第13章　高速Fourier変換へのみちのり

高速 Fourier 変換 (FFT) は，$N = 2^n$ の場合に，1 の N 乗根 ω_N の性質を巧妙に使って計算回数を減らすものである。以下，できる限り平易に解説を行うが，ややこしくなったら,「もともと FFT はややこしいものだ」と素直にあきらめてじっくり考えて頂きたい。

13.1　離散 Fourier 変換再考

具体的に計算量を考えるために，$N = 8 = 2^3$ とした時の DFT を計算する場合で考える。この時は，$f_0, f_1, ..., f_7 \in \mathbb{C}$ が与えられた時に，(12.6) 式に基づいて $C_0, C_1, ..., C_7 \in \mathbb{C}$ を

$$C_k = \frac{1}{8}\sum_{i=0}^{7} f_i \omega_8^{-ik}$$

として計算することになる。これを行列とベクトルの形式で表現すると

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} \omega_8^{-0\cdot0} & \omega_8^{-0\cdot1} & \omega_8^{-0\cdot2} & \omega_8^{-0\cdot3} & \omega_8^{-0\cdot4} & \omega_8^{-0\cdot5} & \omega_8^{-0\cdot6} & \omega_8^{-0\cdot7} \\ \omega_8^{-1\cdot0} & \omega_8^{-1\cdot1} & \omega_8^{-1\cdot2} & \omega_8^{-1\cdot3} & \omega_8^{-1\cdot4} & \omega_8^{-1\cdot5} & \omega_8^{-1\cdot6} & \omega_8^{-1\cdot7} \\ \omega_8^{-2\cdot0} & \omega_8^{-2\cdot1} & \omega_8^{-2\cdot2} & \omega_8^{-2\cdot3} & \omega_8^{-2\cdot4} & \omega_8^{-2\cdot5} & \omega_8^{-2\cdot6} & \omega_8^{-2\cdot7} \\ \omega_8^{-3\cdot0} & \omega_8^{-3\cdot1} & \omega_8^{-3\cdot2} & \omega_8^{-3\cdot3} & \omega_8^{-3\cdot4} & \omega_8^{-3\cdot5} & \omega_8^{-3\cdot6} & \omega_8^{-3\cdot7} \\ \omega_8^{-4\cdot0} & \omega_8^{-4\cdot1} & \omega_8^{-4\cdot2} & \omega_8^{-4\cdot3} & \omega_8^{-4\cdot4} & \omega_8^{-4\cdot5} & \omega_8^{-4\cdot6} & \omega_8^{-4\cdot7} \\ \omega_8^{-5\cdot0} & \omega_8^{-5\cdot1} & \omega_8^{-5\cdot2} & \omega_8^{-5\cdot3} & \omega_8^{-5\cdot4} & \omega_8^{-5\cdot5} & \omega_8^{-5\cdot6} & \omega_8^{-5\cdot7} \\ \omega_8^{-6\cdot0} & \omega_8^{-6\cdot1} & \omega_8^{-6\cdot2} & \omega_8^{-6\cdot3} & \omega_8^{-6\cdot4} & \omega_8^{-6\cdot5} & \omega_8^{-6\cdot6} & \omega_8^{-6\cdot7} \\ \omega_8^{-7\cdot0} & \omega_8^{-7\cdot1} & \omega_8^{-7\cdot2} & \omega_8^{-7\cdot3} & \omega_8^{-7\cdot4} & \omega_8^{-7\cdot5} & \omega_8^{-7\cdot6} & \omega_8^{-7\cdot7} \end{bmatrix}\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} \tag{13.1}$$

となる。これを

$$\mathbf{c}_8 = \frac{1}{8}\widetilde{W_8}\cdot\mathbf{f}_8$$

と表現することにしよう。つまり，離散 Fourier 変換 (12.6) は行列とベクトル積の形で表現できることになる。一般に，複素 N 次正方行列と複素 N 次元ベクトルの積は複素数の乗算が N^2 回，加算が $N^2 - N$ 回必要になる。

