FFT(高速フーリエ変換)

1. FFT とは

高速フーリエ変換 (FFT) は DFT の処理を高速に処理できるようにしたもので，DFT と FFT の処理結果は全く同じである．DFT において回転子は $W_N^m = W_N^{m\pm N}$ という性質を有していた．また，$W_N^m = W_N^{m-\ell} W_N^{\ell}$ と分解することができることも理解できよう．これらの性質を利用し，DFT における計算途中の同じ W_N をまとめるなどして，演算回数を減少させている．

DFT の一般式を偶数項と奇数項に分割してみると

$$X_k = \sum_{n=0}^{\frac{N-1}{2}} x_{2n} W_N^{2nk} + \sum_{n=0}^{\frac{N-1}{2}} x_{2n+1} W_N^{(2n+1)k} \tag{1}$$

$$= \sum_{n=0}^{\frac{N-1}{2}} x_{2n} W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N-1}{2}} x_{2n+1} W_N^{2nk} \tag{2}$$

ここで $W_N^2 = W_{\frac{N}{2}}^1$ とおくと

$$X_k = \sum_{n=0}^{\frac{N-1}{2}} x_{2n} W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\frac{N-1}{2}} x_{2n+1} W_{\frac{N}{2}}^{nk} \tag{3}$$

となり，$W_N^{\frac{N}{2}+k} = -W_N^k$ より

$$X_k = \sum_{n=0}^{\frac{N-1}{2}} x_{2n} W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\frac{N-1}{2}} x_{2n+1} W_{\frac{N}{2}}^{nk} \tag{4}$$

$$X_{\frac{N}{2}+k} = \sum_{n=0}^{\frac{N-1}{2}} x_{2n} W_{\frac{N}{2}}^{nk} - W_N^k \sum_{n=0}^{\frac{N-1}{2}} x_{2n+1} W_{\frac{N}{2}}^{nk} \tag{5}$$

となる．図 1 を参照．ここで $(k = 0, 1, \cdots, \frac{N}{2} - 1)$ である．このように，X_k の前半と後半は符号が違うだけで同じ形で表されていることが確認できる．演算量は DFT が N^2 であるのに対し FFT は $N \log_2 N$ のオーダーとなり，その演算量の違いは，N が大きくなるほど差が開くことがわかる．

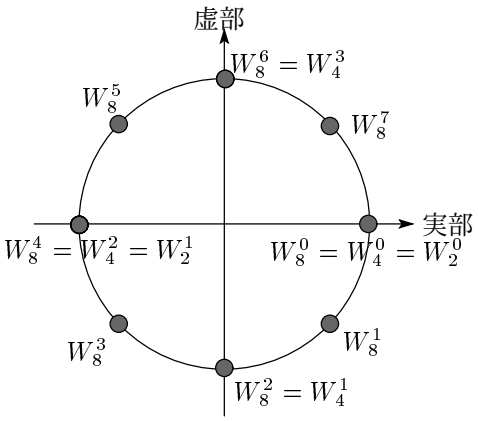

図 1: 回転子の関係 $N = 8$ のとき

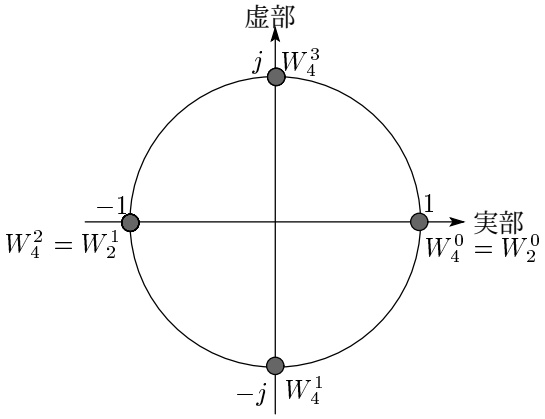

図 2: 回転子の関係 $N = 4$ のとき

FFT は上記の回転子の性質を利用したバタフライ演算およびビットリバーサルという手法で実現されている．一般に，時間間引き FFT と周波数間引き FFT が知られているが演算結果は同じであり，両者どちらとも FFT と総称される．

