6. フーリエ変換

数学で勉強したように、フーリエ変換とは、ある関数を周波数の違う三角関数の重ね合わせで表現するとき、その重ね合わせる係数を求める作業のことだ。例えば音声信号をフーリエ変換すると、周波数、つまり音の高低のスペクトルが得られる。音楽好きな人ならフーリエ変換のスペクトルそのものを表示するイコライザーを知っているだろう。フーリエ変換は音声、画像処理、デジタル通信、信号処理には必ず出てくるし、病院に行って頭の断層写真を撮るときに使われる MRI も CT スキャンも原理はフーリエ変換で、応用例は数限りない。

数学で勉強したフーリエ変換は、連続量を扱うものだが、コンピュータの場合は、連続量を扱えないなので、複素フーリエ変換の代わりに離散フーリエ変換(discrete Fourier transform, DFT)が基礎になる。ただ、実際には DFT を高速化したアルゴリズムである**高速フーリエ変換(fast Fourier transform, FFT)**をもっぱら使うので、数値計算や各種応用分野でフーリエ変換と言えば、この FFT のことを指していると言っても過言ではない。フーリエ変換とその応用をきちんと取り上げると一冊の本になるくらいの奥の深い内容ではあるが、ここではどんなことができるのか、どうやって使うかを中心に、簡単に見てみよう。

6.1. フーリエ変換を憶えていますか?

フーリエ変換(正確には複素フーリエ変換)てなんだっけという人のために、少し説明すると、ある関数 $x(t)$ を、

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{-2\pi i f t} df$$

のように、三角関数 $e^{2\pi i f t} = \cos(2\pi f t) + i \sin(2\pi f t)$ の重ね合わせで表現したとき、その係数 $X(f)$ を求める変換のことをフーリエ変換という。つまり、

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{2\pi i f t} dt$$

これが、フーリエ変換。逆に、$x(t)$ を求める方は、逆フーリエ変換という。一般に $X(f)$ は、複素数になるので、複素数の極形式で $X(f) = A(f) e^{i\phi(f)}$ と表す。こうすると、$X(f)$ の $A(f)$ によって、「周波数 f の波を、どのくらいの重み付けで重ねるか」という振幅の情報が得られ、一方、$\phi(f)$ で「周波数 f の波をどのくらいずらして重ねるか」という位相の情報が得られる。フーリエ解析の多くの場合は、周波数成分の大小、つまりスペクトルを議論することが多いので、実際上は、位相はあまり考えず、振幅 $A(f)$ か、その 2 乗 $A(f)^2$ を議論することが多い。これは、$A(f)^2$ がある波長 f の波が持つエネルギーを表すためで、$A(f)^2$ はパワースペクトルと呼ばれたりする。ちなみに、波の全パワーは、振幅の 2 乗を全区間で積分して得られる。(パーセバル(Parseval)の定理)

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |A(f)|^2 df$$

細かいことを抜きにして、「結局、フーリエ変換って何?」と聞かれたら、「時間変化する情報を周波数スペクトルに変換すること」と言うのが、一番簡単な答えだ。もう少し一般化すれば、「逆数の空間に変換すること」。時間<->周波数でなくて、空間<->波数でもよいのだが、いきなり抽象化すると理解しづらくなる人のために、ここでは、時間<->周波数の関係で話を進めることにする。

6.2. 離散フーリエ変換(DFT)

コンピュータでは、連続量は扱えないので、複素フーリエ変換は使えない。そこで離散量を扱う離散フーリエ変換(DFT)が考案された。手っ取り早く式だけを示して比較すると、

複素フーリエ変換	離散フーリエ変換(DFT)
$X_n = \sum_{k=1}^{N} x_k e^{i\frac{2\pi}{N} n(k-1)}$	$X(f) = \int_{-\infty}^{\infty} x(t) e^{2\pi i f t} dt$
$x_k = \frac{1}{N} \sum_{n=1}^{N} X_n e^{-i\frac{2\pi}{N} n(k-1)}$	$x(t) = \int_{-\infty}^{\infty} X(f) e^{-2\pi i f t} df$

