高速フーリエ変換 (FFT) について

高 橋 秀 俊†

高速フーリエ変換 (Fast Fourier Transform 略して FFT) というものが世にあらわれてから数年[1]，それが各方面にもたらしている影響は，革命的ともいえる．筆者も実はフーリエ変換については昔から興味をもち，PC-1 を使っていた頃もフーリエ変換をなるべく速く計算する方法をいろいろ考え，ライブラリー・プログラムを作ったりして，それは実際に脳波の解析に利用されたりしたものであるが，その頃から，ちょっとの工夫で速度がすぐに2倍になるというようなことがいろいろあるので，一体どこまで速くできるかが一つの課題になっていた．その後そのままになっていたが，東大大型計算機センターができてから，またフーリエ変換のライブラリールーチンをどのように作るかを考えはじめたところ，どうも際限なく速くできるようで，結局 N 項のフーリエ変換が $N\log_2 N$ ぐらいの乗算回数でできるらしいことがわかった．これが本当なら，$N=2^{10}$ としたとき $\log_2 N=10$ だから，普通なら $N^2\approx 10^6$ またはその半分か 1/4 ぐらいの乗算が必要なのが，10^4 ぐらいの乗算ですむということになる．ちょうどその頃，何かの紹介記事に，同じようなことが書かれているのを見て，さては一足先にやられたかと思ったが，とにかく自分の方法で Fortran のプログラム[2]をつくって，センターの HITAC 5020 (当時はまだ 5020 E はなかった) にかけて見ると，2^{10} 項のフーリエ変換がたったの2秒でできたので我ながら驚いた．1966 年の夏頃である．そんなわけで，残念ながら，FFT の創始者 Cooley と Tukey に対してプライオリティを主張することはできないが，全く独立に考えたのは事実である．この考えに至った道程も彼等は実験計画法のようなことかららしいが，私自身はまた違った考えから入って行ったので[3]，そして今日でも FFT に関するやさしい解説はあまりないようなので，私自身の立場から，FFT の考え方とその応用等について，一通り述べて見たいと思う．

FFT 以前の発展

† 東京大学理学部

私自身が FFT を思いついた道程にも二つの経路がある．いずれにせよ，フーリエ変換は各種の統計データの分析，動的システムの解析などで極めて重要なものであり，しかも計算が面倒なので，昔から，少しでも計算を簡単にするため，さまざまな工夫がこらされている．ここでいうフーリエ変換とは，有限離散フーリエ変換 (finite discrete Fourier transform) と呼ばれているもので，$(x_0, x_1, \cdots, x_{N-1})$ という N 個のデータに対して

$$X_j=\sum_{k=0}^{N-1} x_k \varepsilon^{-jk} \quad (j=0,1,\cdots,N-1) \qquad (1)$$

(ただし $\varepsilon=e^{2\pi i/N}$)

を計算することである．(1)を真正直に計算すれば，ε の整数ベキはあらかじめ計算してあるとしても，約 N^2 回の複素乗算と，同じくらいの数の加算が必要である．実際には (1) の式の特殊な性質を使って，計算量を少くすることがいろいろと可能である．たとえば，(1)は ε^j に対する一つの多項式であることに注目すれば，多項式を計算するよく知られた Horner 形式 (掛けては加えるやり方) を使うことにより，乗算の数は別に減らないが，ε^j を一度計算すればその各ベキ ε^{jk} を全部計算したり表を引いたりする必要がなくなる．Horner のアルゴリズムは一階の線形差分方程式を解くことと考えられるが，この考えを進めると，2階の線形差分方程式

$$u_{r+1}=2\cos\frac{2r\pi}{N}u_r-u_{r-1}+x_{N-r} \qquad (2)$$

を初期条件 $u_1=u_0=0$ で解き進んで最後に得られる u_N, u_{N+1} から，簡単な関係

$$X_j=u_{N+1}-\varepsilon^{-1}u_N,\quad X_{N-j}=u_{N+1}-\varepsilon u_N \qquad (3)$$

で X_j と X_{N-j} の両方を求めることができる．この方法によると，乗算の数は一挙に半分の $N^2/2$ になることがわかる．これは PC-1 のライブラリーに使われた方法である[4]．

