このPDFは，CQ出版社発売の「改訂新版 C/C++によるディジタル信号処理入門」の一部分の見本です．
内容·購入方法などにつきましては以下のホームページをご覧下さい．
<http://shop.cqpub.co.jp/hanbai/books/30/30981.htm>

第11章

FFT（高速フーリエ変換）

フーリエ変換は多くの分野で使われており，工学では非常に重要な位置を占めるものです．計算機でフーリエ変換を行う場合は離散的フーリエ変換，略してDFT（discrete Fourier transform）を使います．

しかし，DFTの計算を定義通りに計算すると，計算量が非常に多くなるため実用的ではありません．そこで，DFTの計算量を大幅に減らして，計算のスピードを上げるためのアルゴリズムとして発表されたのがFFT（fast Fourier transform；高速フーリエ変換）のアルゴリズムです．

本章では最初にDFTについて説明した後，FFTのアルゴリズムを紹介し，そのプログラムを作成します．

11.1　DFT

（1）DFTの定義

標本化された信号を$g[n]$とすると，そのDFTである$G[k]$は次のように定義されます．

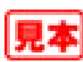
$$G[k]=\sum_{n=0}^{N-1} g[n]\exp\left(\frac{-j2\pi nk}{N}\right)$$
$$=\sum_{n=0}^{N-1} g[n]\left(\cos\left(\frac{2\pi nk}{N}\right)-j\sin\left(\frac{2\pi nk}{N}\right)\right),\quad k=0,1,\cdots,N-1 \qquad (11.1)$$

ここで，jは虚数単位で，$j=\sqrt{-1}$です．

また，ある信号のDFTを$G[k]$とすると，この$G[k]$が与えられたときに元の信号$g[n]$を次の式で計算できます．

$$g[n]=\frac{1}{N}\sum_{k=0}^{N-1} G[k]\exp\left(\frac{j2\pi nk}{N}\right),\quad k=0,1,\cdots,N-1 \qquad (11.2)$$

この操作は，逆DFT（inverse DFT，略してIDFT）と呼ばれています．



以上のことから，ある信号のDFTを求め，そのDFTの逆DFTを計算すると，最初の信号に戻ることがわかります．

DFTの式で，表記を簡潔にするため，複素指数関数の部分を次のように置き換えて表現する場合もあります．

$$W_N = \exp(-j2\pi/N) = \cos(2\pi/N) - j\sin(2\pi/N) \quad \cdots\cdots (11.3)$$

このW_Nは回転子，またはひねり因子（twiddle factor）などと呼ばれています．これを使うと式(11.1)は，

$$G[k] = \sum_{n=0}^{N-1} g[n] W_N^{nk}, \quad k = 0, 1, \cdots, N-1 \quad \cdots\cdots (11.4)$$

のように書くことができます．このW_Nという表現はFFTのところでも使います．

(2) DFTの解釈

式(11.1)で計算される$G[k]$は，$g[n]$の周波数成分に対応します．したがって，$G[k]$の“k”は周波数に対応します．その関係は，次のようになります．

信号を標本化するときの標本化周波数をf_s (Hz)，DFTの計算に使う信号の個数をNとします．そのとき，$0 \leq k \leq N/2$の範囲のkに対して$G[k]$は，$k f_s/N$ (Hz)の周波数成分になります．一方，$N/2 < k \leq N-1$の範囲のkに対して$G[k]$は，$(k-N) f_s/N$ (Hz)の周波数成分になります．この値は負にな