である。N は離散化したデータ点のサイズ。i は複素数。両者ともフーリエ変換、つまり「ある時間の関数を周波数スペクトルに変換すること」なので、DFT も、「ある離散時間データ列 x_k (k=1, 2 ... N) を、その**逆数の空間**である周波数の数列 X_n (n=1, 2 ... N)に変換すること」である。しかし、複素フーリエ変換は**無限の長さ**の**連続量**を扱うのに対し、DFT は**①有限の長さ**の**②離散量**を扱うという二つの制限を受ける。

まず①**有限長の制限**について考える。時間刻みを T_Δ、データ点数を N とすると、全時間は、NT_Δになる。したがって、取り

得る周波数の最小値は、その逆数の $1/NT_\Delta$ と決まってしまう(図6.2参照)。これをf_sとおこう。別の言い方をすると、最も周波数の低い波は、最初(k=1)と最後(k=N)の点を節とした一周期の正弦波ということだ。図 6.2 の一番波長の長い波のこと。次に周波数の低い波は、その二倍波なので、2 f_s になり、以下、3 f_s、4 f_s … と、周波数は、 f_s の整数倍の数列となる。

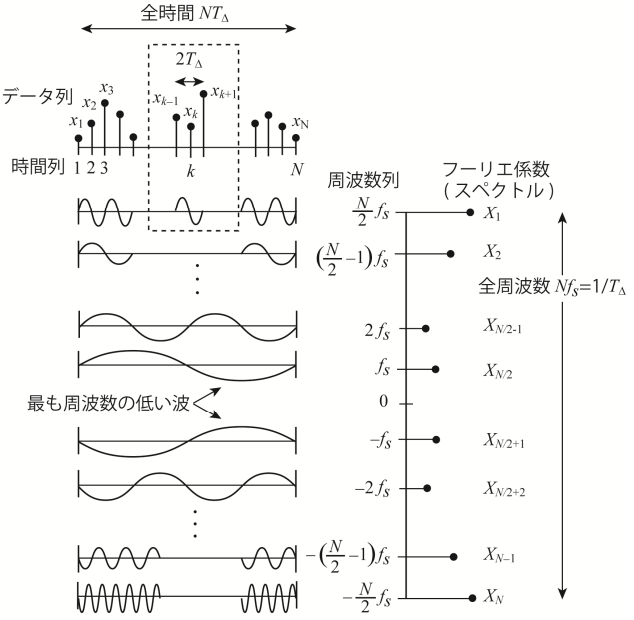

図 6.2 DFT と逆 DFT を模式的に示した図

次に②の**<u>離散量による制限</u>**について考える。時間データの点数はN個なので、周波数データもN個になる。周波数データはf_s刻みなので、f_sからNf_sまでで、N個と思うかもしれないが、周波数には、負の領域もあるので、$-\frac{N}{2}f_s$ から $+\frac{N}{2}f_s$ となる。(ただし周波数 0 は除く) 別の言い方をすると、正弦波として成り立つためには、$2T_\Delta$時間が必要だから、最も高い周波数は、$1/2T_\Delta = \frac{N}{2}f_s$ となるということでもある。

DFT を実際に計算するには、DFT の定義式

$$X_n = \sum_{k=1}^{N} x_k e^{i\frac{2\pi}{N}n(k-1)}$$

を具体的に書き出してみるとわかりやすい。

$$\begin{cases} X_1 = & x_1 e^{i\frac{2\pi}{N}\times 0} & + x_2 e^{i\frac{2\pi}{N}\times 1} & +\ldots+ & x_N e^{i\frac{2\pi}{N}(N-1)} \\ X_2 = & x_2 e^{i\frac{2\pi}{N}2\times 0} & + x_2 e^{-i\frac{2\pi}{N}2\times 1} & +\ldots+ & x_N e^{i\frac{2\pi}{N}2(N-1)} \\ \vdots & & & & \vdots \\ X_N = & x_N e^{i\frac{2\pi}{N}N\times 0} & + x_2 e^{i\frac{2\pi}{N}N\times 1} & +\ldots+ & x_N e^{i\frac{2\pi}{N}N(N-1)} \end{cases}$$