もっと簡単な話として，N が偶数 ($N=2M$) であるなら

$$\varepsilon^M=-1$$

だから

$$X_{j+M}=\Sigma(-1)^k x_k \varepsilon^{-jk}.$$

そこで (1) を偶数番目の項と奇数番目の項とに分けて

$$X_j^0=\sum_{k=0}^{M-1} x_{2k}\varepsilon^{-2jk},\quad X_j^1=\sum_{k=0}^{M-1} x_{2k+1}\varepsilon^{-j(2k+1)} \quad (4)$$

とすると

$$\left.\begin{array}{l} X_j=X_j^0+X_j^1 \\ X_{j+M}=X_j^0-X_j^1 \end{array}\right\}$$

となり，乗算の数は (4) の二つの式で (定った j について) N 回であるが j は $N/2=M$ 個の値しかとらないから，全体でやはり $N^2/2$ 回で，直接計算の半分に減ったことになる．同じようにして，Nが4の倍数なら，$N=4M$ としたとき

$$\varepsilon^M=i,\ \varepsilon^{2M}=-1,\ \varepsilon^{3M}=-i$$

であり，したがって X_j, X_{j+M}, X_{j+2M}, X_{j+3M} を上と同じような方針で，項を4つ目毎にとった部分和に分けて計算すると，同じ和がどれにも共通に出て，乗算の回数は $N^2/4$ になってしまう．

同じようにして，N が8の倍数なら，$N=8M$ として同じやり方ができそうであるが，今度は

$$\varepsilon^M=e^{\pi i/4}=\frac{1+i}{\sqrt{2}}$$

であるため，部分和を加え合わせるところに $1/\sqrt{2}$ による乗算が必要になって，多少面倒になる．そんなことから，これ以上は進められないように見える．(実はあとほんの一歩で FFT に到達するところだったのであるが．)

2段階法 (直積分解)

FFT の芽ばえは，まだほかにもあった．"電子計算機以前" の手計算によるフーリエ変換 (調和解析) によく使われた方法で，72 項の変換を 72=8×9 であることに着目して2段階に分けて，8項の変換を9回と，9項の変換を8回行なう方法である．ここで，8と9とは互に素であるから，任意の整数 j は

$$j\equiv 8j_1+9j_2 \pmod{72} \quad (5)$$

(ただし $0\le j_1\le 8$, $0\le j_2\le 7$)

のようにあらわすことができる．k についても

$$k\equiv 8k_1+9k_2 \pmod{72} \quad (5')$$

(ただし $0\le k_1\le 8$, $0\le k_2\le 7$)

と書く．そこで

$$X_j=X_{8j_1+9j_2}=\sum_{k_1=0}^{8}\sum_{k_2=0}^{7} x_{8k_1+9k_2}\varepsilon^{(8j_1+9j_2)(8k_1+9k_2)}$$

$$=\sum_{k_1}\sum_{k_2} x_{8k_1+9k_2}\varepsilon^{64j_1k_1+81j_2k_2} \quad (6)$$

ただし，$\varepsilon^{72}=1$ であることを利用している．

(6) はこれを2段階に分けて

$$\xi_{k_1,j_2}=\sum_{k_2=0}^{7} x_{8k_1+9k_2}\varepsilon_2^{j_2k_2} \quad (7)$$

$$X_{8j_1+9j_2}=\sum_{k_1=0}^{8}\xi_{k_1,j_2}\varepsilon_1^{j_1k_1} \quad (8)$$

ただし　$\varepsilon_2=\varepsilon^{81}=\varepsilon^9=e^{2\pi i/8}$, $\varepsilon_2^8=1$

$\varepsilon_1=\varepsilon^{64}=\varepsilon^{-8}=e^{2\pi i/9}$, $\varepsilon_1^9=1$

のように書くことができる．(7)，(8)ともに式の数は 72 個あるが，一つの式の項数が(7)は8項，(8)は9項なので，直接 72 項の和を計算するのとくらべて乗算の数は大幅に減ることになる．実際には8項の方の変換では $\varepsilon_1=1+i/\sqrt{2}$ だから，掛け算は $1/\sqrt{2}$ による乗算だけであり，しかも共通なものが多いので，もっと得になる．

