制御情報システム工学科 4 年　制御情報システム工学実験Ⅱ
計測グループ実験課題③

信号解析

作成者：野尻　紘聖

1. 目的

RF 用のフィルタ，スイッチング電源用のライン・フィルタなど，その仕事の性質上さまざまなフィルタが存在する．「さまざまな電子回路に流れる信号」，「音声・音響信号」や「画像信号」，「筋電位・心電・脈波・脳波などの生体信号」および「加速度・ジャイロなどのセンサから得たデータ」の解析や処理を行い，電子デバイスの設計や機器の計測と制御などに応用するためには，信号のフィルタリングの技術が重要である．

そこで，本実験では，昨年度実施の「汎用数値計算ソフトウェアの基礎」の実験で使用した Scilab を使用し，さらに汎用数値計算ソフトウェアについてマスターするべく，主に信号処理に関する関数の使用法を習得することが目的のひとつである．取り扱う信号は，1 次元信号の代表例であり，学生にも興味関心が高いと思われる「音声信号」について，簡単な信号生成および，その信号の解析手法について学ぶ．

2. 実験

2.1　正弦波

定常的な信号は，基本的に正弦波の合成により作られる．基本周波数をf，振幅をA とすると，次の式で表される．

$$y = A\sin(2\pi f t). \tag{1}$$

例題 1）440Hz（中央のラ）の音を正弦波で作ってみよう．

・Scilab の場合

```
--> t = 0 : 1/44100 : 5;
--> y = 0.9 * sin(2 * %pi * 440 * t);
--> wavwrite(y, 44100, 16, 'test1.wav');
--> sound(y, 44100);
```

・MATLAB の場合

```
>> t = 0 : 1/44100 : 5;
>> y = 0.9 * sin(2 * pi * 440 * t);
>> wavwrite(y, 44100, 16, 'test1.wav');
>> soundsc(y, 44100);
```

このような正弦波のみによって作られる音は，**純音**と呼ばれる．

演習 1）262Hz（中央のド）の音を正弦波で作ってみよう．

2.2　正弦波の足し合わせによる複合音の合成

基本周波数の整数倍の正弦波を足し合わせると，1 つの音に聞こえる性質がある．

例題２）440Hz の音に 2 次倍音，3 次倍音を加えてみよう．

・Scilab の場合

```
--> t = 0 : 1/44100 : 5;
--> y = 0.4 * sin(2 * %pi * 440 * t) + 0.3 * sin(2
* %pi * 880 * t) + ...
   > 0.2 * sin(2 * %pi * 1320 * t);
--> wavwrite(y, 44100, 16, 'test2.wav');
--> sound(y, 44100);
```

・MATLAB の場合

```
>> t = 0 : 1/44100 : 5;
>> y = 0.4 * sin(2 * pi * 440 * t) + 0.3 * sin(2
* %pi * 880 * t) + ...
   0.2 * sin(2 * %pi * 1320 * t);
>> wavwrite(y, 44100, 16, 'test2.wav');
>> soundsc(y, 44100);
```

演習２）任意の基本周波数を持つ音に，10 次までの各倍音を任意の振幅で合成するプログラムを書き，次のことを試せ.

1. 基音に対して高次の倍音の振幅が大きいと音はどのように聞こえるか．逆に小さいとどうか.

2. 偶数次倍音の振幅を 0 にすると，どのような音に聞こえるか.

2.3　音量（振幅）を変化させる

振幅を変化させるには，振幅の時間変化を表すベクトル（包絡あるいはエンベロープという）を用意し，元の音響信号と要素毎の掛け算をすればよい.

例題３）最初の0.02 秒間で最大音量まで音量が大きくなり，その後，徐々に音量が小さくなる音響信号を作成しよう．

・Scilabの場合

```
--> t = 0 : 1/44100 : 2;
--> A1 = linspace(0, 0.99, ceil(0.02 / 44100));
--> A2 = linspace(0.99, 0, floor(length(t) - 0.01 /
44100));
--> A = [A1 A2];
--> y = A.*sin(2 * %pi * 440 * t);
--> wavwrite(y, 44100, 16, 'test3.wav');
--> sound(y, 44100);
```

・MATLABの場合

```
>> t = 0 : 1/44100 : 2;
>> A1 = linspace(0, 0.99, 0.02 / 44100);
>> A2 = linspace(0.99, 0, length(t) - 0.01 /
44100);
>> A = [A1 A2];
>> y = A.*sin(2 * pi * 440 * t);
>> wavwrite(y, 44100, 16, 'test3.wav');
>> soundsc(y, 44100);
```

注：　f * t ではなくf .* t であることに注意.