これは行列形式で書くと、

$$\begin{pmatrix} X_1 \\ X_2 \\ \\ X_N \end{pmatrix} = \begin{pmatrix} 1 & e^{i\frac{2\pi}{N}} & & e^{i\frac{2\pi}{N}(N-1)} \\ 1 & e^{i\frac{2\pi}{N}2} & & e^{i\frac{2\pi}{N}2(N-1)} \\ \vdots & & & \\ 1 & e^{i\frac{2\pi}{N}N} & & e^{i\frac{2\pi}{N}N(N-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ x_N \end{pmatrix}$$

のことなので、DFT とは、結局のところ $N\times N$ の行列演算という簡単な変換であることが分かる。しかも、見て分かる通り、行列の要素はデータ点のサイズ N にしかよらない。結局のところ、時間データのベクトル $\boldsymbol{x}$ に DFT の行列をかければ、周波数データのベクトル $\boldsymbol{X}$ に変換される。

6.3. 高速フーリエ変換(FFT)

DFT の行列のサイズは $N\times N$ なので、要素を全部計算するための計算量は、単純には N^2 に比例する。つまり計算量は $O(N^2)$ である。しかし、三角関数の周期性、つまり、$e^{i\frac{2\pi}{N}N} = e^{i\frac{2\pi}{N}2N} = \cdots = 1$ や、$e^{i\frac{2\pi}{N}} = e^{-i\frac{2\pi}{N}(N-1)}$ などの性質を利用すると、実際に計算しなくてはならない要素の数は N^2 より減らすことができる。これをうまく利用したアルゴリズムが**<u>高速フーリエ変換(FFT)</u>**で、データ点のサイズ N を 2 のべき乗にすると、計算量を $O(N^2)$ から $O(N\log_2 N)$まで減らすことができる。仮に N=10^6 程度のデータ列があり、DFT では、5 日ぐらい計算に時間がかかったとすると、FFT では 10 分程度で済む計算になるので、かなり高速だ。それほど難しいアルゴリズムではないのだが、いろいろな本に載っているので、詳しい内容についてはフーリエ変換の本を参照していただきたい。

6.4. FFT を使ってみる

DFT と FFT がどんなものか、だいたい分かったところで、実際にフーリエ解析をしてみよう。まず、解析するために何かのデータが必要だ。ネットを探すといろいろの時系列データがある。例えば、www.robjhyndman.com/TSDL のような時系列データをたくさん集めたサイトがあるので、今回は、このサイトの Physics のカテゴリにある andrews14.dat というデータを拾ってきて解析してみる。これは、1749 年から 1983 年までの太陽の黒点の数を毎月調べたデータだそうだ。(andrews14.dat と検索しても出てくる) このファイルを保存して、まずは表示してみよう。

プログラムは Sunspotplot.m。実行結果は図 6.4.1

```
% Sunspotplot.m
clear; load andrews14.dat   %データのロード
N=(1983-1749+1)*12; t=1:N; %時系列のベクトルを作る
x=reshape(andrews14',N,1);
 %andrews14 は、235 年×12 ヶ月という行列なので、
 %   2820 ヶ月×1 列というベクトルになおす。
save sunspot.mat x  N -V6    %データの保存
figure(1)                    %1 枚目の図
plot(t/12+1749,x);
xlabel('Time  (year)');
ylabel('Sunspot Number');
```