行列 $\widetilde{W_8}$ は，ω_8 の性質から次のような特徴を持つ。

- 第 1 列と第 1 行は全て 1 となる。
- ij 成分は，

$$\omega_8^{-(i-1)(j-1) \mod 8}$$

となる。

- 対称行列 (対角成分を軸として対称) である。

よって，$\widetilde{W_8}$ は次のような値を持つ。

$$\widetilde{W_8} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_8^{-1} & \omega_8^{-2} & \omega_8^{-3} & \omega_8^{-4} & \omega_8^{-5} & \omega_8^{-6} & \omega_8^{-7} \\ 1 & \omega_8^{-2} & \omega_8^{-4} & \omega_8^{-6} & 1 & \omega_8^{-2} & \omega_8^{-4} & \omega_8^{-6} \\ 1 & \omega_8^{-3} & \omega_8^{-6} & \omega_8^{-1} & \omega_8^{-4} & \omega_8^{-7} & \omega_8^{-2} & \omega_8^{-5} \\ \hline 1 & \omega_8^{-4} & 1 & \omega_8^{-4} & 1 & \omega_8^{-4} & 1 & \omega_8^{-4} \\ 1 & \omega_8^{-5} & \omega_8^{-2} & \omega_8^{-7} & \omega_8^{-4} & \omega_8^{-1} & \omega_8^{-6} & \omega_8^{-3} \\ 1 & \omega_8^{-6} & \omega_8^{-4} & \omega_8^{-2} & 1 & \omega_8^{-6} & \omega_8^{-4} & \omega_8^{-2} \\ 1 & \omega_8^{-7} & \omega_8^{-6} & \omega_8^{-5} & \omega_8^{-4} & \omega_8^{-3} & \omega_8^{-2} & \omega_8^{-1} \end{array}\right] \tag{13.2}$$

これだけで，計算回数が大分減ることが分かる。FFT はここから更に巧妙な再帰手続きを使って，計算回数を極限まで減らすのである。

さて，ここで自然数 0〜7 をビット反転，つまり 2 進表現を左右反転させる操作を考える。$N = 2^n$ に限れば，必ず n ビット以内で表現できるので，n ビット表現で反転を行う。今の場合は 3 ビット表現となるので，反転を行うと次のようになる。

$$\begin{aligned} 0 = (000)_2 &\rightarrow (000)_2 = 0 \\ 1 = (001)_2 &\rightarrow (100)_2 = 4 \\ 2 = (010)_2 &\rightarrow (010)_2 = 2 \\ 3 = (011)_2 &\rightarrow (110)_2 = 6 \\ 4 = (100)_2 &\rightarrow (001)_2 = 1 \\ 5 = (101)_2 &\rightarrow (101)_2 = 5 \\ 6 = (110)_2 &\rightarrow (011)_2 = 3 \\ 7 = (111)_2 &\rightarrow (111)_2 = 7 \end{aligned}$$

このビット反転に基づいて $\mathbf{f}_8$ を並べ替えたものを $\mathbf{f}_8^{\mathrm{rev}}$ と書き，これに呼応して $\widetilde{W_8}$ も列の並び替えを行ったものを W_8 と書くと，(13.1) 式は