上の方法は x_k を9番目毎にとった8個のデータによるフーリエ変換を9組求め，次にその各組の同じ次数の項を1組としたデータで9項のフーリエ変換を求めるということである．別の言い方をすれば $x_{8k_1+9k_2}$ を x_{k_1,k_2} のように2次元の配列と考えるとき，これに対して，(結晶解析でやるような) **2次元フーリエ変換**を，2段階に分けて，1次元ごとにフーリエ変換したことである．数学の言葉でいえば，変換 (1) を低次元の変換の**直積**としてあらわしたことである．しかし，こういう直積分解は上の例なら 8×9 というように，Nが**互に素**な因子に分けられるときにしか可能でないことに注意したい．

高速フーリエ変換 (FFT) の原理

さて，N が $N=N_1N_2$ と分解できるとき，N_1 と N_2 が互に素でない場合も，直積の形ではないが，同じように2段階に分けて計算することにより計算量を減らせることがわかった．これが FFT アルゴリズムの基本である．

今度の場合は j, k を

$$j=N_2j_1+j_2$$

$$k=k_1+N_1k_2$$

ただし

$$\left.\begin{array}{l} j_1,\ k_1=0,\ 1,\ \cdots,\ N_1-1 \\ j_2,\ k_2=0,\ 1,\ \cdots,\ N_2-1 \end{array}\right\} \quad (9)$$

のようにあらわす．これなら N_1 と N_2 が互に素である必要はない．すると

$$X_j=\sum_{k=0}^{N-1} x_k\varepsilon^{-jk}$$

$$=\sum_{k_1=0}^{N^1-1}\sum_{k_2=0}^{N_2-1}x_{k_1+N_1k_2}\varepsilon^{-(N_2j_1+j_2)(k_1+N_1k_2)}$$

$$=\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}x_{k_1+N_1k_3}\varepsilon^{-(N_1j_2k_2+N_2j_1k_1+j_2k_1)}$$

$$=\sum_{k_1=0}^{N_1-1}\left(\sum_{k_2=0}^{N_2-1}x_{k_1+N_1k_2}\varepsilon_2^{-j_2k_2}\right)\varepsilon_1^{-j_1k_1}\varepsilon^{-j_2k_1} \quad (10)$$

ただし，$\varepsilon_1=\varepsilon^{N_2}$，$\varepsilon_2=\varepsilon^{N_1}$.

ここで (10) の括弧の中は j_2，k_1 だけに関係していて j_1 にはよらないので，N 個の式を計算すればよい．これを

$$X_{j_2,k_1}{}^{(1)}=\sum_{k_2=0}^{N_2-1}x_{k_1+N_1k_2}\varepsilon_2^{-j_2k_2} \quad (11)$$

と書くと

$$X_j=\sum_{k_1=0}^{N-1}X_{j_2,k_1}{}^{(1)}\varepsilon^{-j_2k_1}\varepsilon_1^{-j_1k_1} \quad (12)$$

ここで

$$\tilde{X}_{j_2,k_1}{}^{(1)}=X_{j_2,k_1}{}^{(1)}\varepsilon^{-j_2k_1} \quad (13)$$

と書けば，(12) は $\tilde{X}_{j_2,k_1}{}^{(1)}$ に対する N_1 項のフーリエ変換（を N_2 組ならべたもの）である．したがって，$N=N_1N_2$ 項のフーリエ変換は次のような3段階に分けられる．

（a） N_2 項のフーリエ変換を N_1 組行なう．((11)式に対応)

（b） (a) の結果 $X_{j_2,k_1}{}^{(1)}$ に $\varepsilon^{j_2k_1}$ を乗ずる ((13)式に対応)（位相調整）

（c） (b) の結果に対し N_1 項のフーリエ変換 ((12) 式) を N_2 組行なう．

（b）の段階が間にはさまるのが直積の場合と違うところである．

そこで，たとえば N が偶数であれば，$N_1=2$，$N_2=N/2$ とすると，（c）の段階は2項のフーリエ変換だから，$\varepsilon_1=-1$ であり，したがって一つも乗算はいらない．（b）の段階では，$k_1=0$，1 のうちの $k_1=1$ の場合にだけ ε^{j_2} による乗算がいるから，$N/2$ 回の乗算がいる．