発音開始から最大音量に達するまでの時間は，**立ち上がり時間**などと呼ばれる．

演習３）振幅の立ち上がり時間やその後の減衰率などをさまざまに変えた音響信号を作り，音がどう変化するか確めよ．

2.4　周波数を変化させる

例題４）

・Scilabの場合

```
--> t = 0 : 1/44100 : 2;
--> f = linspace(440, 880, length(t));
--> y = 0.9 * sin(2 * %pi * f.*t);
--> wavwrite(y, 44100, 16, 'test4.wav');
--> sound(y, 44100);
```

・MATLABの場合

```
>> t = 0 : 1/44100 : 2;
>> f = linspace(440, 880, length(t));
>> y = 0.9 * sin(2 * pi * f.*t);
>> wavwrite(y, 44100, 16, 'test4.wav');
>> soundsc(y, 44100);
```

注f * t ではなくf .* t であることに注意.

演習４）レポート提出課題：以上の技を駆使して，さまざまな音を作ってみよう.

2.5　さまざまなフーリエ変換

2.5.1　フーリエ変換

フーリエ変換（Fourier Transform）とは，本来，無限に続く周期信号の周波数を，正弦波の加算の形で表現する技術である．いま，周期信号を，$f(t)$とすると，$f(t)$を周波数の関数$F(\omega)$を使って

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega t}d\omega \tag{2}$$

と表され，逆に，$F(\omega)$は

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\exp(j\omega t)dt \tag{3}$$

と表される（j：虚数単位）．この式により，与えられた時間領域の信号を周波数領域に変換することをフーリエ変換という.

2.5.2　離散フーリエ変換

上のフーリエ変換を離散化したものを**離散フーリエ変換（Discrete Fourier Transform）**と呼ぶ．離散信号$f(t)$に対する離散フーリエ変換は,

$$Y\left(e^{i\omega}\right) = \sum_{n} y(n)\exp(-j\omega n) \tag{4}$$

と定義される.

2.5.3　高速フーリエ変換

高速フーリエ変換（Fast Fourier Transform; FFT）は，離散フーリエ変換を高速に行うアルゴリズムである．2つに分割する処理を再帰的に繰り返すため，信号の長さが2 のべき乗でなければならない.

ScilabやMATLAB では，高速フーリエ変換の関数が用意されている．これは，信号の長さが2 のべき乗で無い場合には，本来の離散フーリエ変換の定義式にしたがって計算が行われる.

例題５）下記のような純音，もしくは前節で合成した複合音に対してフーリエ変換をしてみよう．

・Scilab の場合（表記の都合上コメント文を改行）

```
--> t = 0 : 1/44100 : 5;
    // 時間軸の作成
--> y = 0.9 * sin(2 * %pi * 440 * t);
    // 440Hz の正弦波を作成
--> Y = fft(y);
    // フーリエ変換
--> f = linspace(0, 44100, length(Y));
// 周波数軸を作成
--> plot(f, abs(Y)); // フーリエ変換結果を図示
--> sound(y, 44100);
```

・MATLAB の場合（表記の都合上コメント文を改行）

```
>> t = 0 : 1/44100 : 5;
    % 時間軸の作成
>> y = 0.9 * sin(2 * pi * 440 * t);
    % 440Hz の正弦波を作成
>> Y = fft(y);
    % フーリエ変換
>> f = linspace(0, 44100, length(Y));
    % 周波数軸を作成
>> plot(f, abs(Y)); % フーリエ変換結果を図示
>> soundsc(y, 44100);
```

フーリエ変換後のデータは，元の信号と同じ長さのベクトルであり，0～fs Hz（fs：サンプリング周波数）に対応している．しかし，<u>シャノンの標本化定理</u>より，再現される周波数成分は0～$fs/2$ Hz のみである．そこで，この区間のみを描画する．

<u>※実験レポート第2章「原理もしくは理論」の中で，「標本化定理（サンプリング定理）」について簡潔にまとめよ．</u>

例題６）レポート提出課題

・Scilab の場合（表記の都合上コメント文を改行）

```
--> t = 0 : 1/44100 : 5;
    // 時間軸の作成
--> y = 0.9 * sin(2 * %pi * 440 * t);
    // 440Hz の正弦波を作成
--> Y = fft(y);
    // フーリエ変換
--> Y = Y(1:length(y) /2 + 1);
    // フーリエ変換結果の後ろ半分を破棄
--> f = linspace(0, 22050, length(Y));
    // 周波数軸を作成
--> plot(f, abs(Y)); // フーリエ変換結果を図示
--> sound(y, 44100);
```