$$\mathbf{c}_8 = \frac{1}{8} W_8 \mathbf{f}^{\mathrm{rev}}$$

となり，結局

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{bmatrix} = \frac{1}{8} \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_8^{-4} & \omega_8^{-2} & \omega_8^{-6} & \omega_8^{-1} & \omega_8^{-5} & \omega_8^{-3} & \omega_8^{-7} \\ 1 & 1 & \omega_8^{-4} & \omega_8^{-4} & \omega_8^{-2} & \omega_8^{-2} & \omega_8^{-6} & \omega_8^{-6} \\ 1 & \omega_8^{-4} & \omega_8^{-6} & \omega_8^{-2} & \omega_8^{-3} & \omega_8^{-7} & \omega_8^{-1} & \omega_8^{-5} \\ \hline 1 & 1 & 1 & 1 & \omega_8^{-4} & \omega_8^{-4} & \omega_8^{-4} & \omega_8^{-4} \\ 1 & \omega_8^{-4} & \omega_8^{-2} & \omega_8^{-6} & \omega_8^{-5} & \omega_8^{-1} & \omega_8^{-7} & \omega_8^{-3} \\ 1 & 1 & \omega_8^{-4} & \omega_8^{-4} & \omega_8^{-6} & \omega_8^{-6} & \omega_8^{-2} & \omega_8^{-2} \\ 1 & \omega_8^{-4} & \omega_8^{-6} & \omega_8^{-2} & \omega_8^{-7} & \omega_8^{-3} & \omega_8^{-5} & \omega_8^{-1} \end{array}\right] \begin{bmatrix} f_0 \\ f_4 \\ f_2 \\ f_6 \\ f_1 \\ f_5 \\ f_3 \\ f_7 \end{bmatrix} \tag{13.3}$$

となる。ここで更に

$$\begin{aligned} \omega_8^{-2} &= \omega_4^{-1} \\ \omega_8^{-4} &= \omega_4^{-2} = \omega_2^{-1} = -1 \end{aligned}$$

であることを考えると，W_8 は

$$W_8 = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & \omega_4^{-1} & -\omega_4^{-1} & \omega_8^{-1} & -\omega_8^{-1} & \omega_4^{-1}\omega_8^{-1} & -\omega_4^{-1}\omega_8^{-1} \\ 1 & 1 & -1 & -1 & \omega_4^{-1} & \omega_4^{-1} & -\omega_4^{-1} & -\omega_4^{-1} \\ 1 & -1 & -\omega_4^{-1} & \omega_4^{-1} & \omega_4^{-1}\omega_8^{-1} & -\omega_4^{-1}\omega_8^{-1} & \omega_8^{-1} & -\omega_8^{-1} \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & \omega_4^{-1} & -\omega_4^{-1} & -\omega_8^{-1} & \omega_8^{-1} & -\omega_4^{-1}\omega_8^{-1} & \omega_4^{-1}\omega_8^{-1} \\ 1 & 1 & -1 & -1 & -\omega_4^{-1} & -\omega_4^{-1} & \omega_4^{-1} & \omega_4^{-1} \\ 1 & -1 & -\omega_4^{-1} & \omega_4^{-1} & -\omega_4^{-1}\omega_8^{-1} & \omega_4^{-1}\omega_8^{-1} & -\omega_8^{-1} & \omega_8^{-1} \end{array}\right] \tag{13.4}$$

まで簡略化できる。左の上下ブロックの行列は同じものであり，右の上下ブロックの行列はちょうど符号が反転したものになっていることは容易に分かる。

13.2 高速 Fourier 変換の考え方

さてここで $N = 2^n$ が最小の時, 即ち $N = 2$ から順に積み上げて FFT のアルゴリズムの考え方を示す。

まず，$N = 2$ の時は，(13.2) と同様に

$$\begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix}$$

となる。ここで

$$\omega_2^{-1} = \exp\left(-\frac{2\pi}{2}\sqrt{-1}\right) = \omega_2 = -1$$

であったことを思い出していただきたい。この時，上の式を

$$\mathbf{c}_2 = \frac{1}{2}W_2\mathbf{f}_2$$

と書くことにする。

この場合は

$$\begin{aligned} 2C_0 &= f_0 + f_1 \\ 2C_1 &= f_0 - f_1 \end{aligned} \tag{13.5}$$

となり，複素数の乗算は不要である。これが FFT のアルゴリズムの最小単位である。

次に，$N = 2^2 = 4$ の時を考える。この時は

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & \omega_4^{-2} & \omega_4^{-3} \\ 1 & \omega_4^{-2} & 1 & \omega_4^{-2} \\ 1 & \omega_4^{-3} & \omega_4^{-2} & \omega_4^{-1} \end{bmatrix}\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