（a）の段階については，$N_1=N/2$ がまた偶数であれば，つまり N が4の倍数であれば，再びこれを3段階に分けて行なうことができる．このようにして，2で割れる限り，同じやり方でどこまでも変換の次数を下げて行けるので，$N=2^m$ の場合には，結局のところ全部を2項のフーリエ変換と位相調整だけで行なうことができることになるのである．これが Cooley-Tukey および私の FFT のやり方である．ここで乗算は位相調整の部分だけであるので，全体で乗算の回数はおよそ $mN/2=N/2\log_2N$ 回ですむ．

以上，やや形式的に述べたが，要するに前の 8×9 項の場合と同じように，総和の式の中で N_1 ずつとびとびに取った部分和をまず求めることにより，N_1 個の j に対して共通な式が出てくることを利用したまでである．しかも N_2 と N_1 に共通因子があってもよいから，$N=2^m$ として各段階で N_1 を2とすることによって，変換の部分の計算を簡単にすることができるのである．

FFT のいろいろの改良

以上の説明からも想像がつくように，FFT アルゴリズムにはいろいろな変形がある．まず N_1 は2以外の数，たとえば3とか4とかでもよいことは明らかで，したがって N は2のベキでなくても，なるべくたくさんの素因子を含むものであれば何でもよい．特に，$N_1=4$ としたときには，$\varepsilon_1=i(=\sqrt{-1})$ であるから，（c）の段階で乗算がいらないことは $N_1=2$ の場合と同じである．そのかわり，位相調整のところで $k_1=1$, 2, 3 に対して $\varepsilon^{j_2k_1}$ が必要なので，$3/4\cdot N$ 回の乗算がいる．それでも全体としての乗算の数は約 $3/8\cdot N\log_2N$ となり，$N_1=2$ の場合より有利になる，同じようにして，$N_1=8$，$N_1=16$ の場合も検討され，適当にプログラムをつくれば，$N_1=4$ よりさらに有利になることが知られている[5]．

逆に $N_1=N/2$，$N_2=2$ とすることができる．こうすると，前述の（a）の段階が2項フーリエ変換となって乗算がなくなる．この場合は（c）の段階の方をさらに分解して進むことになる．こうして得られるアルゴリズムは前述のものとよく似ているが位相調整のところが少し違うものであって，Sande-Tukey アルゴリズムと呼ばれる[6]．

実数データに対するフーリエ変換

今までの話では，データ x_j も結果 X_j もすべて複素数として扱って来た．式の中にあらわれる ε のベキが複素数だから，仮に x_j が実数であっても途中であらわれる量は複素数になるからである．そこで実際のプログラムも FORTRAN の複素演算を使っているものが多いようである．しかし，複素数同士の乗算は実数の乗算を4回やることになるから，前に得た $N/2\times\log_2N$ 回の乗算というのは，実演算の数で数えれば実は $2N\log_2N$ 回ということになる．

ところで実際にはデータ x_j は実数である場合が多

い．その場合に複素数データにも使えるアルゴリズムをそのまま使うのは損であろうことは直観的にも明らかである．実際，そういう場合，二つの全く無関係なデータ x_j, y_j があるとき，これから複素数のデータ

$$z_j = x_j + iy_j \tag{14}$$

をつくり，これに対してフーリエ変換を行った結果を Z_j とすると，x_j, y_j のフーリエ変換 X_j, Y_j との間には

$$Z_j = X_j + iY_j \tag{15}$$

の関係がある．一方，実数値に対するフーリエ変換に対しては

$$X_{N-j}^* = X_j, \quad Y_{N-j}^* = Y_j \tag{16}$$

のような関係が成り立つので，

$$Z_{N-j}^* = X_{N-j}^* - iY_{N-j}^* = X_j - iY_j$$

から

$$X_j = \frac{1}{2}(Z_j + Z_{N-j}^*), \quad Y_j = \frac{1}{2i}(Z_j - Z_{N-j}^*) \tag{17}$$