・MATLAB の場合（表記の都合上コメント文を改行）

```
>> t = 0 : 1/44100 : 5;
    % 時間軸の作成
>> y = 0.9 * sin(2 * pi * 440 * t);
    % 440Hz の正弦波を作成
>> Y = fft(y);
    % フーリエ変換
>> Y = Y(1:length(y) /2 + 1);
    % フーリエ変換結果の後ろ半分を破棄
>> f = linspace(0, 22050, length(Y));
    % 周波数軸を作成
>> plot(f, abs(Y)); // フーリエ変換結果を図示
>> soundsc(y, 44100);
```

2.5.4　短時間フーリエ変換

上で述べたフーリエ変換は，原理的に定常信号にしか適用できない．しかし，音を含むほとんどの信号は非定常的であるため，非定常信号に適用できる技術が不可欠である．

そこで，非定常信号の一部を取り出し，取り出した信号は定常である（すなわち，取り出した部分が無限に続く）と仮定し，フーリエ変換を適用する．これを，**短時間フーリエ変換（Short-time Fourier Transform, Short-term Fourier Transform; STFT）**という．

例題７）レポート提出課題

・Scilabの場合

```
[y, fs, bits] = wavread('enshu4.wav');     // enshu4.wav を読み込む
y1 = y(1 * fs + (0 : 4095));               // 開始から 1 秒後の 4096 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半分を破棄
f = linspace(0, fs/2, length(Y1));         // 周波数軸を作成
plot(f, abs(Y1));                          // フーリエ変換結果を図示
wavwrite(y, 44100, 16, 'test7.wav');
sound(y1, 44100);
```

・MATLABの場合，上記とほぼ相違なし．

2.5.5　窓関数

上で行った，非定常信号から一部を取り出す（たとえば開始からN点取り出す）処理は，元の信号に次に関数を掛け算することに相当する：

$$\omega(n) = \begin{cases} 1 & (\text{if } 0 \le n \le N-1) \\ 0 & (n \ge N) \end{cases} \tag{5}$$

この関数を方形窓（rectangular window）という．短時間フーリエ変換は，この方形窓で取り出した部分が無限に続くと仮定している．しかし，窓長N（窓幅ともいう）が信号の周期の正数倍でないと，この無限に続く信号に不連続点が生じ，本来は存在しないピークが観測される．そこで，この問題を避けるため，次の関数が提案されている．

・ハニング窓（Hanning window）

$$\omega(n) = \begin{cases} 0.5 - 0.5\cos\left(\dfrac{2\pi n}{N-1}\right) & (\text{if } 0 \le n \le N-1) \\ 0 & (n \ge N) \end{cases} \tag{6}$$

・ハミング窓（Hamming window）

$$\omega(n) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi n}{N-1}\right) & (\text{if } 0 \le n \le N-1) \\ 0 & (n \ge N) \end{cases} \tag{7}$$

これらの信号の一部を取り出すための関数を総称して，**窓関数（window function）**と呼ぶ．

Scilabでは，”window”関数の第１引数を指定することで，ハニング窓やハミング窓などを簡単に作成できる．（MATLAB では，ハニング窓はhanning(*N*)，ハミング窓はhamming(*N*) で簡単に作成可能．）

例題８）レポート提出課題

・Scilabの場合

```
[y, fs, bits] = wavread('enshu4.wav');     // enshu4.wav を読み込む
y1 = y(1 * fs + (0 : 4095));               // 開始から 1 秒後の 4096 点を取り出す
w = window('hn', size(y1,2));              // ハニング窓の作成(MATLAB の場合 w = hanning(size(y1,1));)
y1 = w.*y1;                                // 作成したハニング窓をかける
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半分を破棄
f = linspace(0, fs/2, length(Y1));         // 周波数軸を作成
plot(f, abs(Y1));                          // フーリエ変換結果を図示
```

