

38

FFT アルゴリズムについて

東大理 高橋秀俊

1. 序

Fourier 変換は、物理学や工学で、振動や波の波形のいわゆる時間(空間)領域表示と周波数(波数)領域表示との間の変換ということで、根本的な役割を演じる。理論的な面では、線形系に関するいろいろの問題が、この二つの領域の間を自由に移動することによって解かれるが、Cooley-Tukey のいわゆる Fast Fourier Transform (FFT) の出現によって、数値的にもこの二つの領域の間の移り変りが極めて手軽にできるようになり、理論的な計算、実験データの処理等に革命的な変革がおこっている。

FFT についての解説はたくさんあるが必ずしも明晰とはいえないものが多い。実は筆者も Cooley, Tukey とは全く独立に FFT に到達し、HITAC 5020 のプログラムをつくったりしたので、筆者の考え方を中心にして FFT の方法

の原理, 問題点などについてのべてみたい。

2. 多次元 Fourier 変換 (直積分解)

Fourier 変換には有限区間のもの, 無限区間のもの, ランダム信号に対するもの等, いろいろあるが, 実際に計算機にかけられる場合は原則として離散 Fourier 変換 (discrete Fourier transform), 即ち

$$(1) \quad X_n = \frac{1}{N} \sum_{m=0}^{N-1} x_m e^{-2\pi i \frac{mn}{N}} \quad n=0, 1, \dots, N-1$$

の形のものである。 $x(t)$ が T を周期とする t の周期関数であるとき, その第 n Fourier 係数は

$$(2) \quad X_n = \frac{1}{T} \int_0^T x(t) e^{-2\pi i \frac{nt}{T}} dt$$

で与えられるが, (1) は $x(t)$ に対して T/N の間隔でサンプル点をとって (2) を数値積分した式に相当する。このような場合, 数値積分には Simpson 則その他どんな式よりも, 単純な台形則による (1) 式が良いのである。以後は Fourier 変換といえは (1) で定義されたものとする。また, (1) を x_m について解いた式

$$(3) \quad x_n = \sum_{m=0}^{N-1} X_m e^{2\pi i \frac{mn}{N}} \quad n=0, 1, \dots, N-1$$

と逆 Fourier 変換と呼ぶことにする。もちろんこれも本質的には Fourier 変換に過ぎない。なお、 x_n, X_n は一般に複素数であるとする。

FFT は要するに (1) の計算を適当に段階に分けて行なうことである。そのようにして計算が速くなる例として、まず多次元 Fourier 変換についてのべよう。一般に n 次元でもいいのであるがここでは 2 次元の場合をのべる。2 次元 Fourier 変換とは

$$(4) \quad X_{n_1, n_2} = \frac{1}{N_1 N_2} \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \left(\frac{n_1 m_1}{N_1} + \frac{n_2 m_2}{N_2} \right)} x_{m_1, m_2}$$

のような変換である。ところでこの変換の行列は $N_1 \times N_1$ 行列と $N_2 \times N_2$ 行列の直積になっていることがわかる。言い換えれば、これを 2 段階に分けて、 N_1 組の N_2 元変換

$$(5) \quad \xi_{m_1, n_2} = \frac{1}{N_2} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \frac{n_2 m_2}{N_2}} x_{m_1, m_2}$$

に対して更に N_2 組の N_1 元変換

$$(6) \quad X_{n_1, n_2} = \frac{1}{N_1} \sum_{m_1=0}^{N_1-1} e^{-2\pi i \frac{n_1 m_1}{N_1}} \xi_{m_1, n_2}$$

を行えばよい。(4) を直接計算すると $(N_1 N_2)^2$ の乗算が必要になるが、このように 2 段に分けると、(5) に $N_1 N_2^2$ 、(6) に $N_2 N_1^2$ で合計 $N_1 N_2 (N_1 + N_2)$ 回の乗算で済む。つまり

(4)の中に出てくる共通の *subexpression* をさきに計算してしまうので得をするのである。2次元または3次元の Fourier 変換は X線結晶解析で普通に使われ、このような直積分解はよく知られている。