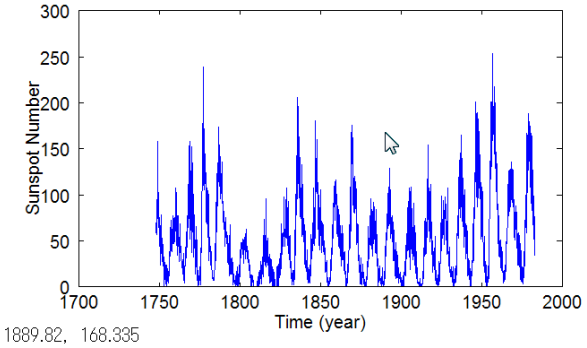

図 6.4.1 太陽黒点振動。縦軸が観測された黒点の数

図 6.4.1 をみると、黒点の数が周期的に振動している様子が分かる。周期を測るのに、図に定規を当てるのも一つの手ではあるが、フーリエ解析という強力なツールを身につけたので、これを使って、周期を求めよう。 Octave には、FFT を実行する関数 `fft` が用意されているので、これを使えば、`X=fft(x)` とするだけで、簡単に複素数列 `X` が求まる。

```
%SunspotFreq.m
clear; load sunspot.mat
fs=1/N;             %周波数刻み（全時間の逆数）
F=(1:N/2)*fs;       %周波数ベクトルを作る
X=fft(x);           %FFT の実行
X(1)=[];           %X の最初のデータ(周波数 0)を捨てる
Amp=abs(X);         %X の絶対値(振幅)を Amp とする
Pow=Amp(1:N/2).^2;
     %Amp の正の周波数成分の 2 乗(パワー)を Pow とする
figure(2)                %二枚目の図
plot(F,Pow,'k-');    %正の周波数成分だけを表示
xlabel('Frequency  (1/month)');
ylabel('Power');
```

今、データの刻み T_Δが 1 ヶ月で、サイズが N なので、全時間の長さは N=2820 ヶ月。したがって、最も低い周波数は、$f_s=1/N=1/2820$(ヶ月$^{-1}$)である。一方、最も高い周波数は、$Nf_s/2$ なので、周波数のベクトルは `F=(1:N/2)*fs` となる。`Period` は周期、つまり周波数の逆数。スペクトル `X` には、周波数の正と負があるが、$|X(f)|=|X(-f)|$なので、正の領域だけ考えることにして、パワーは、`X` の振幅の 2 乗とする。

```
%Sunspot.m
clear; load sunspot.mat
fs=1/N;             %周波数刻み（全時間の逆数）
F=(1:N/2)*fs;       %周波数のベクトル(単位は 1/ヶ月)
Period=1./F/12;     %周期のベクトル(単位は年)
X=fft(x);           %FFT の実行
X(1)=[];           %X の最初のデータ(周波数 0)を捨てる
Amp=abs(X);        %X の振幅を Amp とする
plot(Period,Pow,'ro-'); %正の周波数成分だけ表示
xlabel('Period (year)'); ylabel('Power');
axis([0 50 0 1.6e9])
```

これを実行すると、

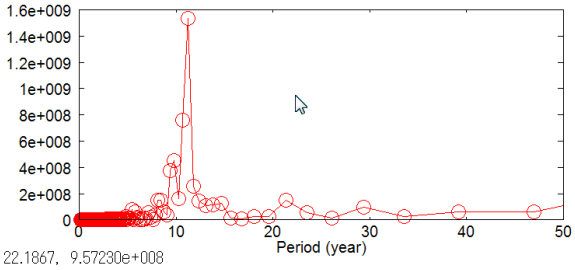

図 6.4.3 黒点データの周期スペクトル

という図が得られる。11 年付近に高いピークが現れていることが分かる。より正確に求めたければ、実行したあと、

```
> format long     （表示する桁数を倍精度にする）
> [P Index]=max(Pow); Period(Index)
ans= 11.1904761904762
```

とすればよい(関数 `max` で、ベクトル `Pow` の最大値 `P` と、その要素番号 `Index` が出力される)。この結果から黒点の振動周期は 11.190...年であると求めることができた。22 年付近にも、ピークがあるように見えるが、黒点の磁場の極性が反転する周期は約 22 年なので、それが表れているのかも。ちなみに 2008 年の夏に、100 年ぶりぐらいで黒点が消えた、というニュースを聞いた人もいるかもしれない。太陽の黒点は、意外にも生活環境に関わりがあるので、興味が有れば swc.nict.go.jp/sunspot などをみてはいかが?

