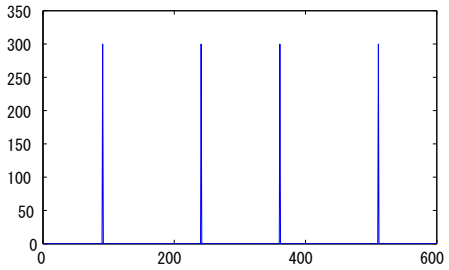
6　フーリエ変換

正弦波を重ね合わせるといろいろな音色の音を作れた。逆に、音をいくつかの正弦波に分解することもできる。それを可能にする技術としてフーリエ変換がある。本講義で扱うディジタル信号に対しては、離散フーリエ変換という技術がある。MATLAB では、`fft` という関数で計算できる。さっそく `fft` を試してみる。

ソースコード 12: `fft` を用いたスペクトルの推定

```
1 >> fs=100;
2 >> t=0:1/fs:7;
3 >> y=sin(2*pi*15*t)+cos(2*pi*40*t);
4 >> cs=fft(y,600);
5 >> plot(abs(cs))
```

この例では、4 行目で信号 `y` に対する 600 点の離散フーリエ変換 (Discrete Fourier Transform) を出力している。このスクリプトを実行すると、次のようなグラフが表示される。



フーリエ変換の結果、スペクトルが得られる。このグラフは、その絶対値をとったものであり、振幅スペクトルとよぶ。FFT を用いて推定した振幅スペクトルは、グラフの真ん中を中心とした線対称になる。

```
>> length(cs)

ans =

   600

>> abs(cs(297:304))

ans =

  1.0e-012 *

    0.4916    0.2391    0.1781    0.2115    0.1807    0.2115    0.1781    0.2391
```

このように、`abs(cs)` は、600 個のデータからなるベクトルであり、301 点目 (0.1807 のところ) を中心に線対称である。スペクトルは、信号の成分の強さを表す。このグラフでは、全部で 4 本の線が観測される。線対称なので、右半分と左半分の情報は同じなので、一般には左半分だけが使われる。左半分には、2 本の線がある。これが `y` には 2 つの成分があることを示し、その線の高さで成分の強さを、位置で成分の周波数をあらわす。

ところで、前回の資料では、コンピュータの中で音は離散的にあらわされることを述べた。本稿では詳しくは述べないが、連続的なデータを離散的に表すためには、データの成分はサンプリング周波数の 1/2 未満でなければならない。この上限の周波数のことをナイキスト周波数とよぶ。**ソースコード 12** のサンプリング周波数は 100Hz なので、ナイキスト周波数は 50Hz となる。

スペクトルの右側では、ナイキスト周波数未満の周波数を表す。つまり、**ソースコード 12** のスペクトルでは、`cs(1)` から `cs(301)` までの 300 個の間隔で 50Hz を表す。つまり、`cs(2)` は $50/300 = 1/6$ Hz を表し、`cs(7)` は $50/300 \times (7-1) = 1$ Hz を表す。したがって、図の中で線のある位置がわかれば、成分がわかる。また、フーリエ変換の点数が変わると、分解できる周波数の細かさが変わることがわかる。分解できる周波数の細かさを周波数分

解能とよぶ。前述の計算方法から、フーリエ変換の点数が多いほど周波数分解能が向上することがわかる。また、FFT の長さは、FFT のアルゴリズムの特性から 2 のベキ乗のときに効率がよいので、2 のベキ乗が用いられる。

どの位置に線があるかを調べるには、プロット図を拡大する方法や、線の周辺だけをプロットしてみる方法などがある。しかし、図から観測するのではなく、プログラミングでも線のある場所を調べることができる。

```
1  >> a=[5 1 0 2 6 3]
2
3  a =
4
5       5     1     0     2     6     3
6
7  >> a>4
8
9  ans =
10
11      1     0     0     0     1     0
12
13 >> a(a>4)
14
15 ans =
16
17      5     6
18
19 >> find(a>4)
20
21 ans =
22
23      1     5
```

この例では、7 行目でベクトルに対して比較演算子である > を適用している。この場合は、a のそれぞれの要素に対し、4 と比較して、4 より大きければ、1 をそうでなければ 0 を返している。(11 行目: 1 は論理値の真を表わし、0 は偽を表わす)

13 行目は MATLAB 独特の書き方である。a() はベクトル a の要素を指定しているのであるが、ここでは論理配列を引数としている。論理配列を引数とした場合、真のインデックスの値だけを出力する。この場合、1 番目と 5 番目だけが真なので、1 番目の値 5 と 5 番目の値 6 が表示される。