ところで、普通の1次元の Fourier 変換でも、 N が 互に素 な二つの因数に $N = N_1 N_2$ のように分解できるときは、直積分解が可能である。いま、与えられた m に対し

$$(7) \quad m \equiv N_2 m_1 + N_1 m_2 \pmod{N}$$

となるような m_1, m_2 が (それぞれ $\text{mod } N_1, \text{mod } N_2$ で) *unique* に定まるから、それによって x_m を x_{m_1, m_2} と書くと、

(1) は

$$(8) \quad X_n = \frac{1}{N} \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \left(\frac{m_1 n}{N_1} + \frac{m_2 n}{N_2} \right)} x_{m_1, m_2}$$

となる。ここで更に

$$(9) \quad \begin{aligned} n &= n_1 \pmod{N_1} & (0 \leq n_1 < N_1) \\ n &= n_2 \pmod{N_2} & (0 \leq n_2 < N_2) \end{aligned}$$

とすると (8) は

$$(10) \quad X_{n_1, n_2} = \frac{1}{N_1 N_2} \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \left(\frac{m_1 n_1}{N_1} + \frac{m_2 n_2}{N_2} \right)} x_{m_1, m_2}$$

となり、2次元 Fourier 変換 (4) と全く同じになる。したがって直積分解される。

3. Fast Fourier Transform の原理

さて, $N = N_1 N_2$ の N_1 と N_2 が互に素でない場合は(1)の直積分解はできないが, 計算量を減らすという目的は, 2段階に分けることによってやはり達することができ.

N_1 と N_2 が互に素でなくても

$$(11) \quad \begin{aligned} m &= m_1 + N_1 m_2 & (0 \leq m_1 < N_1, 0 \leq m_2 < N_2) \\ n &= N_2 n_1 + n_2 & (0 \leq n_1 < N_1, 0 \leq n_2 < N_2) \end{aligned}$$

のように置くことは可能である. そこで前と同様にいて

$$(12) \quad \begin{aligned} X_n &= \frac{1}{N} \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \frac{(N_2 n_1 + n_2)(m_1 + N_1 m_2)}{N_1 N_2}} x_{m_1 + N_1 m_2} \\ &= \frac{1}{N_1} \sum_{m_1} e^{-2\pi i \frac{n_1 m_1}{N_1}} e^{-2\pi i \frac{m_1 n_2}{N_1 N_2}} \left(\frac{1}{N_2} \sum_{m_2} e^{-2\pi i \frac{n_2 m_2}{N_2}} x_{m_1 + N_1 m_2} \right) \end{aligned}$$

ここで括弧の中

$$(13) \quad \xi_{m_1, n_2} = \frac{1}{N_2} \sum_{m_2=0}^{N_2-1} e^{-2\pi i \frac{n_2 m_2}{N_2}} x_{m_1 + N_1 m_2}$$

は前と同様 N_2 元の Fourier 変換である. また

$$(14) \quad \tilde{\xi}_{m_1, n_2} = e^{-2\pi i \frac{m_1 n_2}{N_1 N_2}} \xi_{m_1, n_2}$$

と書けば

$$(15) \quad X_n = \frac{1}{N_1} \sum_{m_1=0}^{N_1-1} e^{-2\pi i \frac{n_1 m_1}{N_1}} \tilde{\xi}_{m_1, n_2}$$

であって, これも単なる N_1 元 Fourier 変換にほかならない.

ただ二つの Fourier 変換の間に乗算 (14) が一つ入ったために直積とはいえないただけである。(14)は異なる m_1, n_2 の組について $N_1 N_2 = N$ 個の式があるが、その中で m_1 または n_2 が 0 である場合は乗算の必要はないから、 $(N_1 - 1)(N_2 - 1)$ 回の乗算を必要とする。これを考えに入れても計算量は大幅に減らされる。

こうして、 $N_1 N_2$ 元の Fourier 変換は、

(a) N_2 元の Fourier 変換を N_1 回行なう。

(b) (a) の結果 ξ_{m_1, n_2} に $e^{-2\pi i m_1 n_2 / N}$ を乗ずる (位相回転)。

(c) (b) の結果に N_1 元の Fourier 変換を行なう。

の 3 段階に分解された。ここで N_1 または N_2 (または両方) が更に因数に分けられれば、更に同じ方法で簡略化できることはいうまでもない。こうしてできる限り計算量を減らそうというのが FFT の考え方である。

