第12章　Fourier級数とFourier変換・逆変換

実は前章まで述べ立ててきたことだけで，本章以降の事柄はすべて語り尽くせるのである。序論 (第 1 章) で述べたように，FFT とは，Fourier(無限) 級数として表現された周期 (複素) 関数 $f(\theta)$ を有限 Fourier 級数として近似し，その有限個の項の定積分を更に台形公式を用いて離散化して求める，それを更に ω_N の性質を利用して計算量を激減させる手法であるから，今ここで述べたことは，本書を理解できた賢明なる読者にはタチドコロに理解できるはずである。

しかし，概念は理解できたとしても，それを具体的に実践せよ，と言われると人間なかなか手が動かないものである。そこで，FFT への道のりを 2 章に分け，具体例を交えつつ，少し饒舌に語っていくことにする。余計な御世話だ !，と思った方は，もう一度序論の解説に目を通した後，最後の第 13 章へワープして頂きたい。

12.1　Fourier級数再考

実数を引数とする複素関数 $f(\theta) \in \mathbb{C}$ が，周期 2π の周期関数，即ち

$$f(\theta \pm 2\pi) = f(\theta) \tag{12.1}$$

であるとする。FFT は一般の関数に対してもドカドカ適用されるものだが，こんなきつい縛りをつけて大丈夫か，と思う人は，

1. 応用上は有限区間 $[a,b] \subset \mathbb{R}$ の範囲内だけ扱うのが殆ど

2. 適当な変数変換を使って，$x \in [a,b]$ を $\theta \in [0,2\pi]$ に落とし込むことは簡単

という事実があることを認識して頂きたいものである。

さて，前章でも述べた，$[0,2\pi]$ における正規直交基底 (11.14)

$$e_0(\theta) = \frac{1}{\sqrt{2\pi}},\ e_1(\theta) = \frac{1}{\sqrt{2\pi}} \exp\left(\theta\sqrt{-1}\right), e_2(\theta) = \frac{1}{\sqrt{2\pi}} \exp\left(2\theta\sqrt{-1}\right), \ldots,$$
$$e_n(\theta) = \frac{1}{\sqrt{2\pi}} \exp\left(n\theta\sqrt{-1}\right) = \frac{1}{\sqrt{2\pi}}, \ldots \tag{12.2}$$

を用いると，$f(\theta)$ は無限級数展開 (11.10) の形で

$$f(\theta) = \sum_{i=0}^{\infty} (f, e_i) e_i(\theta) = \sum_{i=0}^{\infty} c_i \exp(n\theta\sqrt{-1})$$

と表現できる。ここで各係数 $c_i \in \mathbb{C}$ は $L^2([0, 2\pi])$ 空間における内積であり

$$c_i = \frac{1}{\sqrt{2\pi}}(f, e_i(\theta)) = \frac{1}{2\pi}\int_0^{2\pi} f(\theta)\exp\left(-n\theta\sqrt{-1}\right)\, d\theta \ (i = 0, 1, ...) \tag{12.3}$$

となる。

12.2　Fourier 変換

周期関数の周期を無限大に引き延ばすと，$[0, 2\pi] \Longrightarrow [-\infty, +\infty] = \mathbb{R}$ となるので，(12.3) の右辺は

$$\int_{-\infty}^{+\infty} f(\theta)\exp\left(-n\theta\sqrt{-1}\right)\, d\theta$$

となる。いわゆる広義積分の一種となるが，ある条件の元に，これは必ず有限の値を持つことが知られている。ここで n の代わりに任意の $x \in \mathbb{R}$ に置き換えた時は関数となる。これを $F(x)$ と書くことにする。この時，

$$F(x) = \int_{-\infty}^{+\infty} f(\theta)\exp\left(-x\theta\sqrt{-1}\right)\, d\theta \tag{12.4}$$

を f の Fourier 変換 (Fourier Transform) と呼ぶ。

ここで詳細は述べないが，f を Fourier 変換した F から元の関数を戻すには

$$f(\theta) = \int_{-\infty}^{+\infty} F(x)\exp\left(x\theta\sqrt{-1}\right)\, dx \tag{12.5}$$

という，Fourier 逆変換 (Inverse Fourier Transform) を行うことで得られる。

12.3　離散 Fourier 変換 (DFT) の導出

上記の Fourier 変換 (12.4) は，結局のところ，Fourier 係数 (12.3) を計算することに帰着される。しかし一般にこの定積分を，被積分関数の原始関数を求めて計算することは難しいので，第 10 章で述べた近似積分計算法を導入する。本書では説明しないが，特に周期関数の場合，最も次数の低い台形公式が理論誤差の点か