値を直接出力するのではなく、インデックスを調べるときには、find という関数を利用する (19 行目)。find の使い方は help で各自調べること。

この find を使うと、**ソースコード 12** の続きで、次のようにすれば、線の位置がわかる。

```
>> find(abs(cs)>250)

ans =

    91   241   361   511
```

つまり、91 点目と 241 点目に成分があることがわかる。91 点目は $50/300 \times (91-1) = 15$ Hz 、241 点目は $50/300 \times (241-1) = 40$ Hz を表す。それぞれ **ソースコード 12** の 3 行目の sin 、cos の周波数に対応していることがわかる。このように、フーリエ変換でわかる信号の成分とは、正弦波の重み付き和に分解したときの成分である。

ところで、cos 関数の一般形は、次のようになる。

$$y = A\cos(2\pi ft + \phi) \tag{4}$$

この ϕ のことを位相とよぶ。

フーリエ変換の結果は、実は複素数である。

```
>> cs(91)

ans =

 −9.2300e−012 −3.0000e+002i

>> cs(241)

ans =

  3.0000e+002 −1.0978e−011i
```

5 行目の値は `cs(91)` が、$-9.23 \times 10^{-12} - 300i$ つまり、$300i$ であることを表していて、11 行目は、`cs(241)` が 300 であることを表している。ϕ は、複素平面 (位相平面) での角度 (位相角) である。この位相角を求める MATLAB 関数が `angle` である。

```
>> angle(cs([91 241]))

ans =

   −1.5708   −0.0000

>> ans/pi

ans =

   −0.5000   −0.0000
```

`cs(91)` の位相は $-1/2\pi$ で、`cs(241)` の位相は 0 であることがわかる。つまり、フーリエ変換の結果をそのまま利用すると、**ソースコード 12** の `y` の成分は、$\cos(2\pi 15t - 1/2\pi)$ と $\cos(2\pi 40t)$ であることがわかるのである。

7 窓関数

フーリエ変換は信号を正弦波に分解できる、と書いたが、実はその説明では不十分である。厳密には、周期性の信号でないと正弦波には分解できない。離散フーリエ変換では、フーリエ変換する範囲を有限としているが、その範囲の外側では、範囲の中と同じ変化が繰り返されていると仮定している。つまり、**ソースコード 4** の場合、`y(1)` から `y(600)` までの範囲での変化が、`y(601)` から先も続くとしている。実際、15Hz と 40Hz の正弦波は $y(i)$ と $y(i+600)$ は同じ値となるので問題ない。

しかし、逆にいうと、次のような場合は問題が生じる。(`y` は**ソースコード 12** で作成したもの)

```
>> plot(abs(fft(y,599)))
>> plot(abs(fft(y,601)))
```

`fft` の長さが 600 点ではないので、最後のところで周期性が若干崩れる。そのため、`y` の成分に対応したインデックスだけでなく、その周辺の値も 0 より大きくなってしまっている。このような現象をリーク (leak, 漏れ) と呼ぶ。要するに、実際に含まれている成分が、真の値より小さく推定される一方で、実際には含まれていない成分が少し含まれるという分析結果になってしまっている。

信号に含まれている成分の周期があらかじめわかっているのであれば、`fft` の点数は、周期にあわせた点数にすればよい。しかし、`fft` は、信号にどのような成分が含まれているかわからないときに、それを知りたくて用いることが多い。したがって、ちょうどよい点数で `fft` をかけることは不可能に近い。

そこで、実際の信号を `fft` で分析するときには、窓関数を利用することが多い。窓関数を使うことで、`fft` をかける信号の両端の変位 (値) をなるべく 0 に近い値にし、擬似的に周期性を担保する。

窓関数の例としてハン窓 (ハニング窓) を見てみる。

```
>> w=hann(600);
>> plot(w)
```

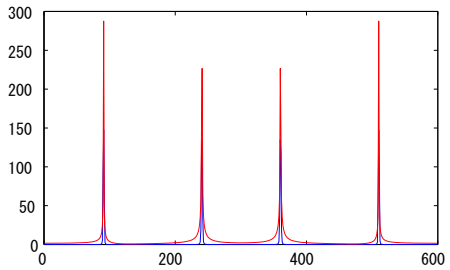
このように中央部分は 1 で裾に向かってゆるやかに 0 になるように変化する。信号に窓関数をかけるというのは、窓関数で振幅変調をおこなうことである。

