード

```
1  n = 1 : 100;
2  Fs = 8000;
3  t = 0 : 1/Fs : 1-1/Fs;
4  DS=[0];
5  y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成
6
7  s = sin(2 * %pi * 440 * t);
8  y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
9  Y_sn = fft(y_sn);
10 Y_sn = Y_sn(1:length(y_sn)/2 + 1);
11 f = linspace(0, 4000, length(Y_sn));
12
13 scf(0);
14
15 plot2d(n, y_sn(n));
16
17 for i = 2 : 1: 100
18    DS(i) = 0.25 * (y_sn(i-1) + 2*y_sn(i) + y_sn(i+1));
19 end
20
21 scf(1);
```

```
22 plot2d(n,DS);
23
24 xs2png(1, 'kenkyu1.png')
```

Code 9: 5 点平均移動のソースコード

```
1  n = 1 : 100;
2  Fs = 8000;
3  t = 0 : 1/Fs : 1-1/Fs;
4  DS=[0];
5  y_nz = grand(1, length(t), 'nor', 0, 1); //ノイズの作成
6
7  s = sin(2 * %pi * 440 * t);
8  y_sn = 0.8 * s + 0.1 * y_nz; //ノイズが混じったsin波形の作成
9  Y_sn = fft(y_sn);
10 Y_sn = Y_sn(1:length(y_sn)/2 + 1);
11 f = linspace(0, 4000, length(Y_sn));
12
13 scf(0);
14
15 plot2d(n, y_sn(n));
16
17 for i = 3 : 1: 100
18    DS(i) = 0.0625 * (y_sn(i-2) + 4*y_sn(i-1) + 6*y_sn(i) + 4*y_sn(i+1) + y_sn(i+2));
19 end
20
21 scf(1);
22 plot2d(n,DS);
23
24 xs2png(0, 'noiz.png');
25 xs2png(1, 'kenkyu2.png')
```