のようにして X_j, Y_j が求められる．つまり(複素数の) FFT を一度行なうことによって，二組の実数に対するフーリエ変換が同時に求められることになる．

しかし，二つのデータを組にしなければならないのはあまり便利といえないので，一つの実のデータ x_j に対して FFT アルゴリズムを直接使って，半分の計算量($N\log_2 N$ の実乗算) で X_j を求めることも考えられる[7]．それには，データが実数である場合，最終結果が (16) のように共役複素数が対になってあらわれるだけでなく，すべての中間結果が，実数かそうでなければ必らず互に共役な対で存在するということに注目する．つまり，そのような式の対に対しては，一方を計算すれば，他方はあらためてやるに及ばないので，計算量は半分ですむのである．そのようにして，実のデータに対するFFTのアルゴリズムが得られる．筆者が最初に (1966 年) つくったのもそういう実数 FFT である．これについては Bergland[7] (1968) に対してプライオリティを主張できる．

なお，$N/2$ 個の複素データに対するアルゴリズムを利用して N 個の実データを変換する方法もある．それには，元のデータの偶数番目を実部，奇数番目を虚部にもつ $N/2$ 個のデータ

$$z_j = x_{2j} + ix_{2j+1}$$

をつくり，これに対して FFT を行なうのである．こうして得た結果から

$$X_j = \frac{1-i\varepsilon^j}{2} Z_j + \frac{1+i\varepsilon^j}{2} Z_{N/2-j}^* \quad (\varepsilon = e^{2\pi i/N})$$

によって X_j が求まる．

さらに，データが実数でかつ対称性

$$x_{N-j} = x_j$$

をもつときは変換の結果も実数(cosine 変換)となり，また反対称性

$$x_{N-j} = -x_j$$

のときは結果は純虚数 (sine変換) となる．それらの場合には計算量はさらに半分になる．そのようなアルゴリズムについても最近あちこちで発表されているようである．

記憶場所の問題

以上のようなアルゴリズムを実際にプログラムにする場合は，データや途中の数値の収納場所が問題になる．一般に FFT は低次元のフーリエ変換の繰り返しであるから，各要素の変換について，変換後の結果を変換前のデータのあった場所に入れるようにすれば，余分な記憶場所はほとんどいらないわけである．しかし，そうやると，最終結果 X_j は最初のデータ x_k とは全く違う妙な順序にならんでしまう．普通の2項変換に分解する方法の場合，j を m 桁の2進数としてあらわし，これを逆順に (右の方から) 2進数として読んだ数値を j^* と書くならば，X_j は x_{j^*} のあった場所に入るのである (**ビット反転**)．たとえば $N=8$ とすると

$$X_0 \quad X_4 \quad X_2 \quad X_6 \quad X_1 \quad X_5 \quad X_3 \quad X_7$$

の順になる．(第1図参照.) そこで，あとで普通の順にならべかえる操作が必要になる．もっとも，応用によってはフーリエ変換の結果にまたフーリエ変換 (逆変換) を行なう場合があり，そのときは，2番目の逆変換のプログラムを，このような特殊のならび方をし