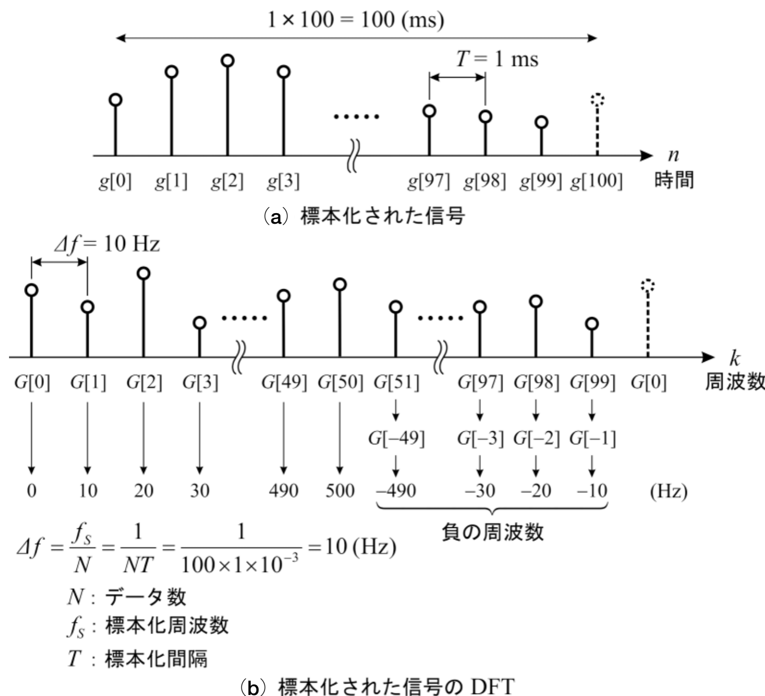


図11.1　標本化された信号と離散的周波数の関係（N=100，T=1 msの場合）

見本

るので，$N/2 < k \leq N-1$の範囲のkに対して$G[k]$は負の周波数成分[1)]に対応します．

一例として，標本化周波数が1 kHz（標本化間隔は1 ms），標本化した信号の個数を100としたときの$G[k]$と周波数の関係を**図11.1**に示します．

(3) DFTのプログラム

DFTを式(11.1)の定義に基づいて直接計算するということは，標本化された信号の個数が少ない場合を除いて，あまり行いません．しかし，後で作成するFFTのプログラムの動作を確かめるときに比較の対象として使うので，DFTを計算するプログラムを作成します．

リスト11.1（`DFT_67x.hpp`）は，浮動小数点演算でDFTの計算を行う関数です．この関数`DFT_67x()`は複素信号に対してもDFTを計算できるようにするため，第1引数`x`は`Complex`型の配列になっています．また，計算結果は一般に複素数になるので，計算結果に対応する第2引数`y`も`Complex`型の配列になっています．

通常，C++言語では標準テンプレート・クラス・ライブラリとして複素数のためのクラスが提供されています．しかし，C6000シリーズのDSP用のC/C++コンパイラでは，複素数のためのクラスは提供されていません．そこで，ここでは筆者が以前に作成した複素数を扱うためのクラス・ライブラリ[2)]を使っています．インクルードしている“`MyComplex.hpp`”がこのライブラリで，**リスト11.1**で使っている`Complex`型は，このクラス・ライブラリの中で定義されているものです．このクラス・ライブラリについては，付録Dに簡単な説明を書いたので，そちらを参照してください．

リスト11.1　複素データに対するDFT（C6713 DSK用）（`DFT_67x.hpp`）

```
//----------------------------------------------------------
//        複素データに対するDFT（C6713 DSK用）
//            浮動小数点演算
//            作成者，著作権者： 三上直樹 2008/03/14
//----------------------------------------------------------

#ifndef MK_DFT_67x

#include "MyComplex.hpp"

void DFT_67x(const Complex x[], Complex y[], int nDFT);

// 定義に従ったDFTの計算
void DFT_67x(const Complex x[], Complex y[], int nDFT)
{
    const float PI2_N = -6.28318531f/float(nDFT);

    for (int k=0; k<nDFT; k++)
    {
        y[k] = 0.0;
        for (int n=0; n<nDFT; n++)
            y[k] = y[k] + x[n]*Complex(std::cos(PI2_N*n*k),
                                       std::sin(PI2_N*n*k));
    }
}

#define MK_DFT_67x
#endif
```


見本

リスト11.2（DFT_64x.hpp）は，固定小数点演算でDFTの計算を行う関数です．この関数DFT_64x()も，複素信号に対してもDFTを計算できるようにしました．第1，2引数のxReとxImが，それぞれ信号の実部と虚部に対応するshort型の配列になっています．また，計算結果も一般に複素数になるので，計算結果は，第3，4引数のyReとyImが，それぞれ計算されたDFTの実部と虚部に対応するshort型の配列になっています．