となる。

ここで添え字の 2 進表現を考える。何でそんな無茶なことを言い出すかは，後で分かる。

$$\begin{aligned} 0 = (00)_2 &\rightarrow (00)_2 = 0 \\ \underline{1 = (01)_2} &\rightarrow \underline{(10)_2 = 2} \\ \underline{2 = (10)_2} &\rightarrow \underline{(01)_2 = 1} \\ 3 = (11)_2 &\rightarrow (11)_2 = 3 \end{aligned}$$

この順にベクトル $\mathbf{f}_4$ を並べ替えると，行列も列が入れ替えられて

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \frac{1}{4}\left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-2} & \omega_4^{-1} & \omega_4^{-3} \\ \hline 1 & 1 & \omega_4^{-2} & \omega_4^{-2} \\ 1 & \omega_4^{-2} & \omega_4^{-3} & \omega_4^{-1} \end{array}\right]\begin{bmatrix} f_0 \\ f_2 \\ f_1 \\ f_3 \end{bmatrix}$$

となる。ここで

$$\omega_4^{-2} = \omega_2^{-1} = -1 \tag{13.6}$$

$$\omega_4^{-3} = \omega_4^{-1} \cdot \omega_2^{-1} = -\omega_4^{-1} \tag{13.7}$$

であるので[1](だから，N は 2 のべき乗に限定しているのだ)，上式は更に簡略化されて

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \frac{1}{4} \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & \omega_4^{-1} & -\omega_4^{-1} \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -\omega_4^{-1} & \omega_4^{-1} \end{array}\right] \left[\begin{array}{c} f_0 \\ f_2 \\ \hline f_1 \\ f_3 \end{array}\right]$$

となる。これを

$$\mathbf{c}_4 = \frac{1}{4} W_4 \mathbf{f}_4^{\mathrm{rev}}$$

と書くことにする。

ここで対角行列として

$$D_4 = \begin{bmatrix} 1 & 0 \\ 0 & \omega_4^{-1} \end{bmatrix}$$

を導入すると，上式の行列部分 W_4 は

$$\left[\begin{array}{c|c} W_2 & D_4 W_2 \\ \hline W_2 & -D_4 W_2 \end{array}\right]$$

と表現できることになる。これが，添え字の 2 進表現を考えた理由である。(⟶ 演習問題 1)

では，この計算を構成してみよう。書き下してみると

$$\begin{aligned} 4C_0 &= (f_0 + f_2) + (f_1 + f_3) \\ 4C_1 &= (f_0 - f_2) + \omega_4^{-1}(f_1 - f_3) \\ 4C_2 &= (f_0 + f_2) - (f_1 + f_3) \\ 4C_3 &= (f_0 - f_2) - \omega_4^{-1}(f_1 - f_3) \end{aligned} \tag{13.8}$$

となる。共通する部分を組み合わせてみると，図 13.1 のように行うことが分かる。これを FFT のバタフライ演算と呼ぶ。

さて，同様にして $N = 2^3$ の場合を考えると，(13.2) においては更に

$$\begin{aligned} \omega_8^{-4} = \omega_4^{-2} = \omega_2^{-1} &= -1 \\ \omega_8^{-5} &= -\omega_8^{-1} \\ \omega_8^{-6} &= -\omega_8^{-2} \\ \omega_8^{-7} &= -\omega_8^{-3} \end{aligned}$$