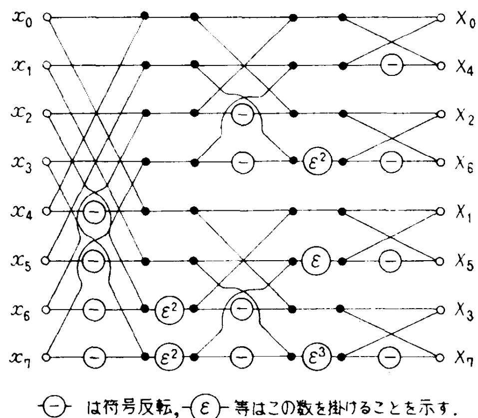

図1 $N=8$ の場合の信号流れ図

たデータに対するように組んでおけば，逆変換の結果は元のならび方に戻ってしまうので，ならべかえの操作は省略できる．

応用——周波数分析

フーリエ変換がこのように速く（データ数の数倍程度の乗算回数で）実行できるとなると，さまざまな応用が考えられてくる．まず周波数分析（調和解析）の目的に使えることはいうまでもない．これはフーリエ変換そのものであるので，従来用いられていたアナログ的方法からディジタル法に切りかえる傾向が顕著である．また，雑音的な信号の周波数分析には，従来はまず自己相関関係を求めて，それをフーリエ変換してパワースペクトルを得るのが普通であったが，FFT はむしろ相関関数の計算より速いので，信号を直接フーリエ変換して，その絶対値2乗を適当に平滑してパワー・スペクトルを求めるようになった．しかし，信号解析にディジタルな方法を使うときは，時間的にサンプルしたデータを使うこと，有限の時間のデータを使うこと等に起因するいろいろな誤差があらわれるので，それ相応の注意が肝要である．しかしそれはディジタル的方法を使う場合の一般的な問題であり，(1) という式を計算するのに FFT を使ったためにおこる問題ではない．(1) の計算法としては，FFT はむしろ他のどの方式に比較しても誤差（丸め誤差）が少ないことが認められている．

畳込み演算と相関関数

それほど，自明でなくて重要な応用は，**畳込み演算** (convolution)，つまり二つの数列 a_j, b_j から

$$c_j=\sum_k a_k b_{j-k} \tag{18}$$

を求める計算である．いま a_j, b_j のフーリエ変換を A_j, B_j とすると，c_j のフーリエ変換 C_j は

$$C_j=A_jB_j \tag{19}$$

のように積になることが知られている．また c_j は C_j から逆変換

$$c_j=\frac{1}{N}\sum_{k=0}^{N-1} C_k\varepsilon^{jk} \tag{20}$$

によって求められるから，フーリエ変換を3回使えば (18) が計算できるのである．(18) はこれ以上簡単にできそうもない簡単な式に見えるが，FFT を使うと，データの数が多いときはこんなにまわりくどいことをやっても直接計算より速いのである．

(18) に似た式

$$\phi_j=\sum_k a_k b_{k-j} \tag{21}$$

は a_j, b_j の**相互相関関数**と呼ばれて，統計的な問題で重要である．これを (18) とくらべると，b の添字の符号が逆になっただけであるので，ϕ_j のフーリエ変換 Φ_j は

$$\Phi_j=A_jB_j^* \tag{22}$$

で与えられることになる．

ところで，実際には，フーリエ変換を考えている場合，数列 a_j, b_j 等は，N を周期として周期的にくりかえすもののように考えている．したがって，(20) によってつくった畳込みの結果も，正確にはいわゆる**循環畳込み** (cyclic convolution)，つまり

$$c_j=\sum_{k=0}^{j} a_k b_{j-k}+\sum_{k=j+1}^{N-1} a_k b_{N+j-k} \tag{23}$$

になる．ところが右辺の第2項の方は普通は余分なもので，第1の総和だけがほしいのである．

この問題はしかし簡単に解決される．いまデータが，a_j, b_j 共に N 個あったとすると，それぞれに 0 というデータを N 個つけたして，それぞれ $2N$ 個のデータとして，$2N$ 項のフーリエ変換を行なうのである．すなわち

$$\left.\begin{aligned}\tilde{a}_j&=\begin{cases}a_j, & j=0,\ 1,\ \cdots,\ N-1\\ 0, & j=N,\ \cdots,\ 2N-1\end{cases}\\ \tilde{b}_j&=\begin{cases}b_j, & j=0,\ 1,\ \cdots,\ N-1\\ 0, & j=N,\ \cdots,\ 2N-1\end{cases}\end{aligned}\right\} \tag{24}$$

そうして $\tilde{a}_j$, $\tilde{b}_i$ に対する（循環）畳込みを行なえば，Nだけの“安全距離”のおかげで“裏側での重なり”が避けられて（図2参照），

$$c_j=\begin{cases}\displaystyle\sum_{k=0}^{j} a_k b_{j-k} & (0\leqq j\leqq N-1)\\ \displaystyle\sum_{k=j+1-N}^{N-1} a_k b_{j-k} & (N\leqq j\leqq 2N-1)\end{cases} \tag{25}$$