6.5. FFT とフィルター

FFT の応用にはさまざまなものがあるが、その一つにデジタルフィルターがある。データは音声や画像、天体からの光や電波でもいいし、一次元でも二次元でもいい。ここでは、音声データに対してデジタルフィルターをかけることを例題として、フーリエ変換とデジタルフィルターに関してすこし学んでみよう。

まずは音声データをどこかからダウンロードしてくる。今回は、http://www.freesound.org/samplesViewSingle.php?id=17161 というところから、「17161_acclivity_Crickets1.wav」というファイルをダウンロードした。 Windows Media Player や QuickTime などを使って、まずどんな音か聞いてみよう。コオロギ?の声に混じって、フクロウ?カエル? 人間の声や雑踏や風の音も少し聞こえる。

CD やパソコン上の音声データは、1 秒間にデータ点がいくつあるかを規定するサンプリングレート F_s (Hz)が規格で決まっている（f_s とは別物なので注意)。F_s が高ければ高いほど高音領域まで記録できるが、あまり高音まで広げすぎでも、人間の耳に聞こえなくなるので無駄である。そこで、CD の規格では、F_s =44.1kHz と決まっている。つまり 1 秒間に 44,100 個のデータ点があり、データ点の時間間隔はその逆数 $1/F_s$~22μ秒程度になっている。Octave では **wavread** というコマンドで wav 形式の音声ファイルを読み出すことができるので、まずはデータを FFT して、スペクトルを表示してみよう。太陽黒点の解析も、コオロギの音の解析も、プログラム自体はほとんど同じもの。

```
%Crickets.m
clear
x=wavread('17161_acclivity_Crickets1.wav');
                          %wav ファイルの読み込み
[N M]=size(x);            %データサイズ N を抽出
Fs=44100;                 %サンプリングレート(Hz)
F=(1:N/2)*Fs/N/1e3;       %周波数のベクトル(kHz)
X=fft(x);                 %FFT の実行
save Crickets.mat x X Fs F N M -V6
semilogx(F,real(X(1:N/2,1)));
xlabel('Frequency (kHz)');
axis([0.001 20 -3000 3000]);
```

これを実行すると、図 6.5.1 のようになる。

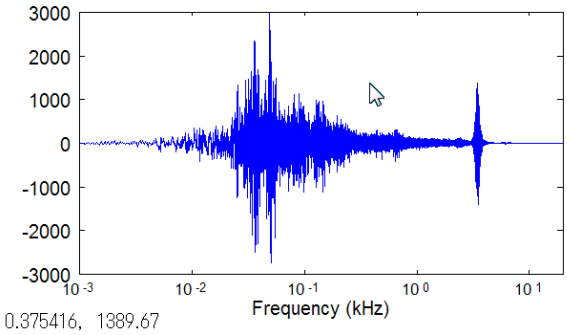

図 6.5.1 元データの音声スペクトル

人間の耳に聞こえる音の周波数は、0.02 から 20 kHz ぐらいだそうなので、だいたいその周波数帯をカバーしていることが分かる。この図をよく見ると、3.5kHz ぐらいの所に細いピークがあることに気づくだろう。これはなんだろうか? それを調べるために、このピーク以外の周波数データをフィルターして消してみよう。フィルターしたあとのデータを逆 FFT してやれば、その周波数帯のみの音声になるはずだ。ある周波数帯だけを通すフィルターをバンドパスフィルタという。逆にある周波数帯だけを通さないものをバンドストップという。他にもローパス、ハイパスなどのフィルターなどがある(これらの用語は、音声だけではなく、通信、エレクトロニクス、光学の分野でも頻繁に使われるので知っておいて損はない)。まずは、バンドパスフィルタを定義する関数 **bandpass** を作る。これは通過させたい周波数帯域を **f1** から **f2** として入力すると、行列の形で **Filt** というフィルターを出力してくれるものだ。フィルターといっても、**f1**<f<**f2** で、要素が **1**、それ以外は **0** であるような、サイズが N×M の行列だ。ここでの M は音声のチャンネルを表すもので、今の音源はステレオで L と R の二つのチャンネルがあるので、M=2 である。

```
%bandpass.m
function Filt=bandpass(f1,f2,F,N,M)
[A I1]=min(abs(F-f1));
    %F=f1 となる要素番号 I1 を求める(本文参照)
[A I2]=min(abs(F-f2));      %f2 も同様
Filt=zeros(N,M);            %まずはすべて 0 に
Filt(I1:I2,:)=1;        %f1 から f2 までを 1 に
Filt(N-I2:N-I1,:)=1;  %-f1 から-f2 までを 1 に
```