[1] $\omega_4 = \sqrt{-1}$ であるが，説明のために 1 のべき乗根の記号のままにしておく。

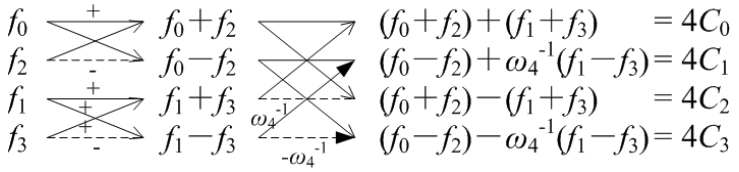

図 13.1: FFT のバタフライ演算: $N = 2^2 = 4$

となるので,

$$D_8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega_8^{-1} & 0 & 0 \\ 0 & 0 & \omega_8^{-2} & 0 \\ 0 & 0 & 0 & \omega_8^{-3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega_8^{-1} & 0 & 0 \\ 0 & 0 & \omega_4^{-1} & 0 \\ 0 & 0 & 0 & \omega_4^{-1}\omega_8^{-1} \end{bmatrix}$$

を用いると，行列 W_8 は

$$W_8 = \left[\begin{array}{c|c} W_4 & D_8W_4 \\ \hline W_4 & -D_8W_4 \end{array}\right]$$

となる。そして，W_4 の計算は (13.8) へ，更に W_2 の計算 (13.5) へと分解されていくことになるのである。こうして計算量を劇的に減らすことができる。(⟶ 演習問題 2)