ここで最も有利なのは N が 2 のべき $N = 2^k$ である場合である。そのとき

$$(1b) \quad N_1 = 2, \quad N_2 = N/2 = 2^{k-1}$$

とすると、上の (c) の段階の変換行列は 2×2 行列 $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ であり、乗算は一つもない。 N_2 の方から更に因数 2 を出して同じようにし、同様のことを繰り返して行くと、Fourier 変換の部分はすべて 2 元の変換となってしまう。そこで乗算

の必要なのは (b) の段階の位相回転だけで、これが 1 段階につき $N/2$ 回の乗算となるので、全部でおよそ $kN/2 = N \log_2 N / 2$ の乗算ですむことになる。加算の回数と同じ程度である。こうして、普通にやれば N^2 の程度の乗算が必要になる Fourier 変換が、飛躍的に高速化される。

4. FFT のいろいろの変形

以上の説明からも想像がつくように、FFT の algorithm にはいろいろの形が考えられる。まず、 N は 2 のべきでなくても、3 とか 5 とかの小さい素因数だけからできた数であれば、同じくらいに速い algorithm をつくることができる。また、 N が 2 のべきである場合 N_1 として 2 をとらずに $2^2 = 4$ をとることが考えられる。この場合の N_1 元の変換

は $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -i \\ 1 & i & -1 & -i \end{pmatrix}$ となり、やはり乗算の計算はいらない。

位相回転に必要な乗算の数は $(4-1) \times N/4 = \frac{3}{4}N$ であるが、全体の繰り返しの数が半分になるので、全体では乗算回数はおよそ $\frac{3}{8}N \log_2 N$ となり、 $N_1 = 2$ とした場合よりも更に有利となる。

同じようにして $N_1 = 8$ の場合、 $N_1 = 16$ の場合も検討されている。これらの場合は変換行列の部分に 1 乗算があらわ

れてくるが、それを考慮に入れても、上手にプログラムを組めば $N_1 = 4$ のときより更によくなることが知られている。しかし同時にプログラムは非常に長くなる。

N の因数分解を逆にして、 $N_1 = N/2$, $N_2 = 2$ とすることもできる。こうして、毎回 N_2 の方を小さくして分解を進めて行って得られる algorithm は一見したところ前述のものとはほとんど違わないが、ただ毎回 (k) の段階で乗じる位相因子の形が違ってくる。こうして得られた algorithm は Sandé-Tukey algorithm と呼ばれている。元の Cooley-Tukey algorithm と特に優劣の差は見当らないようである。

逆 Fourier 変換. FFT の algorithm が $(X) \rightarrow (x)$ の逆変換にも適用できることは (3) 式の形からも当然であるが、また 2 元変換や位相回転から成る FFT の process を逆にたどって、それぞれの step の逆操作を逆の順序で行なえば逆変換が得られることから明らかである。こうして逆行した場合、Cooley-Tukey algorithm の逆 process は Sandé-Tukey algorithm であることがわかる。

5. 実数データに対する FFT

今までの話ではデータ x_n と結果 X_n と、その他の中間結果とすべて複素数と考えて来た。つまり 1 回の乗算とは複素

数と複素数の乗算のことであって、これを実数の乗算になおせば4回の乗算に相当するものであった。だから標準的な1回のFFTに含まれる実数の乗算の数は実は $2N \log_2 N$ である。

ところで実際にはデータ(x)は実数である場合が多い。その場合に複素数データに対するalgorithmをそのまま適用しては損であろうことは直観的にも明らかである。そこで実数データに対するFFTのalgorithmが開発された。