ら有利であることが知られており，これに基づいた離散化を行って Fourier 変換の近似値を求めるのが普通である。以下，この方針に基づいて離散化した Fourier 変換，即ち離散 Fourier 変換 (Discrete Fourier Transform, DFT) を導出する。

まず，積分区間 $[0, 2\pi]$ を M 等分し，各離散点を θ_i $(i = 0, 1, ..., M)$ とする。この時，M 個の小区間は $h = 2\pi/M$ の幅を持つので

$$\theta_i = 0 + ih = \frac{2\pi i}{M}$$

と表わされる。これを台形公式 (10.6) を用いて和を取ると

$$\int_0^{2\pi} f(\theta) \exp\left(-n\theta\sqrt{-1}\right) d\theta \approx h\left[\frac{1}{2} f(\theta_0) \exp\left(-n\theta_0\sqrt{-1}\right) + \sum_{i=1}^{M-1} f(\theta_i) \exp\left(-n\theta_i\sqrt{-1}\right) + \frac{1}{2} f(\theta_M) \exp\left(-n\theta_M\sqrt{-1}\right)\right]$$

となるが，この両端の値は周期関数の性質 (12.1) から，それぞれ同じ値になるので，中央の $M-1$ 項の和の部分に統合でき，結局

$$\begin{aligned} C_k &= h \cdot \frac{1}{2\pi} \sum_{i=0}^{M-1} f(\theta_i) \exp\left(-k\theta_i\sqrt{-1}\right) = \frac{1}{M} \sum_{i=0}^{M-1} f\left(\frac{2\pi i}{M}\right) \exp\left(-\frac{2\pi ik}{M}\sqrt{-1}\right) \\ &= \frac{1}{M} \sum_{i=0}^{M-1} f_i \omega_M^{-ik} \approx c_k \end{aligned} \tag{12.6}$$

を計算すればいいことになる。ここで $f_i = f(2\pi i/M)$，ω_M は 1 の M 乗根 ((5.10) 式) である。

さて，有限項 $N-1$ で打ち切った Fourier 級数 (11.15) を用いるとき，$M = N$ としておくと，結局 DFT では，与えられた関数によって決まる N 個の複素数値 f_0, f_1, ..., f_{N-1} に対して，(1.5) の積和式に基づいて

$$C_k = \frac{1}{N} \sum_{i=0}^{N-1} f_i \omega_N^{-ik}$$

によって，これも N 個の $C_0, C_1, ..., C_{N-1}$ を求める，ということになる。

これを C++のプログラムにすると，次のような関数にまとめられる。

```
1:// DFT
2:// INPUT : complex<double> f[n], long int n
3:// OUTPUT: complex<double> x[n]
```

```
 4:int dft(complex<double> x[], complex<double> f[], long int n)
 5:{
 6:    long int i, j;
 7:    complex<double> omega, tmp;
 8:    complex<double> d[n], d_org[n];
 9:
10:    // d = [1, omega^1, omega^2, ..., omega^(n-1)]
11:    omega = exp(complex<double>(0, -2 * M_PI / n)); // M_PI = 3.1415...
12:    for(i = 0; i < n; i++) {
13:        d_org[i] = pow(omega, i);
14:        d[i] = d_org[i];
15:    }
16:
17:    // x
18:    x[0] = f[0];
19:    for(i = 1; i < n; i++) x[0] += f[i];
20:
21:    for(i = 1; i < n; i++) {
22:        x[i] = f[0];
23:        for(j = 1; j < n; j++) {
24:            x[i] += f[j] * d[j];
25:            d[j] *= d_org[j];
26:        }
27:    }
28:
29:    // 1/n
30:    for(i = 0; i < n; i++) x[i] /= (double)n;
31:
32:    return 0;
33:}
```

さて，ω_N は 1 の N 乗根であるので，前述したように N の倍数乗が必ず 1 になるという特殊な性質がある。これを用いて DFT の計算量を更に減らすことはできないか … と考えられたのが FFT である。次はこのことについて，よく考えてみたい。

練習問題

1. 逆 Fourier 変換に対応した DFT は，$C_0, C_1, ..., C_{N-1}$ に対して

$$f_k = \sum_{i=0}^{N-1} C_i \omega_N^{ik}$$

と計算できることを示せ。また，これに対応した C++プログラムを作れ。