2. アルゴリズム

ここでは時間間引き FFT について流れを追ってみることにする．まずは，4 点，8 点 DFT の関係を確認してみよう．

2.1 4 点 DFT

図 2 は $N = 4$ 及び $N = 2$ の時の回転子を表したものであので，図 2 用いながら追ってもらいたい．
4 点 DFT を行列表記すると

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{6}$$

であった．これを x_n の偶数項と奇数項に分割すると

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \\ W_4^0 & W_4^4 \\ W_4^0 & W_4^6 \end{pmatrix} \begin{pmatrix} x_0 \\ x_2 \end{pmatrix} + \begin{pmatrix} W_4^0 & W_4^0 \\ W_4^1 & W_4^3 \\ W_4^2 & W_4^6 \\ W_4^3 & W_4^9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \tag{7}$$

と表される．ここで，式 (7) の右辺第 2 項は

$$\begin{pmatrix} W_4^0 & W_4^0 \\ W_4^1 & W_4^3 \\ W_4^2 & W_4^6 \\ W_4^3 & W_4^9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} W_4^0 W_4^0 & W_4^0 W_4^0 \\ W_4^0 W_4^1 & W_4^1 W_4^2 \\ W_4^0 W_4^2 & W_4^2 W_4^4 \\ W_4^0 W_4^3 & W_4^3 W_4^6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \tag{8}$$

となり，共通部分が現れる．従って，式 (7) は図 2 からも分かるように

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \\ W_4^0 & W_4^4 \\ W_4^0 & W_4^6 \end{pmatrix} \begin{pmatrix} x_0 \\ x_2 \end{pmatrix} + \begin{pmatrix} W_4^0 & 0 & 0 & 0 \\ 0 & W_4^1 & 0 & 0 \\ 0 & 0 & W_4^2 & 0 \\ 0 & 0 & 0 & W_4^3 \end{pmatrix} \begin{pmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \\ W_4^0 & W_4^4 \\ W_4^0 & W_4^6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \tag{9}$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ \hline 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_2 \end{pmatrix} + \begin{pmatrix} W_4^0 & 0 & 0 & 0 \\ 0 & W_4^1 & 0 & 0 \\ 0 & 0 & -W_4^0 & 0 \\ 0 & 0 & 0 & -W_4^1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ \hline 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \tag{10}$$

と表すことが出来る．このように，4 点 DFT は 2 点 DFT から構成されており，しかも上手くすれば簡単な形（計算回数を少なくできる）で構成できそうである．

2.2 8 点 DFT

少し面倒であるが，図 1 を利用しながら同様に 8 点 DFT の場合を考えてみると

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \begin{pmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ W_8^0 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ W_8^0 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ W_8^0 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{11}$$

$$= \begin{pmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} \\ W_8^0 & W_8^8 & W_8^{16} & W_8^{24} \\ W_8^0 & W_8^{10} & W_8^{20} & W_8^{30} \\ W_8^0 & W_8^{12} & W_8^{24} & W_8^{36} \\ W_8^0 & W_8^{14} & W_8^{28} & W_8^{42} \end{pmatrix} \begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{pmatrix}$$

$$+ \begin{pmatrix} W_8^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & W_8^1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & W_8^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & W_8^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & W_8^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & W_8^5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & W_8^6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & W_8^7 \end{pmatrix} \begin{pmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} \\ W_8^0 & W_8^8 & W_8^{16} & W_8^{24} \\ W_8^0 & W_8^{10} & W_8^{20} & W_8^{30} \\ W_8^0 & W_8^{12} & W_8^{24} & W_8^{36} \\ W_8^0 & W_8^{14} & W_8^{28} & W_8^{42} \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{pmatrix} \tag{12}$$