といってもそれは本質的には複素数データに対するものと異なるところはない。ただその場合重複して出てくるものを省いたに過ぎない。即ち x_n がすべて実数である場合は、FFTのprocessの途中であらわれるどの数値に対しても、それが本来的に実数である場合を除けば、必ずそれに共役な数値がどこかで計算されている。そこで互に共役な数値の一方に対する計算は全く余分だから省いてしまうようなプログラムを使えば、計算の量をほぼ半分ですますことができる。こうして、実数データに対するCooley-Tukey algorithmに必要な実数の乗算の数はおよそ $N \log_2 N$ であることがわかる。この場合、結果 X_n はもちろん複素数であるが、

$X_{N-n} = X_n^*$ という共役関係があるので、 X_n は半分の区間、つまり $0 \leq n \leq N/2$ について求めれば十分である。

別のやり方として、複素数データに対する algorithm をそのまま使って実数データの Fourier 変換を無駄なく行うような工夫もある。その一つは、二つの（全く無関係な）データに対する Fourier 変換を一度に計算するという方法である。二組のデータ $(x), (y)$ があるとき、これから複素数のデータ $z_n = x_n + iy_n$ をつくり、これに対する Fourier 変換 Z_n を FFT で求める。すると求めたい Fourier 変換の結果である X_n と Y_n は

$$(17) \quad X_n + iY_n = Z_n, \quad X_n - iY_n = Z_{N-n}^*$$

によって求められる。

もう一つの方法は、 N 個から成る実数データ (x) から、 $N/2$ 個の複素数データ

$$(18) \quad z_n = x_{2n} + ix_{2n+1} \quad 0 \leq n \leq \frac{N}{2} - 1$$

をつくらせて、 z_n に $\frac{N}{2}$ 元の FFT を行なうのである。すると、その結果の Z_n から、

$$(19) \quad X_n = \frac{1}{4} (1 - ie^{-2\pi i \frac{n}{N}}) Z_n + \frac{1}{4} (1 + ie^{-2\pi i \frac{n}{N}}) Z_{\frac{N}{2}-n}^*$$

によって X_n がきまる。

6. 記憶場所の問題

以上のような algorithm を実際のプログラムにしようと思えば、計算の途中の数値の入れ場所が問題になる。大きい N

について変換する場合、途中の数値のために別の場所を確保しておくことはできれば避けたいわけであるが、幸いこの algorithm は低次元の変換の繰り返しの形をとっているから、各ステップでいつも変換結果を変換前のデータのあった場所に overwrite するようにすれば、ほとんど余分の記憶場所を使わずに、最初にデータのあった場所に最後に結果が入るようにできる。ところがそうやるとちょっと具合の悪いことがある。変換の結果の X_n が、 n の順序とは全く違う妙な順序にならぶのである。全部を2元変換に分解する普通のやり方の場合だと、 n を2進法であらわして、それを逆読み（最上位を最下位と思う読み方）した値が順序よく並ぶのである。たとえば、 $N=8$ ならば

$$X_0, X_4, X_2, X_6, X_1, X_5, X_3, X_7$$

という順に並ぶ。そこで、あとでこれを添字の大きさの順にならべなおすか、そうであるければ値が必要になったときにそのありかを見つけるためのルーチンを用意する必要があるが、いずれにしてもちょっと厄介なことである。しかし、ある種の応用、たとえば「たたみこみ」の場合などは、周波数領域の量 X_n を直接に使うことはなく、同一の n に対する値同志で演算をした後に、逆変換でまた x_n の領域に戻る。そのような場合には、逆変換プログラムをこのような特殊なやり方

のデータに合うようにつくっておけば、わざわざならべかえをすることは必要がなくなる。

[FFTの応用]

7. たたみこみと相関の計算

Fourier 変換がこのようにして非常に速く計算できるとなると、さまざまな応用がひらけてくる。本来の周波数分析に用いられるのはいうまでもないことで、いままでアナログ的に変換していたのをデジタルに切りかえる傾向が顕著である。しかし最も興味ある応用は、二つの数列の間のたたみこみ演算 (convolution), つまり a_n, b_n から

$$(20) \quad c_n = \sum_r a_r b_{n-r}$$

を求める演算であろう。(20) を数列 (b) と数列 (c) との間関係と見ると、それは一つの線形系 (線形 filter) の入力信号 $b(t)$ と出力信号 $c(t)$ との関係であり、 (a) はその線形系の余効関数 (重み関数, 遅延スペクトル) と呼ばれるものに相当する。したがって (20) は一つの filter を simulate するもの (digital filter) であり、実際に信号の処理に使うことができる。