※実験レポート第2章「原理もしくは理論」の中で，「窓関数」について簡潔にまとめよ．

2.5.6　スペクトログラム（時間周波数解析）

スペクトログラム（spectrogram）とは，横軸に時刻，縦軸に周波数をとり，各時刻，各周波数における信号の強さ（パワー）を色で表したものである．

これは，信号の取り出す部分をずらしながら**短時間フーリエ変換**を繰り返すことで得られる．

Scilabでは，mapsound という関数を使うことで簡単にスペクトログラムを得ることができる．

```
mapsound (w, dt, fmin, fmax, simpl, rate)    // サウンドマップをプロット（
引数）dt,fmin,fmax,simpl,rate
  スカラー．デフォルト値 dt=0.1, fmin=100, fmax=1500, simpl=1, rate=22050;
説明）
  音のサウンドマップをプロット．強い音は赤，弱い音は青で表示．薄緑が中間くらい．
  時間増分dt毎にFFTを行う．rate はサンプリング・レート．
  速度面を考慮してsimpl 点が収集される．fminおよびfmaxがグラフィック境界として使用される．
※MATLABとの相違点）
  オーバーラップさせられない，窓関数をかけていないなど，基本的な機能が不足．
  窓関数をかけたスペクトログラムを生成したい場合は，下記のWebサイトを参考にすると良い．
  ・引用元：Scilab使いこなしブログ，Scilabで時間周波数解析（スペクトログラム）2
    http://hotic.blog129.fc2.com/blog-entry-6.html
```

例題９）

```
[y, fs, bits] = wavread('enshu4.wav');     // enshu4.wavを読み込む（読込む音声ファイルは任意）
scf();                                     // グラフィックウインドウを開く
mapsound(y, 0.02, 200, 10000, 1, fs);   // スペクトログラムをプロット（引数は任意に変更）
f = gcf();
f.color_map = jetcolormap(256);          // 色を変更
```

【参考：MATLABの場合】

MATLAB では，spectrogram 関数を使うことで簡単に得られる．

短時間フーリエ変換 (STFT) を使ったスペクトログラム

```
[S, F, T, P] = spectrogram(X, WINDOW, NOVERLAP, NFFT, Fs)
返値）
S：ベクトル X で指定された信号のスペクトログラム
F：計算したスペクトログラムにおける周波数ベクトル
T：時間ベクトル
P：各セグメントのパワースペクトル密度 (PSD) の行列表現
引数）
X：信号が格納されたベクトル
WINDOW：窓関数
NOVERLAP：信号の取り出す部分をずらすのに，どの程度オーバラップさせるか
          Xの各セグメントがオーバラップするサンプル数.
          指定されない場合，デフォルト値は50%のオーバラップを得るために使用.
NFFT：離散フーリエ変換を計算するために使われる周波数点数
      （指定されない場合，デフォルトの NFFT）
```

例）

```
[y, fs, bits] = wavread('enshu4.wav');
spectrogram(y, hamming(64), 32, 256, fs, 'yaxis');
[s,f,t,p] = spectrogram(y, hamming(64), 32, 256, Fs, 'yaxis');
```

2.6　ディジタルフィルタ

※以下の課題では，subplot関数を有効に活用し，時間領域と周波数領域でそれぞれ，波形の比較を必ず行うこと．

フィルタリング

信号を足し合わせると音が重ね合わされた．信号を遅らせて足し合わせると，違う効果が得られる．

例題１０）⇒　実施しないこと．

```
exec wshift.sci;
Fs = 8000;
t = 0 : 1/Fs : 1-1/Fs;
s = sin(2 * %pi * 800 * t) + sin(2 * %pi * 500 * t);
sound(s, Fs);
subplot(3,1,1); plot(s(1:100));
sd = wshift(1, s, -5);                    // sを５点遅らせる
subplot(3,1,2); plot(sd(1:100));
soud(sd, Fs);
ss = s + sd;
subplot(3,1,3); plot(ss(1:100));
sound(ss, Fs);
```

演習５）5点遅らせて足し合わせることで，どちらかの成分が除去された．どの成分が除去されたか？

演習６）レポート課題

別の成分を除去するには，何点遅らせればよいか？

このように，遅れを上手く利用すると，ある成分を除去したり，強調したりすることができる．一般的には，ある成分を除去することが多いため，このような処理を*ディジタルフィルタ*と呼ぶ．

遅れの点数に対応した数列を用いることで，フィルタの処理方法を表すことがある．上記の信号それ自身に，5点遅らせてそのままの大きさで足す処理を$h = [1\ 0\ 0\ 0\ 0\ 1]$という数列で表すとする．つまり，$h(1)$は信号自身を表す．$h(6)$は$h(1+5)$である．つまり，5点遅らせていることを示す．