このプログラムの中では，計算は実部と虚部に分けて行いました．計算の中で，sin，cosの値が必要になりますが，この計算には第8章で作成したCORDIC法を使ってsin，cosの値を同時に計算する関数sin_cos_cordic()を使いました．この関数を使えば，与えた引数をxとすると$\cos(\pi x/2)$，$\sin(\pi x/2)$の値を計算し，結果は小数点以下に14ビットを割り当てた数値として得られます．

なお，固定小数点演算の場合，式(11.1)のままではオーバフローの発生する可能性があるので，この関数DFT_64x()では，次の式で計算しました．

リスト11.2　複素データに対するDFT（C6416 DSK用）（DFT_64x.hpp）

```
//----------------------------------------------------------
//        複素データに対するDFT（C6416 DSK用）
//            固定小数点演算
//            作成者，著作権者： 三上直樹 2008/03/14
//----------------------------------------------------------
#include "CORDIC_sin_cos.hpp"

#ifndef MK_DFT_64x

void DFT_64x(const short xRe[], const short xIm[],
             short yRe[], short yIm[], int N);

//                N-1
//   G[k] = (1/N)Σg[n]*exp(-j2πnk/N)を計算
//                n=0
void DFT_64x(const short xRe[], const short xIm[],
             short yRe[], short yIm[], int N)
{
    const short Ninv = int(65536)/N;
    short sx, cx;
    int tRe, tIm;

    for (int k=0; k<N; k++)
    {
        tRe = 0;
        tIm = 0;
        for (int n=0; n<N; n++)
        {
            sin_cos_cordic(cx, sx, -n*k*Ninv);
            tRe = tRe + Round14(xRe[n]*cx - xIm[n]*sx);
            tIm = tIm + Round14(xRe[n]*sx + xIm[n]*cx);
        }
        yRe[k] = Round16(tRe*Ninv);
        yIm[k] = Round16(tIm*Ninv);
    }
}

#define MK_DFT_64x
#endif
```


見本

$$G[k]=\frac{1}{N}\sum_{n=0}^{N-1}g[n]\left(\cos\left(\frac{2\pi nk}{N}\right)-j\sin\left(\frac{2\pi nk}{N}\right)\right),\quad k=0,1,\cdots,N-1 \quad\cdots\cdots(11.5)$$

式(11.1)との違いは，$\sum$の前に$1/N$がある点だけです．

11.2 FFT

DFTの計算量はデータ数の2乗に比例する量となるため，データ数が多い場合，実用的には問題があります．そこで，計算量を減らすために考えられたのがFFTアルゴリズムです．

FFTを使うとDFTの計算量を大幅に削減でき，計算を高速に実行することができます．私たちが通常よく使うのは，データ数が2^M(M：整数)になるときに適用できるFFTアルゴリズムで，これを2を基底とする(radix-2)アルゴリズムといいます．また，FFTのアルゴリズムには色々なものがありますが，ここでは周波数間引き(decimation-in-frequency)アルゴリズムについて説明します．

(1) 2を基底とする周波数間引きFFTアルゴリズム

図11.2には，$N=2^4=16$に対する周波数間引きFFTアルゴリズムを示します．**図11.2(a)**は全体のようすです．**図11.2(b)**はFFTを行うときの基本的な演算単位で，バタフライ演算(butterfly operation)と呼ばれています．

処理の手順は，次のようになります．

第1段目では，データを前半と後半の二つのグループに分け，$N/2=8$だけ離れたところにあるデータ同士，つまり$g[0]$と$g[8]$，$g[1]$と$g[9]$，・・・・・，$g[7]$と$g[15]$でバタフライ演算を行います．また，○の中の数値はバタフライ演算で使われている回転子W_{16}^{k}の指数kに対応し，その値は上から順に0, 1, 2, 3, 4, 5, 6, 7になっています．

第2段目では，まず第1段目で分けられた二つのグループを，さらにそれぞれ二つのグループに分け，全体では四つのグループに分けます．次に，$N/4=4$だけ離れたところにあるデータ同士でバタフライ演算を行います．また，○の中の数値は，上から順に0, 2, 4, 6, 0, 2, 4, 6になっています．