さて、 a_n, b_n の Fourier 変換を A_n, B_n とすると、 c_n の Fourier 変換 C_n は

$$(21) \quad C_n = N A_n B_n$$

のように積になることはよく知られている。そこで C_n は逆変換

$$(22) \quad c_n = \sum_{m=0}^{N-1} C_m e^{2\pi i \frac{mn}{N}}$$

によって求められるから、一つのたたみこみ演算は Fourier 変換を 3 回使うことによって実行できる。(20) はこれ以上簡単になりそうに見えない簡単な式であるが、実際は Fourier 変換が非常に速くできるために、こんなにまわり道をして、その方が速いのである。

ここで注意しなければならないのは、いま考えている有限 Fourier 変換では、数列 (a) , (b) 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ということである。

したがって、以上の方法でつくったたたみこみは、いわゆる循環たたみこみ (cyclic convolution) で、正確に書けば

$$(23) \quad c_n = \sum_{m=0}^n a_m b_{n-m} + \sum_{m=n+1}^{N-1} a_m b_{N+n-m}$$

となるものである。ところが普通に求めたいのは (23) の右辺の第一項の方であって、第二項は余分なのである。

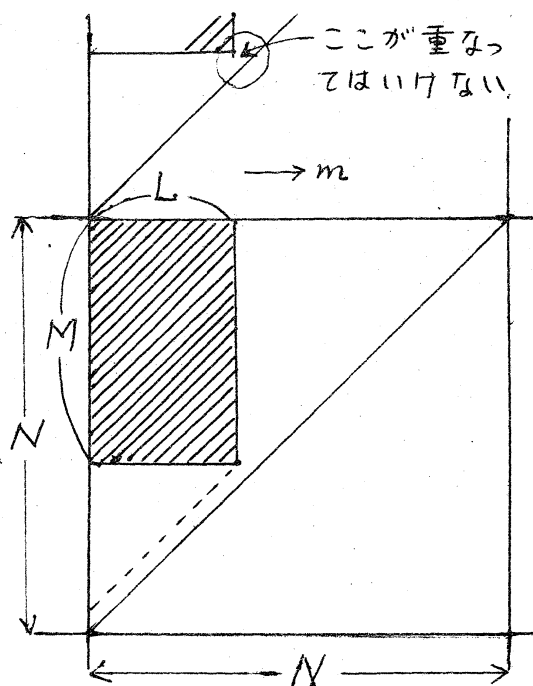
この問題はしかし簡単に解決される。データの数列のあとに適当に 0 のデータを補って、Fourier 変換における周期を少し長くしてやればよい。具体的には、データ a_n の数が L ,

データ t_n の数が M であれば、周期 N は

$$N \geq L + M - 1$$

となるようにとる。つまり $L + M - 1$ より大きい適当な N のべきをとればよい。そうして L より大きい n に対する a_n , M より大きい n に対する t_n はすべて 0 にしてしまうのである。そうした上で循環的なたたみこみを行えば、数列の終りにとった十分に大きい空白区間のおかげで、“裏側”での重なりが完全に避けられる。即ち(23)の右辺第二項は 0 になる。

ここで長さ L と M とが甚だしく異なるとき、たとえば $L \ll M$ であるような場合は、たたみこみの計算を一度にやらないで、いくつかの区間に分けて行なうことができる。そうすることによって、計算に必要な記憶容量が少なくてすむ。長い時系列を filter するような場合、filter の余効関数の長さの方はあまり長くないというような場合がこれに該当する。



これには N を適当に大きく（少くとも $2L$ より大きく）選

び, $M' = N - L + 1$ として t_n の数列を M' の長さに区切って, それぞれの区間の t_n について a_n とのたたみこみを行ない, それらの結果を, 端の重なり合う部分では加え合わせながら一連の数列につなぎ合わせればよい.