窓関数を用いることで、リークが減ることを確認する。

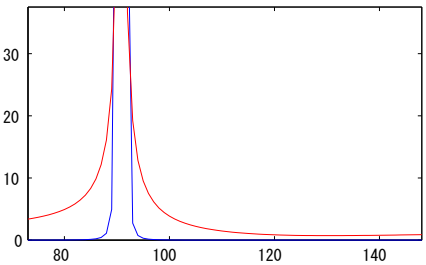
ソースコード 13: ハン窓の効果

```
1 >> hy=y(1:599).*hann(599)';
2 >> plot(hy)
3 >> hcs=fft(hy);
4 >> plot(abs(hcs))
5 >> lcs=fft(y(1:599));
6 >> hold on
7 >> plot(abs(lcs),'r')
```

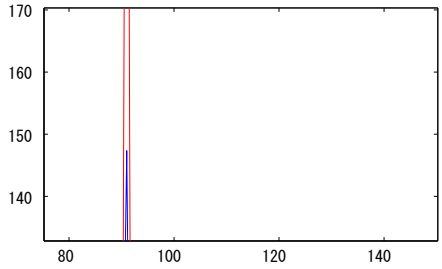
7 行目を実行した後には、次のようなグラフが表示できる。(6 行目は、重ねてプロットするモードに変更する命令である) 7 行目の plot 関数の第 2 引数の'r' は、赤い線でプロットすることを示す。



このグラフを見ると、例えば、横軸の 90 あたりの成分のふもとあたりでは、赤いグラフの方が広がっていることがわかる。つまり、**ソースコード 13** の 4 行目でプロットしたグラフでは、リークが軽減されていることがわかる。その部分のふもとあたりを拡大すると、その様子がはっきりと確認できる。



その部分の先端の方を拡大すると、次のようになる。



窓関数をかけるとリークが軽減されるかわりに、大きさが小さくなることがわかる。

8　音声のフレーム処理

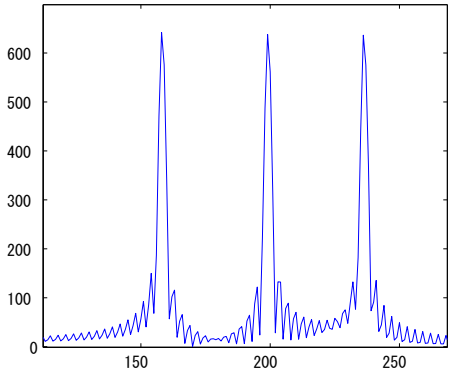
フーリエ変換は fft の長さの間は、対象となる信号は周期的であると仮定している。したがって、対象となる信号の周期性や性質によって fft の長さを適切に選ばなければならない。例えば、120BPM (1 分間に 4 分音符が 120 回演奏される速さのこと) のメロディーの 4 分音符の音色を分析したい場合であれば、その音符の間ずっと全く同じ性質であったとしても $60s \div 120 = 0.5s$ なので、500ms 以内の長さで分析しなければならない。

不適切な長さで分析してしまうと、意図しない結果が得られてしまう[2]。。

ソースコード 14: 複数の音が含まれた場合のフーリエ変換

```
>> [y,fs]=wavread('domiso.wav');
>> plot(abs(fft(y)))
```

このグラフの成分のあるあたりを拡大すると、次のようになる。



domiso.wav は、それぞれ正弦波からなる「ド」「ミ」「ソ」の 3 つの音が並ぶフレーズである。それらの全体でフーリエ変換してしまうと、3 つの成分がある音のようなスペクトルになってしまっている。

また、fft が、分析対象となる区間に常にぴったり合うとは限らない。例えば、「おんせい」という単語の発声データでは、「お」「ん」「せ」「い」という音、それぞれの長さは違うし、どこからどこまでがどの音なのかも実ははっきりしない。このような対象に対しては、少しずつずらしながら分析することが一般的である。

長いデータを少しずつの区分に分割して処理するとき、この区分のことをフレーム (frame) とよぶことがある。MATLAB には、1 次元のデータをフレームに分割し、窓関数をかけて fft 処理することをまとめて処理する関数が用意されている。spectrogram である。

ソースコード 15: spectrogram を用いてフレームごとのスペクトルを求める例

```
>> [y,fs]=wavread('domiso.wav');
>> S=spectrogram(y,hann(256),128,256);
>> size(S)

ans =

   129    36
```

ソースコード 15 の S は 129 行 36 列の行列となっている。この 1 列目は y の最初の 256 点にハン窓をかけて fft した結果が含まれている。

ただし、fft の結果の後半は前半から計算できる (6 章では絶対値をとったものに関して前半と後半が線対称になると述べたが、複素数としては、対応する点が共役複素数となる) ので、前半の 129 点だけを含んでいる。したがって、列ベクトルの長さは 129 となっている。