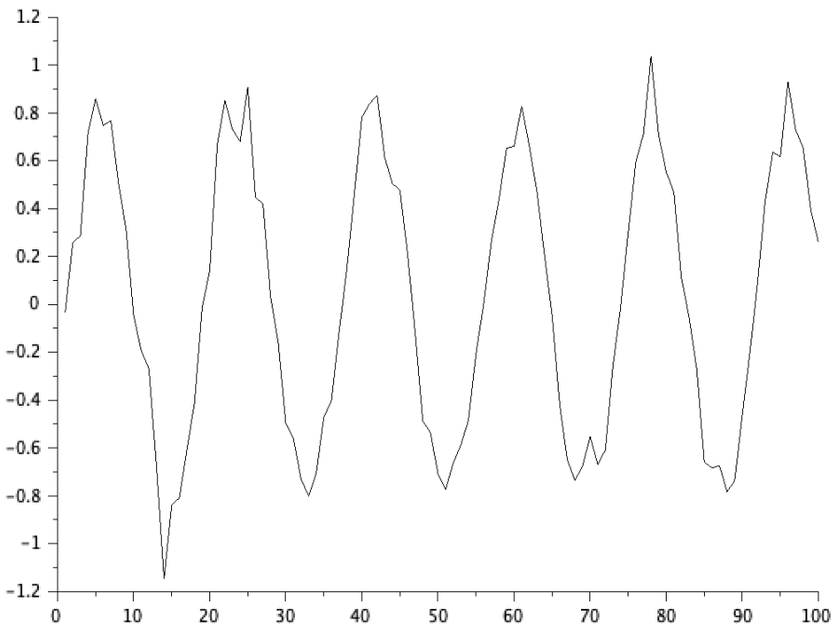

図 11: ノイズが加わっている sin 波形

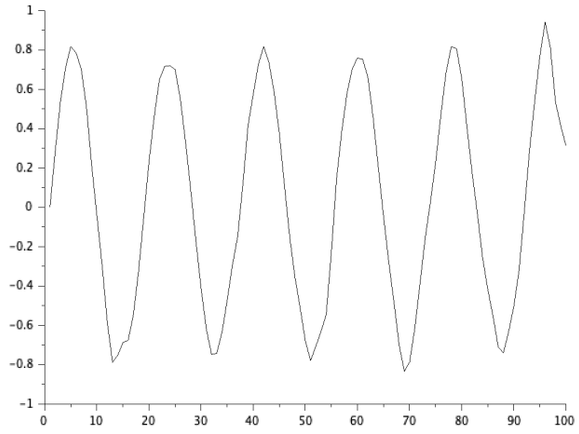

図 12: 3 点平均移動のグラフ

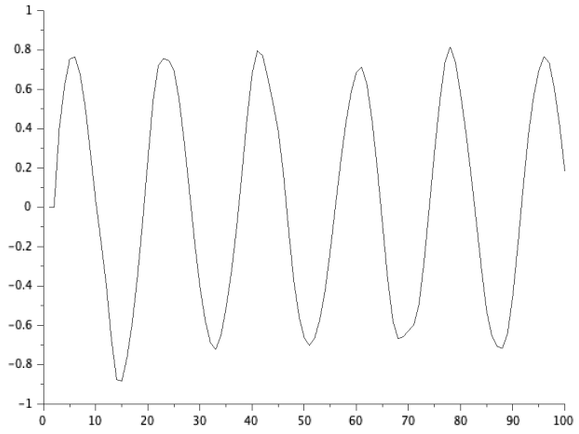

図 13: 5 点平均移動のグラフ

6 考察

今回の実験で音の大きさ (振幅) や音の高さ (周波数) をグラフ化することができるようになった．そのことを応用し，自分の声を録音し，それをグラフ化することを本実験の考察とする．音声はブラウザ上で wav 形式の録音が可能な「音声録音くん」[5] を使用して，「あーーー」という声を高い声，低い声それぞれ録音した．以下の Code**??**に低い声の場合のソースコード，Code**??**に高い声の場合のソースコード，図 14,15 にそれぞれ低い声と高い声の場合のグラフを示す．

Code 10: 低い声の場合のソースコード

```
[y, fs, bits] = wavread('voice1.wav'); //voice1.wavを読み込む
y1 = y(8 * fs + (0 : 50000)); //開始から8秒後の50000点を取り出す
Y1 = fft(y1); //フーリエ変換
Y1 = Y1(1 : length(y1)/ 2+1); //フーリエ変換結果の後ろ半分を破棄
f = linspace(0, fs/2, length(Y1)); //周波数軸を作成
scf(0);
subplot(2,1,1);
xlabel('t');
ylabel('y');
plot(y1);

subplot(2,1,2);
xlabel('f');
ylabel('Y');
plot(f, abs(Y1)); //フーリエ変換結果を図示
//wavwrite(y, 44100, 16, 'test7.wav');
sound(y1, 44100); //音を出す
xs2png(0, 'ex_kadai1.png');
```