この場合, 重なり合う部分で加算をすることは, 計算量としてはとくに足りないものであるが, そのために前段の結果の一部を主記憶に残しておくか, または補助記憶に書いたものをもう一度読み出すことになる. そこで, はじめのデータ t_n の方を適当に重なるようにとることによって, この後からの加算をなしですます工夫もなされている.

たたみこみと同じようなものに

$$(24) \quad \varphi_n = \sum_m a_m t_{m-n}$$

で定義される相互相関関数がある. これを (20) とくらべると, t の添字の符号が逆になったただけの違いなので, φ_n の Fourier 変換 Φ_n は

$$(25) \quad \Phi_n = N A_n B_n^*$$

で与えられることになる (a_n, t_n は実数とした場合). そこでたたみこみの場合と同様に, 3 回の Fourier 変換で φ_n を求めることができる. もちろんこの場合も“循環的相関関数”になることを避けるためには, データに適当に 0 を附加してやる必要がある.

Φ_n は相互パワースペクトルと呼ばれ、従来はこれを求めるためにはまず相互相関関数 φ_n を計算し、これに Fourier 変換を行なって Φ_n を得るという方法が多く使われた。それが計算手段の進歩の結果、 φ_n を求めるのに逆に Φ_n を (25) から直接計算し、それに逆 Fourier 変換を使うという全く主客転倒になったのは面白いことである。

相関関数を求める場合のデータ a_n, b_n は非常に長い時系列である場合が多い。しかもその両方の長さは等しく、また相関関数 φ_n の方は 0 を中心とした比較的短い区間でだけ求めればよいというのが普通である。そこで、この場合もまた区別して計算する方法が役立つ。

いま N を Fourier 変換の項数(周期)としたとき、 a_n を $\frac{N}{2}$ の長さで区切る。したがって第 $r+1$ 切片は $\frac{rN}{2}$ から始まるが、ここで第 $r+1$ 番目の a_n の配列は隣同志の二つの切片をつなげたものとする。したがって

$$(26) \quad a_n^{(r)} = a_{n + \frac{rN}{2}} \quad (0 \leq n \leq N-1)$$

で a_n の配列は半分ずつ重り合っている。 b_n の方も $\frac{N}{2}$ で区切るが、この方は二つつなげないで、データは前半だけに入れ、後半には 0 をつめる、即ち

$$(27) \quad b_n^{(r)} = \begin{cases} b_{n + \frac{rN}{2}} & (0 \leq n \leq \frac{N}{2} - 1) \\ 0 & (\frac{N}{2} \leq n \leq N-1) \end{cases}$$

そこで第 $r+1$ 区間の
部分和 $\varphi_n^{(r)}$ は

(28)

$$\varphi_n^{(r)} = \sum_{m=n}^{n+\frac{N}{2}-1} a_{m+\frac{rN}{2}} b_{m+\frac{rN}{2}-n}$$