このフィルタhを時間領域の信号sに適用する．

```
h = [1 0 0 0 0 1];
sf = conv(s, h);
plot( sf (1 : 100) );
```

convは，畳み込みという計算を行う．

※実験レポート第2章「原理もしくは理論」の中で，「畳み込み」について簡潔にまとめよ．

演習７）conv([1 3],[2 4]);などの計算を試し，convとはどのような計算かを確認せよ．

***h*の設計によって，さまざまな効果が得られる**ことを確認する．

例題１１）

まず，雑音交じりの信号s_nを作成する．

```
n = 1 : 100;
Fs = 8000;
t = 0 : 1/Fs : 1-1/Fs;
nz = grand(1, length(t), 'nor', 0, 1);     // MATLABの”randn”関数に近いノイズ
plot(n, nz(n));
sound(nz, Fs);
s = sin( 2 * %pi * 440 * t);
sn = 0.8 * s + 0.1 * nz;
plot(n, sn(n));
sound(sn, Fs);
```

フィルタhを適用することによって，雑音の影響が軽減したことがわかる．

```
h = ones(1, 3) / 3
fsn = conv(sn, h);
plot(n, fsn(n));
sound(fsn, Fs);
```

演習８）長さ5の整数の乱数列を作成し，riという変数に代入せよ．次に，conv(ri, h)を計算し，
その結果からhを畳み込むことは，riに対してどのような計算を行っているのかを考察せよ．

演習９）レポート提出課題

***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．**

演習１０）h=[1 -0.98]として，上記のs_nに適用するとどのような音になるか確認せよ．

演習１１）これらのフィルタの効果をspectrogramで観察し，どのような効果なのかを確認せよ．

サンプリング周波数8 [kHz]で，2[kHz]以下の低域だけを通過させるLPFの数列は，

```
h = wfir('lp', 40, 0.5, 'hn', [0 0]);   //  線形位相FIRフィルタ
plot(h);                                  // MATLABの場合，離散データ列プロット”stem(h)”
```

で計算および確認できる．かなり複雑な数列であることが分かる．

演習１２）*h*を上記のs_nに適用し，どのような音になるかを確認せよ．

演習１３）レポート提出課題

wfirを用いて，550 [Hz]以下の低域だけを通過させるLPFを設計し，そのLPFをs_nに適用せよ．

演習１４）レポート提出課題

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．

2.7　データの平滑化（移動平均）

ノイズなどの影響によりデータを平滑化することで，それらの影響を除去することが行われている．平滑化手法としては，移動平均（単純移動平均，加重移動平均）やバターワースなどの手法が一般的である．本節では，もっとも基本的な単純移動平均について述べる．

単純移動平均は，時系列に限らず系列データをある一定区間ごとの平均値を区間をずらしながら求める平滑化手法である．問題点として，遮断周波数が固定で，必ずしも所望の遮断周波数で平滑化できないことが挙げられる．

3 点移動平均

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

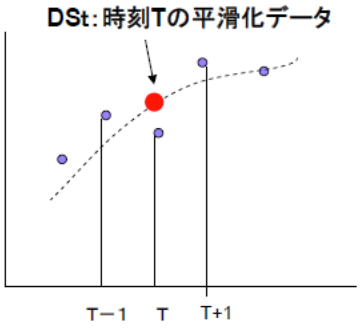
DS t　＝ 時刻 t における平滑化データ

D t -1　＝ 時刻 t -1 における生データ

Dt　＝ 時刻 t における生データ

Dt+1　＝ 時刻 t +1 における生データ

遮断周波数 ＝ 0.182×(サンプリング周波数)



5 点移動平均

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

DS t　＝ 時刻 t における平滑化データ

D t -2　＝ 時刻 t -2 における生データ

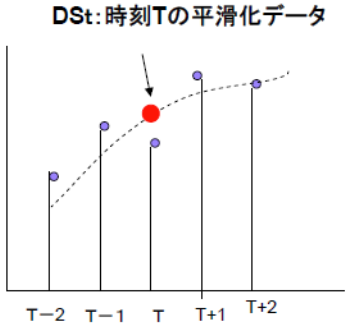
D t -1　＝ 時刻 t -1 における生データ

Dt　＝ 時刻 t における生データ

Dt+1　＝ 時刻 t +1 における生データ

Dt+2　＝ 時刻 t +2 における生データ

遮断周波数 ＝ 0.131×(サンプリング周波数)



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s_nとする．