[A I1]=...の行の解説をしておこう。周波数 **f1** をベクトル **F** の要素番号に変換したい。つまり、周波数の行列 **F** の要

素のうちで、最も f1 に近い要素の、要素番号 I1 を求めたい。そのため、まず、行列 F からスカラーf1 を引いた新たなベクトル F-f1 を作る。数学では行列とスカラーの引き算はできないが、Octave の場合、F-f1 は、「行列 F の要素のそれぞれから、スカラーf1 を引いた値を要素にもつ行列」と解釈される。次に F-f1 の絶対値 abs をとると、F の各要素から f1 までの距離が並んだベクトルが作られる。さらにその min をとれば、abs(F-f1)の要素のなかで最も 0 に近い要素の値 A と、その要素番号 I1 を出力してくれる。これによって、周波数 f1 をベクトル F の要素番号に変換できる。なお A は関数 min の書式上必要なので書いているが、特に使用しない。

バンドストップフィルタ bandstop も作っておこう。bandpass で作った Filt を使って、1-Filt とすれば、要素が 0 なら、1 になるし、要素が 1 なら 0 になるので簡単。

```
%bandstop.m
function Filt=bandstop(f1,f2,F,N,M)
Filt=bandpass(f1,f2,F,N,M);
Filt=1-Filt;      % bandpass の逆を作る
```

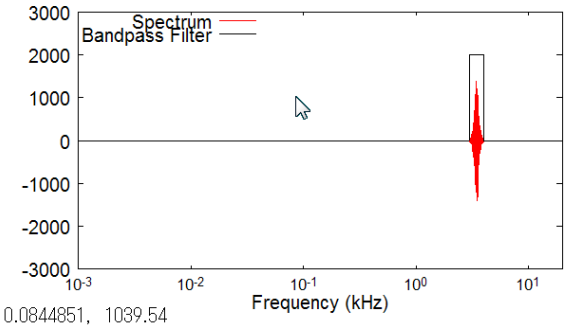

図 6.5.2　バンドパスフィルタをかけた後のスペクトル

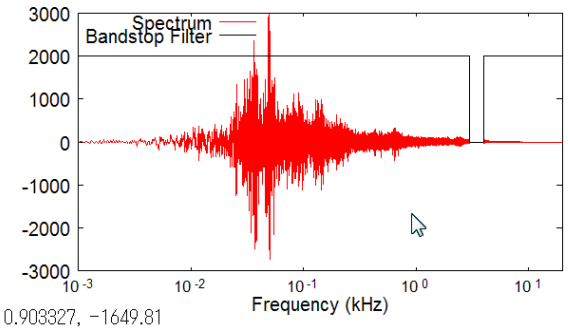

図 6.5.3　バンドストップフィルタをかけた後のスペクトル

3 から 4 kHz の帯域で、バンドパスフィルタをかけるプログラム(CricketsPass.m)は、次のようになる。

```
%CricketsPass.m / CricketsStop.m
clear; load Crickets.mat
f1=3; f2=4;                    %Band の帯域(kHz)
Filt=bandpass(f1,f2,F,N,M); %Bandpass フィルタ
%Filt=bandstop(f1,f2,F,N,M); %Bandstop フィルタ
Filted=Filt.*X;   %スペクトル X にフィルタをかける
semilogx(F,real(Filted(1:N/2,1)),'r', ...
   F,Filt(1:N/2,1)*2000,'k')  %ステレオの片方
        % だけを図示。*2000 は見やすさのため
xlabel('kHz');
axis([0.001 20 -3000 3000]);
S_data=ifft(Filted);  %逆 FFT を実行して音声に戻す
wavwrite(S_data,Fs,'CricketsPass.wav');
%wavwrite(S_data,Fs,'CricketsStop.wav');
```