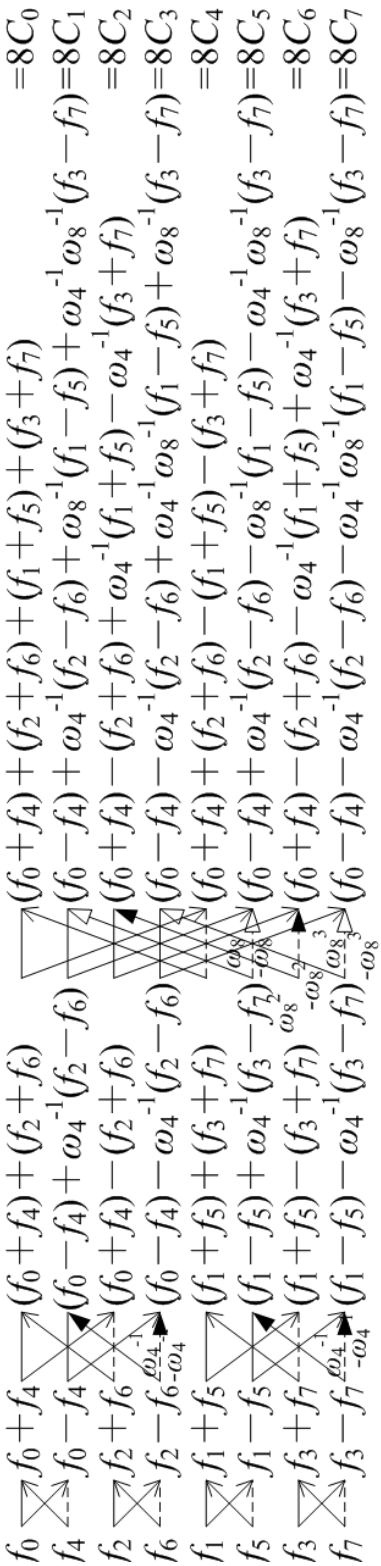

図 13.2: FFT のバタフライ演算: $N = 2^3 = 8$

13.3　高速 Fourier 変換 (FFT) のプログラム

以上述べてきたように，FFT の計算は再帰的に行うことが出来る。これを $\mathbf{f}_N^{\mathrm{rev}}$ を作らずに，$\mathbf{f}_N$ の並びをそのまま使用するようにプログラム化すると，次のようになる。

```
 1:// FFT radix-2
 2:// INPUT : complex<double> f[2^power2_n], long int start_index,
plus_index
 3:// OUTPUT: complex<double> x[2^power2_n]
 4:int fft_radix2_recursive(complex<double> x[], complex<double> f[],
long int start_index, long int plus_index, long int power2_n,
int flag_inv)
 5:{
 6:  long int total_n, m, i, index_even, index_odd, index_half[2];
 7:  complex<double> omega, tmp;
 8:  complex<double> *z, *d, *tmp_x;
 9:
10:  total_n = (long int)pow(2.0, (double)power2_n);
11:
12:  // power2_n == 0 (total_n == 1)
13:  if(total_n == 1)
14:  {
15:    x[start_index] = f[start_index];
16:    return 0;
17:  }
18:
19:  m = total_n / 2;
20:  z = new complex<double>[m];
21:  d = new complex<double>[m];
22:  tmp_x = new complex<double>[m];
23:
24:  // y_t: even
25:  fft_radix2_recursive(x, f, start_index
, plus_index * 2, wer2_n - 1, flag_inv);
26:
27:  // y_b: odd
```

```
28:  fft_radix2_recursive(x, f, start_index + plus_index
, plus_index * 2, power2_n - 1, flag_inv);
29:
30:  // d = [1, omega^1, omega^2, ..., omega^(m-1)]
31:  if(flag_inv >= 1)
32:    omega = exp(complex<double>(0, 2 * M_PI / total_n));
33:  else
34:    omega = exp(complex<double>(0, -2 * M_PI / total_n));
35:
36:  for(i = 0; i < m; i++)
37:    d[i] = pow(omega, i);
38:
39:  // z := d .* y_b
40:  for(i = 0; i < m; i++)
41:  {
42:    index_even = (start_index             ) + (plus_index * 2) * i;
43:    index_odd  = (start_index + plus_index) + (plus_index * 2) * i;
44:    z[i] = d[i] * x[index_odd];
45:    tmp_x[i] = x[index_even]; // temporarily moved
46:  }
47:
48:  // x := [y_t + z, y_t - z]^T
49:  for(i = 0; i < m; i++)
50:  {
51:    index_half[0] = start_index + plus_index * i;
52:    index_half[1] = index_half[0] + m * plus_index;
53:    x[index_half[0]]  = tmp_x[i] + z[i];
54:    x[index_half[1]]  = tmp_x[i] - z[i];
55:  }
56:
57:  // destructor
58:  delete[] z;
59:  delete[] d;
60:  delete[] tmp_x;
61:
62:  return 0;
```

```
63:}
```

この fft_fft_radix2_recursive 関数を用いて，離散 Fourier 変換のプログラムを次のように作成することができる。

```
 1:// FFT radix-2
 2:// INPUT : complex<double> f[2^power2_n], power2_n
 3:// OUTPUT: complex<double> x[2^power2_n]
 4:int fft_radix2(complex<double> x[], complex<double> f[],
 long int power2_n)
 5:{
 6:  long int total_n, i;
 7:
 8:  total_n = (long int)pow(2.0, (double)power2_n);
 9:
10:  fft_radix2_recursive(x, f, 0, 1, power2_n, 0);
11:
12:  // 1/n * x
13:  for(i = 0; i < total_n; i++)
14:    x[i] /= (double)total_n;
15:
16:  return 0;
17:}
```

なお，再帰関数を用いず，$\mathbf{f}_N^{\mathrm{rev}}$ を，添え字の bit 反転を用いて計算するようにプログラム化した例は奥村 [2] にあるので参照されたい。これは ω_N の計算において初等関数の呼び出しを一度しか行わないように工夫がしてあるものである。

演習問題

1. $N = 2^n$ ($n \in \mathbb{N}, n > 1$) である時，任意の自然数 $m < 2^n$ は n 桁の 2 進数として表現でき，これを

$$m = (m_{n-1}m_{n-2}...m_1m_0)_2 = \sum_{i=0}^{n-1} m_i 2^i$$

と書くと，$\omega_N^m = \omega_{2^n}^m$ は

$$\omega_{2^n}^m = \omega_{2^n}^{m_0} \cdot \omega_{2^{n-1}}^{m_1} \cdots \omega_4^{m_{n-2}} \cdot \omega_2^{m_{n-1}} \tag{13.9}$$

として表現できることを説明せよ。

2. FFT の計算量が $N \log_2 N$ に比例する量まで減ることを説明せよ。