となる。ここで、後半部、つまり
 $\frac{N}{2} < n \leq N-1$ の場合は裏側の重

なりのために出てきた無意味な項で“汚染”
されているので使えないが、前半部、

$0 \leq n \leq \frac{N}{2}$ の部分は正しい値になっている。

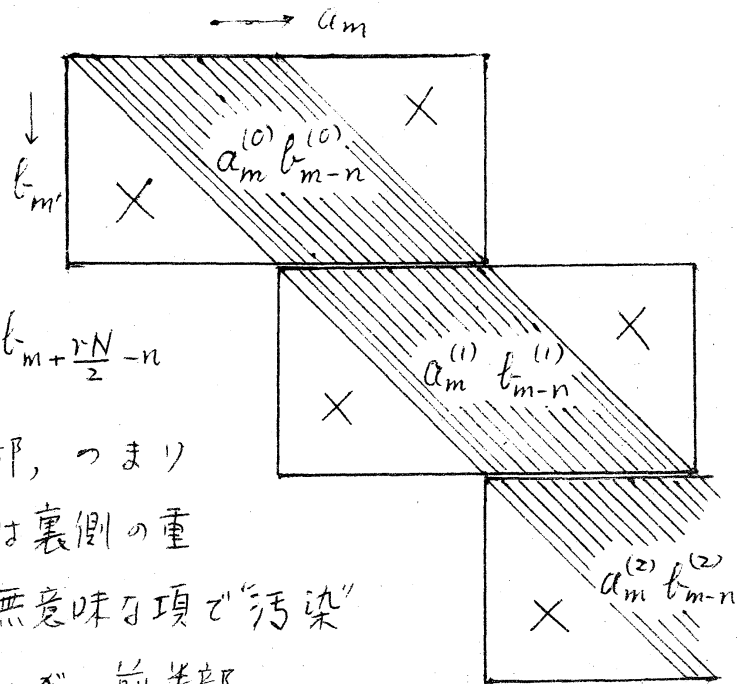
こうして得た $\varphi_n^{(r)}$ を全部加えて

$$(29) \quad \varphi_n = \sum_r \varphi_n^{(r)} \quad \left(0 \leq n \leq \frac{N}{2} \right)$$

とすれば、これが求める相関関数となる。

これで $n \geq 0$ の相関関数が求まったが、 $n \leq 0$ の方の
値は a_n と b_n の役割を交換して同じようにやれば計算でき
る。

このようにして、相関の長さの最大値 $\frac{N}{2}$ があまり大きく
なればデータの長さはどんなに長くても計算ができる。な
お、実際の計算では、 r についての総和は最後の逆 Fourier
変換を行なう前に行なった方が FFT の回数を節約できる。



8. 任意の項数の Fourier 変換

有限 Fourier 変換 (1) が連続関数の積分の近似としてではなく、本来的に有限な Fourier 変換として出てくるような問題では N の値ははじめから定っていて、たとえば 2 のべきというような特別な値に指定できないであろう。 N は素数であるような場合さえ考えられる。また (1) を一般化して、

$$(30) \quad X_n = \sum_{m=0}^{N-1} x_m W^{nm} \quad n=0, 1, \dots, M-1$$

を 1 のべき根でも何でもよい値 W に対して計算したいような場合もある。ところが、このような“不完全 Fourier 変換”でも FFT なみに速く計算する方法がある。それは Fourier 変換と、次の Fresnel 変換

$$(31) \quad U_n = \sum u_m W^{-\frac{(m-n)^2}{2}}$$

との関係に着目するのである。(31) は明らかに一つのたたみこみ演算である。(これを Fresnel 変換というのは筆者がつけた名前であるが、これは光の Fresnel 回折像を求める式であり、特に、よく知られた Fresnel 積分は (31) (で u を積分に変えたもの) の特別な場合であることから、こう呼ぶのが適当だと思われる。) さて、(31) は

$$(32) \quad U_n = W^{-\frac{n^2}{2}} \sum (u_m W^{-\frac{m^2}{2}}) \cdot W^{mn}.$$

と書けるから, Fresnel 変換と Fourier 変換とはデータと結果とにそれぞれ適当な位相因子を乗ずることによって互に移り変るものである。(これは光の Fresnel 回折像が, 適当にレンズで焦点をずらすことによって Fraunhofer 回折像に移ることに相当している。) そうして Fresnel 変換はたたみ込みだから, FFT を利用して計算ができる。しかも, たたみこみについては適当に 0 をつけ加えさえすれば特別な周期性というふうなものは問題にならないから, (31) で W の値が何でもあってもかまわないのである。 W は絶対値が 1 であることさえ原理的には要求されない, しかし $|W| \neq 1$ であると実際には丸め誤差のために計算結果が全く意味をもたなくなることもあるので注意を要する。

9. むすび

以上, FFT の algorithm とその応用についての原理的なことの概要をのべた。実際にこれをプログラムにするためには, 更にいろいろな考慮, たとえば $e^{-2\pi i \frac{mn}{N}}$ をどうしてつくるか (表をひくか, 計算するか), 記憶場所をどうとるか, ならべかえの algorithm, ± 1 や $\pm i$ による乗算をとばす方法, 実数に対する routine で逆変換を行なう方法などいろいろ必要であるが, それらについてはそれぞれの文献にゆずろう。