となる．普通に欲しいのはこの上の方の値である．

二つの数列の長さがそれぞれ N, M で等しくない

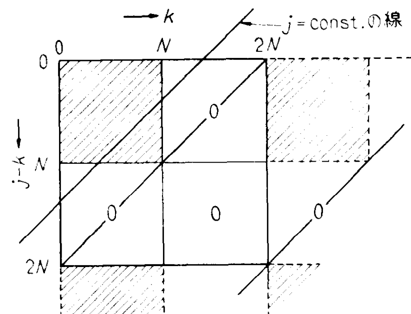

図2 畳込みにおける“裏側での重なり”の防止

ときには，各々に0をつぎ足して，長さが $N+M$ 以上になるようにすればよい．しかし，N と M がはなはだしく異るときには，そして一方が非常に大きいような場合には，一度に全部を計算しないで区間を分けるようにすることによって，計算量はさして変らないが必要な主記憶の容量を大幅に節約できる．

いま $N<M$ とし，(適当に0を補って) $M=lN$ としよう．そこで，a_j の方は前と同様に N 個の 0 を補って $2N$ 個のデータ $\bar{a}_j$ とする．b_j $(0\leqq j\leqq lN-1)$ の方からは長さが $2N$ で半分ずつが重なり合った $(l+1)$ 組のデータをつくる．

$$b_j^{(r)}=b_{j+rN-N} \quad (0\leqq j\leqq 2N-1) \qquad r=0,\ 1,\ 2,\ \cdots,\ l \tag{26}$$

ただし，b_j ははじめの方 $(j<0)$ も終りの方も 0を補うものとする．

こうしてできたデータ $\bar{a}_j$ と $b_j^{(r)}$とから普通に $2N$ 項の循環畳込みを行なった結果を $c_j^{(r)}$ とすると $c_j^{(r)}$ の前半 $(0\leqq j\leqq N-1)$ は "汚染" されていて無意味であるが，後半については

$$c_{j+N}^{(r)}=\sum_k a_k b_{j+N-k}^{(r)}=\sum_k a_k b_{j+rN-k} \tag{27}$$

となり，これらを一連のデータにつなぎ合わせて

$$c_{j+rN}=c_{j+N}^{(r)} \tag{28}$$

とすれば，これが求める結果である．このように，データの片方を区切ることによって，高速記憶に一度に入れるデータの数を少くすることができる．(a_j, b_j の両方を区間に分けることも可能であるが，そうすると計算量が増してしまう．)

相関関数の計算もまた0を補ったデータを使うことによって，"循環的相関関数" になることを避けることができる．相関関数の場合には二組のデータの長さは等しいのが普通であるが，それは多くの場合非常に大きく，それに対して，結果 ϕ_j の j の範囲はあまり大きくない．つまり j が大きくなると $\phi_j\approx 0$ となることが多い．そこで今度は ϕ_j の範囲は $-N\leq j\leq N-1$, a_j, b_j の範囲は $0\leqq j\leqq lN-1$ としたとき，やはり区間に分けることを考えよう．今度は a_i については0を補った（重複しない）部分数列

$$a_j^{(r)}=\begin{cases}a_{j+rN} & 0\leqq j\leqq N-1\\ 0 & N\leqq j\leqq 2N-1\end{cases} \tag{29}$$

をつくる．b_j については前と同じ (26) で定義する．そうして $a_j^{(r)}$ と $b_j^{(r)}$ とから（循環）相関関数をつくると，その後半部は

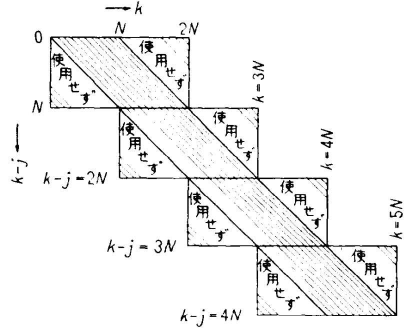

図3　区間分けによる相関関数の計算

$$\phi_j^{(r)}=\Sigma a_k^{(r)}b_{k+N-j}^{(r)}=\sum_{k=0}^{N-1}a_{k+rN}b_{k+rN-j} \quad (0\leqq j\leqq N) \tag{30}$$