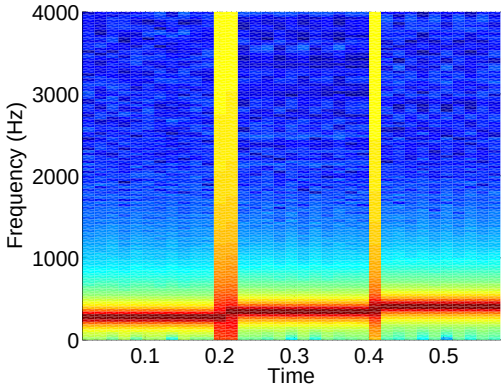
ソースコード 6 の spectrogram の第 2 引数はフレームにかける窓関数、第 3 引数はずらすときに重なるデータ点数、第 4 引数は fft の長さである。

spectrogram を用いて、フレーム処理して求めたスペクトルの様子を可視化することもできる。

[2]音声ファイルは授業支援システムからダウンロードしておくこと

```
>> spectrogram(y,hann(256),128,256,fs,'yaxis')
```

表示されているように横軸が時間で縦軸が周波数である (第 4 引数は横軸を時間にすることを指定している)。正しく表示されると次のようなグラフが表示される。



赤い部分が強い周波数成分を表す。このような可視化の方法およびその図をスペクトログラムとよぶ。スペクトログラムでは、周波数成分の時間変化を見られる。縦軸は正規化周波数となっている。これは、ナイキスト周波数を 1 としたものである。

同じデータに対し、ずらし幅はそのままで `fft` の長さを増やしてみる。

```
>> figure(2)
>> spectrogram(y,hann(512),128,256,fs,'yaxis')
```

`figure(2)` は図を出力するウィンドウを指定している。何も指定しない場合は、`figure(1)` と指定するのと同じである。`figure` 関数を利用すると同時に複数の図を表示できる。

9 逆フーリエ変換

フーリエ変換は時間領域の信号を周波数領域に変換する計算である。その逆に、周波数領域の信号を時間領域に変換する計算が逆フーリエ変換である。MATLAB では `fft` の逆は `ifft` である。

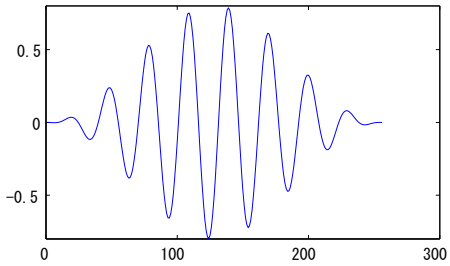
スペクトログラムの結果得られたフレームごとのスペクトルは省略されている部分を追加 (復元) することによって `ifft` で時間波形を復元できる。

ソースコード 16: `ifft` を用いたスペクトログラムからのフレームの時間波形の復元

```
1 >> SL=S(:,10);
2 >> SR=flipud(conj(S(2:end-1,10)));
3 >> S10=[SL;SR];
4 >> plot(abs(S10))
5 >> plot(ifft(S10))
```

前章でも述べたようにスペクトルの中央の右側と左側は中央を中心に線対称のような位置で対応していて、値は複素共役になっているので、2 行目のように右側の値から左側の値を計算できる。2 行目の `end` はベクトルの最後のインデックスを返す変数である (詳しくは `help end` を参照のこと)。

5 行目でプロットされるグラフは次のようになる。



窓関数をかけた信号が復元されていることがわかる。

フレームをずらし幅を意識しながら足し合わせると (完全に同一にはならないが) 時間波形を復元できる。

ソースコード 17: フレーム表現の複素スペクトルからの時間波形の復元

```
1  >> [y,fs]=wavread('domiso.wav');
2  >> fftlen=256;
3  >> noverlap=128;
4  >> S=spectrogram(y,hann(fftlen),noverlap,fftlen);
5  >> [spsize slen]=size(S);
6  >> ry=zeros(1,slen*fftlen-(slen-1)*noverlap);
7  >> for i=1:slen,
8  SL=S(:,i);
9  SR=flipud(conj(S(2:end-1,i)));
10 i1=1+(fftlen-noverlap)*(i-1);
11 ry(i1:i1+fftlen-1)=ry(i1:i1+fftlen-1)+ifft([SL;SR])';
12 end
13 >> plot(ry)
14 >> soundsc(ry,fs)
15 >> soundsc(y,fs)
```

6 行目の `ry` は、復元される時間波形を格納するベクトルを予め作成している。