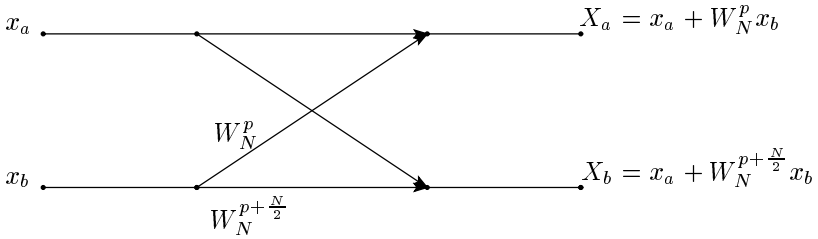

図 3: バタフライ演算の基本形

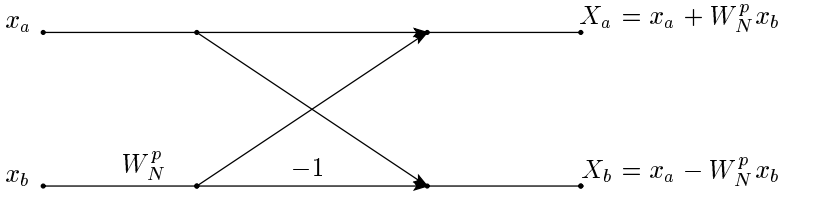

図 4: 簡素化されたバタフライ演算

と分割することが出来る．これは前述の様に

$$
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \\ \hline 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{array}\right) \begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{pmatrix}
$$

$$
+ \begin{pmatrix} W_8^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & W_8^1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & W_8^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & W_8^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -W_8^0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -W_8^1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -W_8^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -W_8^3 \end{pmatrix} \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \\ \hline 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{array}\right) \begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{pmatrix} \tag{13}
$$

と，4 点 DFT を含む形で構成されていることが確認できる．

これらのことから，N 点 DFT は $N/2$ 点 DFT で，$N/2$ 点 DFT は $N/4$ 点 DFT を含む形で構成されていることが分かる．このような関係を利用し，演算回数を減らすのに一役買っているのが，次に示すバタフライ演算と，ビットリバーサルという手法である．

2.3 バタフライ演算

図 3 が DFT におけるバタフライ演算の基本構成の流れ図を表している．2 つの入力に対し，何らかの演算を行い足し合わせる形となっている．見た目蝶のような形をしているので，このような名前が付けられた．

さて，図 3 において，任意の p における W_N^p と $W_N^{p+\frac{N}{2}}$ の関係は 4 点 DFT の図 2 又は式 (9) から式 (10) の対角行列の関係，あるいは 8 点 DFT の図 1 又は式 (12) から式 (13) の対角行列の関係から分かるように，$W_N^{p+\frac{N}{2}}$ は $-W_N^p$ であることから図 4 の様に簡素化でき，乗算回数を減らすことが出来る．