バンドストップフィルタをかけるプログラム(CricketsStop.m)は 4 と 13 行目をコメントにし、5 と 14 行のコメントを外せばよい。音声スペクトル X にフィルターをかけるには、フィルターとなるベクトル Filt を要素同士で、文字通りかけてやればよい。フィルターをかけた後のスペクトルが、図 6.5.2 と 3 である。フィルターをかけた部分が 0 になっているのが分かるだろう。周波数スペクトルから音声データに戻すには、逆 FFT(ifft)を実行すればよい。最後に wavwrite を使って、音声テータを、それぞれ CricketsPass.wav、CricketStop.wav として wav 形式で保存する。メディアプレーヤーなどで、CricketsPass を再生すると、コオロギの音しか聞こえない。一方、CricketStop を再生すると、逆にコオロギの声は聞こえず、カエル(?)や人間、その他の雑踏の音しか聞こえない! というわけで 3.5kHz ぐらいのピークはコオロギの声であり、デジタルフィルターを使って、コオロギの音だけを取り出す or 消すことができた。

このようにフィルター関数をいろいろ変えてみると、ボーカルだけを消してカラオケを作ったり、あるいは、周波数を変えて高音にしたり(ヘリウムを吸ってしゃべった時みたいな!?)、いろいろなことができるかも。自分の好きな曲や声などいじってみて、試してみてはどうだろうか。ただ、これはフーリエ変換を勉強する教育目的であり、著作権には注意して欲しい。

(http://www.ihokamo.net/school.html 参照)

6.6. 二次元 FFT

二次元の FFT は、特に画像処理や画像解析に関連してよく利用される。ここでは一例として、安藤広重の有名な浮世絵を解析してみよう。

図 6.6.1 安藤広重作「大橋あたけの夕立」

こんな感じの浮世絵。画像データは安藤広重の wiki などでダウンロードできる。非常に鋭いタッチで描かれた雨が印象的で、ゴッホも強く影響を受けたらしく、この絵を模写した絵画があるそうだ。でも、今回はフーリエ変換を使ってこのシーンから雨を消し去ってしまおう。プログラムは最後に掲載しておくので、実行結果だけを説明する。二次元 FFT をすると、実空間の像(浮世絵)は、図 6.6.2 左図に示すような、波数空間の像(スペクトル)に変換される。

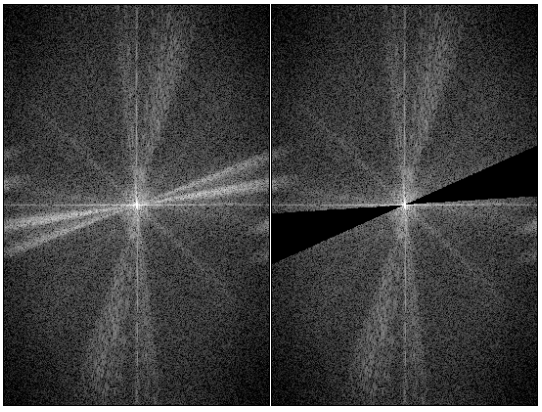

図 6.6.2 フィルター前(左図)と後(右図)の波数空間スペクトル

通常、2 次元の FFT を行うと、このようなスペクトルが得られる。図の中心の明るい点は波数がゼロの点で、最も長い波長の成分を表すので、電流の直流交流との類似性から、DC 成分や直流成分と呼ばれたりする。一次元 FFT と同様に、波数には正と負の領域があり、原点に対して対称であるが、二次元の場合は像が原点に関して 180 度回転対称になっている。像の中央を縦横に走る線は、それぞれ x 軸、y 軸に平行な波を表す。通常画像は長方形をしていることが多いので、画像の端の影響やその他画像の端に並行な構造の効果によってこのような線が現れることが多い。また、特徴的な周期構造があるときは(例えば原子の格子像など)、その周期に特徴的なピークが現れたりする。今回の浮世絵の場合は、雨や橋桁など直線が多く描かれ、特徴的な角度をもった線が多いので(角度の異方性が強い)、波数空間の像も、斜めの輝線が特徴的に現れているようだ。