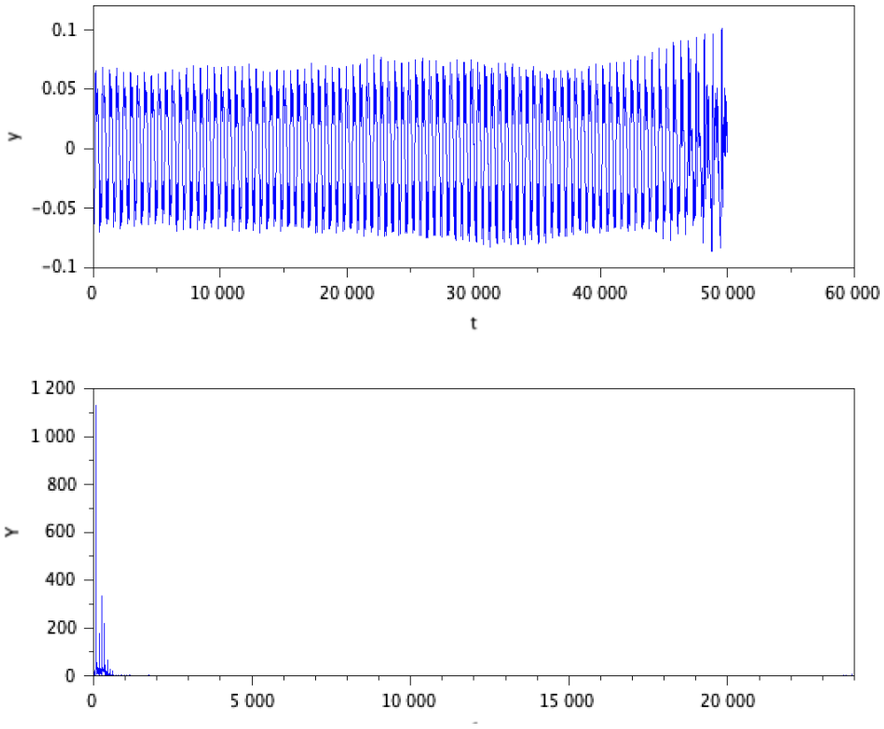

図 14: 低い声場合のグラフ

Code 11: 高い声の場合のソースコード

```
1  [y, fs, bits] = wavread('voice2.wav'); //voice2.wavを読み込む
2  y1 = y(3 * fs + (0 : 50000)); //開始から3秒後の50000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, 'ex_kadai2.png');
```

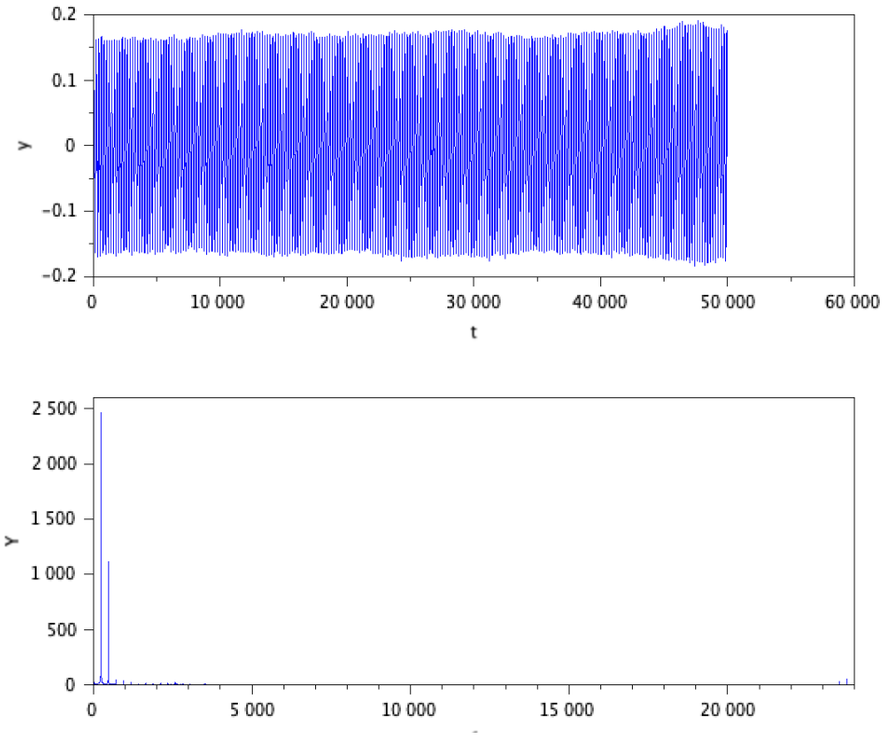

図 15: 高い声場合のグラフ

図 14,15 からわかるように，声の高さが変わることによって時間軸上では波の密度が変化し，周波数軸ではスペクトルの位置が少し変化していることがわかる．Scilab で音声解析を行うことによって，音の情報を視覚化することができ，例え音声を聞かなくとも音について理解することができる．これらの技術を用いることで，システムの中で音声を処理し，その他のところへ用いることができるようになるため，とても便利であり重要な技術であることがわかった．

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

[4] 中川朋子: たたみこみ（合成積）, 東北工業大学 情報通信工学科 中川研究室 (最終閲覧日 2021 年 5 月 28 日)
https://www.ice.tohtech.ac.jp/nakagawa/laplacetrans/convolution1.htm

[5] 音声録音: 音声録音くん, NCH software (最終閲覧日 2021 年 5 月 28 日)
https://www.petitmonte.com/labo/voice-recording/