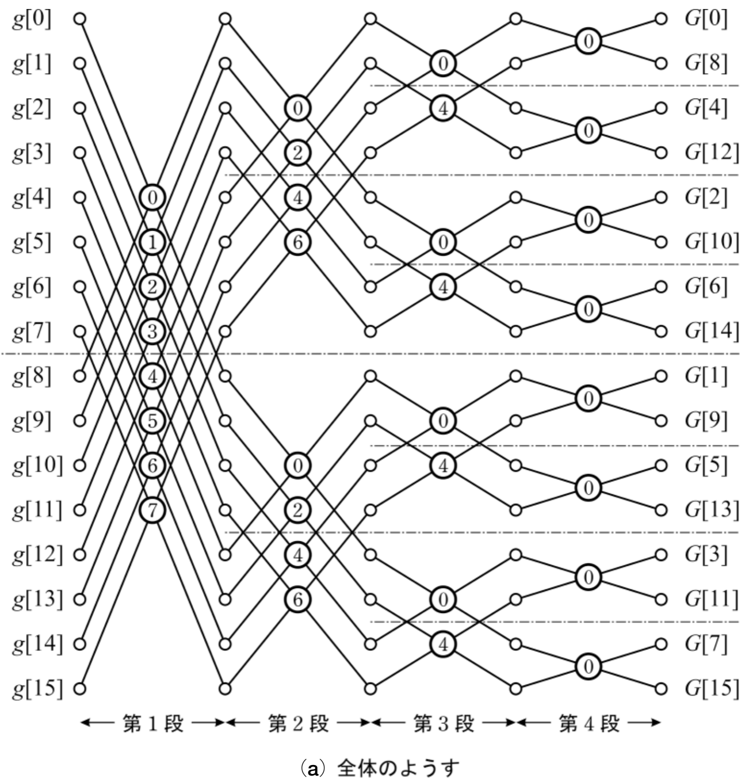
同様に，第3段目，第4段目も同じように処理を行っていくと，最後の段の出力にDFTの結果が現れます．

この例では，全体の処理は4段で構成されていますが，データ数が2^Mのときは全体で，M段で構成されることになります．

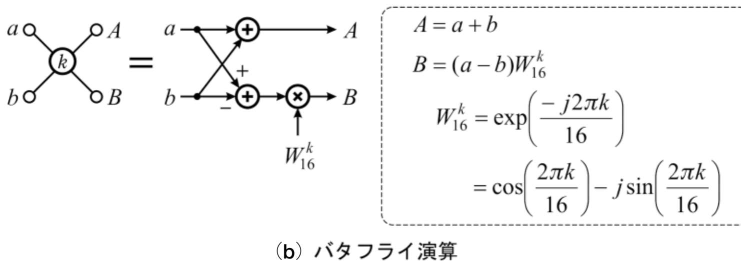
ところで，**図11.2**のデータの並び方には注意が必要です．入力データ$g[n]$は，[ ]の中の数字の小さいものから大きいものへと順に並んでいるので，何も注意する必要はありません．一方，求められた$G[k]$は，[ ]の中の数字の小さいものから順には並んでおらず，このような並び方をビット逆順(bit reversal)の並び方と呼んでいます．したがって，このアルゴリズムを使う場合は，ビット逆順の順番に並んでいるデータ$G[k]$を普通の順番に並べ換える必要があります．

ビット逆順の数とは，次のような数です．ある数がMビットの2進数で，次のように表されるものとします．ただし，b_k($k=0,\ 1,\ \cdots,\ M-1$)は，0または1とします．

見本



(a) 全体のようす



(b) バタフライ演算

図11.2　周波数間引きによる2を基底とするFFTアルゴリズム(N=16)

$$b_{M-1} \cdots b_2\ b_1\ b_0$$

このとき，ビットの前後の順番を入れ換え，次のようにした数がビット逆順の数です．

$$b_0\ b_1\ b_2 \cdots b_{M-1}$$

N=16の場合の通常の順番とビット逆順の順番の対応を，**表11.1**に示します．

ビット逆順の数は，ビットを入れ替えることで作ることができますが，それをそのままプログラムにするのは効率の良い方法ではありません．そこで，本章でプログラムを作成する際は，**図11.3**に示す方法でビット逆順の表を作成します．

見本