この斜めの輝線のいくつかが、雨を表しているだろうという予想のもと、図 6.6.2 の右図のように、ある領域だけフィルターでカットしてみる。黒い三角形の部分がフィルターでカットされた領域だ。この波数空間像を逆 FFT すると、実空間像が得られる(図 6.6.3)。

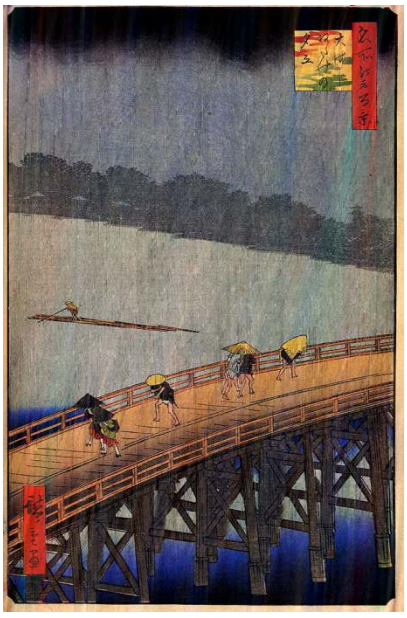

図 6.6.3 フィルター後の実空間像

図 6.6.3 を見て分かるように、橋桁や人物の輪郭の鋭さを保ったまま、ほぼ雨だけ消すことができた。

フーリエ変換は音声や画像処理だけでなく、いろいろな場面

で登場するので、身につけておくと将来きっと役に立つだろう。

使用したプログラムを以下に載せておく。まずはフィルターをかけるための関数(triangle_filter.m)。

```
function M=triangle_filter(M_in,a1,a2)
[a b]=size(M_in);
m_down=zeros(a,b); m_up=zeros(a,b);
for x=1:a/2
    y1=round(b/(a-2*a1) * (x-a1));
    y2=round(b/(a-2*a2) * (x-a2));
    if y1 < 1
        y1=1;
    end
    if y2 > b
        y2=b;
    end
    m_up(x,y1:b)=1; m_down(x,1:y2)=1;
end
m_up=m_up+rot90(m_up,2); %180 度回転して重ねる
m_down=m_down+rot90(m_down,2);
M=(m_up+m_down).*M_in;
M(floor(a/2),:)=M_in(floor(a/2),:);
M(floor(a/2)+1,:)=M_in(floor(a/2)+1,:);
```

つぎに、メインのプログラム(FFT2D.m)を示す。

```
% FFT2D
clear
M=imread('hiroshige.jpg');
  %画像は RGB の三つのチャネルをもつ 8 ビット符号無し整数
    の 3 次元行列(m×n×3 の行列)として読み出される
[a b c]=size(M);
a1=360; a2=490; %フィルター領域の指定
for k=1:3  %RGB の三つのレイヤーそれぞれで FFT を行う
    R=double(M(:,:,k));
          %8 ビット符号無し整数から倍精度に変換
    Avg=sum(sum(R))/a/b;
    Rf=fftshift(fft2(R-Avg));
      %平均値を差し引いたあとで、二次元 FFT を実行し、
      DC 成分をスペクトルの中心に移動する。
    K=Rf;
    K=triangle_filter(K,a1,a2); %自作フィルター
    Rfilt=ifft2(ifftshift(K))+Avg; %逆 FFT
    M(:,:,k)=uint8(real(Rfilt));
          %倍精度から、8 ビット符号無し整数に戻す
end
imwrite(M,'test.png','png')
          %画像データとして保存
v=5;   %画像をそのまま図示すると重たいので画素を間引く
Pf=Rf(1:v:a,1:v:b); Kf=K(1:v:a,1:v:b);
figure;
subplot(1,2,1)
pcolor(1:v:b,1:v:a,real(log(Pf)));
shading flat; caxis([7 15]);
subplot(1,2,2);
pcolor(1:v:b,1:v:a,real(log(Kf)));
shading flat ;caxis([7 15]);
```