となり，したがって

$$\phi_j=\sum_{r=0}^{l-1}\phi_j^{(r)}=\sum_{k=0}^{lN-1}a_k b_{k-j} \tag{31}$$

が $0\leqq j\leqq N$ の場合の求める相関関数になる．（図3参照．）$j\leqq 0$ の方は同様に

$$\phi_j^{(r)\prime}=\Sigma a_i^{(r)}b_{k-j}^{(r+1)}=\Sigma a_{k+rN}b_{k+rN-j} \quad (-N\leqq j\leq 0) \tag{32}$$

から

$$\phi_j=\sum_{r-0}^{l=1}\phi_j^{(r)\prime} \tag{33}$$

として得られる．実際，相関関数は，結果の統計的な変動を少くするために相関長の最大値 N の何十倍，何百倍という長さのデータについてとるのが普通であるから，このような区間分けの手法は本質的である．なお (r) についての総和は実際には最後の逆フーリエ変換を行なう前にすることができ，そうすれば逆フーリエ変換はただ1回ですむのでさらに速くなる．

任意の項数に対する FFT

場合によっては2のベキでない勝手な項数についてフーリエ変換したい場合もある．もちろん2のベキでなくても，項数が 3, 5 など小さい因数に分かれるときは似たような方法が適用できるが，たとえば項数が素数であるような場合も考えられる．また (1) のような完全なフーリエ変換でなくて，W を任意の数として

$$X_j=\sum_{k=0}^{M-1}x_k W^{jk} \quad (j=0,1,\cdots,L-1) \tag{34}$$

を ($W^M=1$ となるような特別の値ではない場合に) 計算したいこともあろう．そのような "不完全フーリ

エ変換"にも FFT の方法が適用できることが指摘された[8]. それはフーリエ変換と次の**フレネル変換**

$$U_j=\Sigma u_k W^{-(j-k)^2/2} \quad (36)$$

との関係を利用するのである. (36)は明らかに一つの畳込み演算である. これをフレネル変換というのは筆者がつけた名前であるが, これは光のフレネル回折像を求める式であり, 特に, よく知られたフレネル積分は (36)(で和を積分にしたもの) の特別な場合であることからも, こう呼ぶことは適当であろう. さて, (36)は

$$U_j=W^{-j^2/2}\Sigma u_k W^{-k^2/2}\cdot W^{jk} \quad (37)$$

であるから, フレネル変換とフーリエ変換は, u_j, U_j に適当な位相因子を乗ずることによって互に移りかわるのである. そうして, (36)は畳込みだから, 適当な項数のフーリエ変換と逆変換によってあらわすことができ, その場合, 数列 u_j に適当に0を補えば, 項数を2のベキになるようにできるから, FFT が有効に使えるのである.

結び

以上は FFT に関する筆者自身の考えと, 主としてアメリカで行なわれた研究の現状の主だったことの紹介である. ここでとり上げなかった重要な話題として FFT を実行するためのハードウェアのことがある. FFT のために特別なハードウェアをつくればさらに高速化できるのは当然であり, 適当な並列処理を行なうなどして, 音波の程度のものは実時間でディジタルにフーリエ解析をすることとさえ可能になる.

参考文献

1) Cooley, J. W. and Tukey, J. W.: An algorithm for the machine calculation of complex Fourier series, Math. Comput. **19** 297-301 (1965).
2) 東大大型計算機センターライブラリー. No. 226, D6/TC/FFTR (ニュース別冊 2-S2, p. 4).
3) 高橋: フーリエ変換よもやま話, 数学セミナー **6** No. 11, 27—31 (1967).
4) PC-1 Library V2, (H. Takahasi, 1960); Goertzel, G.: Fourier analysis, Mathematical Methods for Digital Computers (A. Ralston & H. S. Wilf ed.) Wiley, 1960.
5) Bergland, G. D.: A fast Fourier transform algorithm using base 8 iterations, Math. Comput. **22** 275-279 (1968).
6) Bergland, G.D.: The fast Fourier transform recursive equations for arbitrary length records, Math. Comput. **21** 236-238 (1967).
7) Bergland, G.D.: The fast Fourier transform algorithm for real valued series, Comm. A. C. M. **11** 703-710 (1968).
8) Rabiner, L. R., Shafer, R. W. & Rader, C. M.: The chirp z-transform algorithm and its applications, B. S. T. J. **48** 1249-1292 (1969).

(昭和48年1月5日)