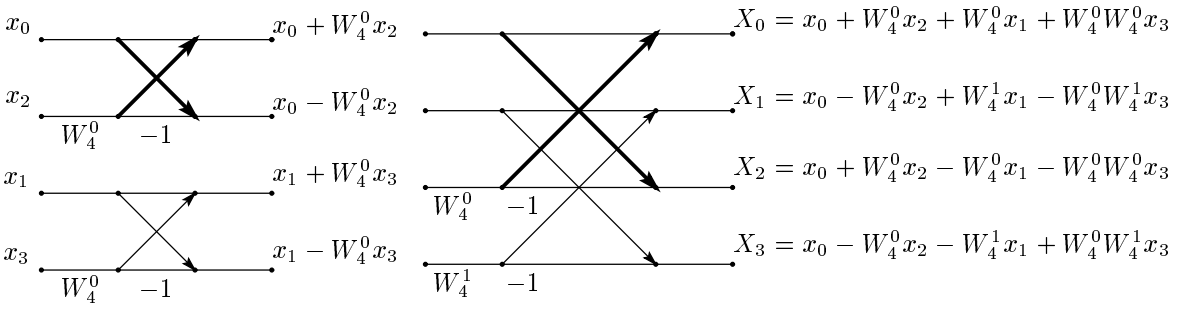

図 5: 4 点 FFT の流れ図

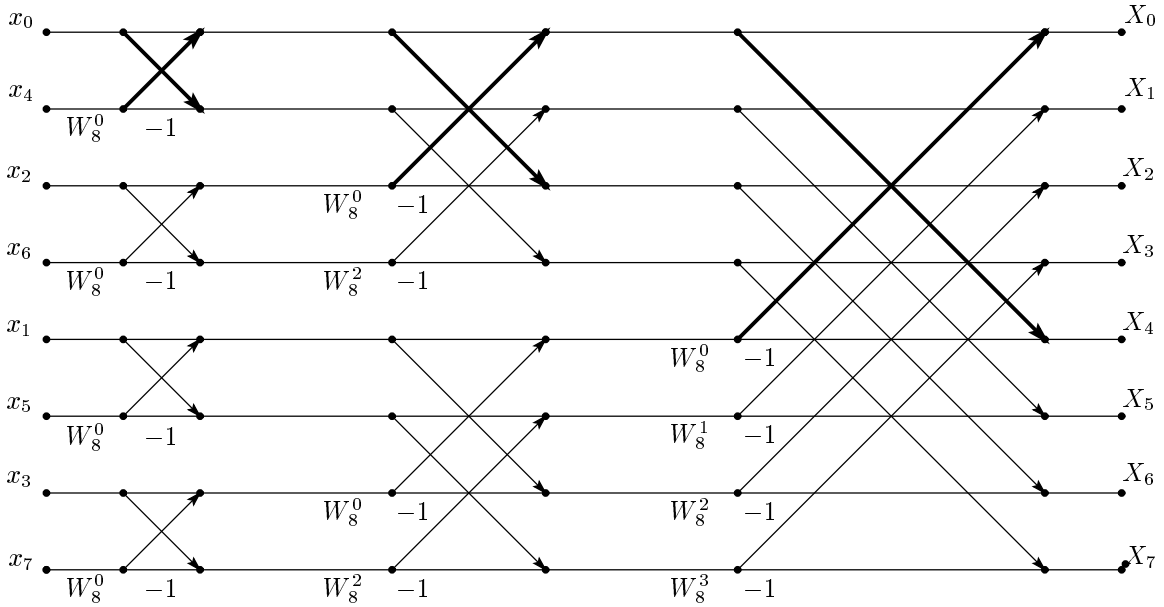

図 6: 8 点 FFT の流れ図

ここで，バタフライ演算で構成された 4 点 FFT を図 5 に示す．式 (10) と図 5 の出力の結果が等しいことを確認してもらいたい．左から入力されたデータは右方向に流れ，出力時には結果となっている．すなわち，それぞれのバタフライ演算の出力結果が次のバタフライ演算の入力につながっており，演算結果を一時的に別のメモリに待避させることなく計算が可能であることが分かる．8 点 FFT の場合を図 6 に示す．

さて，図 5，図 6 を見れば分かるように，バタフライ演算を導入することにより，入力側の並び順を都合がいいように入れ替える必要がある．これを解決するのが，ビットリバーサルである．

2.4 ビットリバーサル

バタフライ演算の導入により，入力側の順番をうまく対応するように入れ替えなければならないが，これはある決まった法則の順番となっている．ビットリバーサルという言葉からも推測できるように，実は 2 進数のビットをビットの真ん中を中心にして左右を入れ替えた順番になっており，この順番に予め並び替えておくことにより解決される．4 点および 8 点の場合の対応を表 1 に示している．

表１　ビットリバーサルの対応

4 点 DFT			
通常の場合		反転した場合	
10 進数	2 進数	10 進数	2 進数
0	00	0	00
1	01	2	10
2	10	1	01
3	11	3	11

8 点 DFT			
通常の場合		反転した場合	
10 進数	2 進数	10 進数	2 進数
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

2.5　ビットリバーサルのアルゴリズム

ビット演算およびシフトを利用してビットの反転を行う．以下にその例をあげる．

i は通常の入力 x_i に対応する番号である．i のビットを j 個右シフトし，その LSB を今度は MSB から (n-j-1) づつ左シフトして詰めていく．最終的に bit_r[i] にはビット反転した結果が保存される．ここで $N=2^n$ の関係である．

演習のため省略

一方，10 進数を基にして反転を行う方法もある．これはビット操作をしなくてもよいため，環境に依存せず利用できる．流れとしては，初期値として bit_r[0] = 0，次に bit_r[1] = bit_r[0] + N / 2, そして，bit_r[2] = bit_r[0] + N / 4，bit_r[3] = bit_r[1] + N / 4，という風に，ビット反転した結果順に MSB から対応するビットへ 1 を当てはめていくのと同様になっている．

演習のため省略

2.6　バタフライ演算によるアルゴリズム

式 (4)，(5) に対応するアルゴリズムの例を下記に示す．上述のビットリバーサルでデータを入れ替えた後に行う処理となる．

Nbig は 1.2.4... と増えていき，N/2 まで増加，Nsma は N/2...4.2.1 と減少する変数であり，これらを利用し，各バタフライ演算とデータの対応を取っている．また，inv は逆変換の時に利用する変数で，逆変換時に-1 を代入することにより得られる．ただし，最後に N で割る必要がある．

演習のため省略