第7章　高速 Fourier 変換: FFT

7.1　はじめに

FFT(Fast Fourier Transform = 高速フーリエ変換) は, DFT(Discrete Fourier Transform = 離散フーリエ変換) を高速に実行する手法です. ここでは, フーリエ変換の意味や性質については既知として, DFT を簡単に導入したうえで, FFT そのものの構成や歴史に絞って解説していきます.

7.2　離散フーリエ変換

周期 2π もつ関数 $f(x)$ を以下のようにフーリエ級数展開することを考えます:

$$f(x) = \sum_{k=-\infty}^{\infty} a_k e^{ikx}. \tag{7.1}$$

$f(x)$ の値が一周期にわたって連続的に分かっていれば, 展開係数 a_k の値は以下のフーリエ積分で求めることができます:

$$a_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}dx. \tag{7.2}$$

しかし, 関数 $f(x)$ が観測データ等である場合, 関数値は連続的には得られず, 離散的な x の値に対してしか関数値が与えられないことが普通です. 例えば, N 個の等間隔の点 x_j:

$$x_j = \frac{2\pi}{N}j, \quad (j = 0, 1, \cdots, N-1), \tag{7.3}$$

[1]数学セミナー 1998 年 12 月号 [4] より著者, 石岡圭一 東京大学 数理科学研究科 助教授の許可を得て抜粋.

での関数値 f_j:

$$f_j = f(x_j), \quad (j = 0, 1, \cdots, N-1), \tag{7.4}$$

のみが得られている場合を考えましょう. このとき, フーリエ展開 (7.1) における展開関数 e^{ikx} は N 個の点 x_j でしか評価されませんので, N 次元のベクトルとみなされます. N 次元のベクトルですので, 当然一次独立な成分は N 個しかありえません. 特に今の場合のように点 x_j を等間隔にとっている場合には,

$$e^{i(k+N)x_j} = e^{i(k+N)\frac{2\pi}{N}j} = e^{2\pi ij} e^{ik\frac{2\pi}{N}j} = e^{ik\frac{2\pi}{N}j} = e^{ikx_j} \tag{7.5}$$

となり, 波数 k を N だけずらしたベクトルが波数 k のベクトルと全く同じものになります. これは, 有限の点で関数を評価するために, ある波数の成分が別の波数の成分と区別がつかなくなって同一視されてしまうことによります. この現象は, ある波数の成分に「別名」を与えてしまうということで,「エリアジング」と呼ばれています. 従って, 今の場合, フーリエ級数展開 (7.1) は無限個の和を考える代わりに, 独立な N 個の成分の和:

$$f_j = \sum_{k=0}^{N-1} a_k e^{ikx_j}, \quad (j = 0, 1, \cdots, N-1), \tag{7.6}$$

のみを考えるべきであることが分かります.

では, 展開 (7.6) において, 与えられた関数値 f_j から展開係数 a_k を求めるにはどうすればよいでしょうか? すぐ思いつく方法は,「(7.6) の左辺が既知なのだから, (7.6) を N 元連立一次方程式として解けば a_k が求まる」というものですが, 実はもっとうまい方法があります. 等比級数の和の公式により, 整数 l が N の倍数でない場合には,

$$\sum_{j=0}^{N-1} e^{ilx_j} = \sum_{j=0}^{N-1} e^{il\frac{2\pi}{N}j} = \sum_{j=0}^{N-1} \left(e^{il\frac{2\pi}{N}}\right)^j = \frac{\left(e^{il\frac{2\pi}{N}}\right)^N - 1}{e^{il\frac{2\pi}{N}} - 1} = 0, \tag{7.7}$$

となることが分かりますし, 整数 l が N の倍数の場合には

$$\sum_{j=0}^{N-1} e^{ilx_j} = \sum_{j=0}^{N-1} e^{il\frac{2\pi}{N}j} = \sum_{j=0}^{N-1} 1^j = N, \tag{7.8}$$

となることが分かります. 従って, (7.6) の両辺に $e^{-ikx_j} \quad (k = 0, 1, \cdots, N-1)$ を掛けて j について総和をとると,

$$(\text{左辺}) = \sum_{j=0}^{N-1} f_j e^{-ikx_j}, \tag{7.9}$$

$$(\text{右辺}) = \sum_{j=0}^{N-1} \left(\sum_{k'=0}^{N-1} a_{k'} e^{ik'x_j} \right) e^{-ikx_j} = \sum_{k'=0}^{N-1} a_{k'} \left(\sum_{j=0}^{N-1} e^{i(k'-k)x_j} \right) = Na_k, \qquad (7.10)$$

が得られます. ここで, (7.6) における k を k' に書換え, $k'-k$ が N の倍数になるのは, $k'=k$ となるときだけであることを用いました. 以上から, a_k を求めるための公式:

$$a_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-ikx_j}, \quad (k = 0, 1, \cdots, N-1), \qquad (7.11)$$

が得られたことになります. $f(x)$ が連続量として与えられていた場合のフーリエ積分の式 (7.2) と見くらべると, この式は, 積分を区分求積法的な和に置き換えたものになっていることが分かります.

以上により, 関数の評価点が離散的に与えられた場合の展開係数 a_k と関数値 f_j との間の関係式 (7.6),(7.11) が導かれましたが, これらの式は離散フーリエ変換 (Discrete Fourier Transform = DFT) と呼ばれています.

7.3 高速フーリエ変換

DFT の 2 つの式 (7.11) と (7.6) とは, 互いに変換と逆変換との関係にありますが, 殆ど同じ形をしていることに気がつきます. 実際, (7.11) の両辺を N 倍して複素共役をとると,

$$(Na_k)^* = \sum_{j=0}^{N-1} f_j^* e^{ikx_j} = \sum_{j=0}^{N-1} f_j^* e^{ik\frac{2\pi}{N}j} = \sum_{j=0}^{N-1} f_j^* e^{ij\frac{2\pi}{N}k} = \sum_{j=0}^{N-1} f_j^* e^{ijx_k}, \qquad (7.12)$$

となり, (7.6) と全く同じ形に帰着されます. 従って, DFT の計算について考える場合, (7.6) 式の形のみを考えるだけで十分であることが分かります. というわけで, 以下 (7.6) 式の計算について考えていくわけですが, 便宜上, $x_j = 2\pi j/N$ であることを使って, 以下のような形に書換えておきます:

$$F(j) = \sum_{k=0}^{N-1} W_N^{jk} A(k), \quad (j = 0, 1, \cdots, N-1). \qquad (7.13)$$

ここに, $F(j) = f_j$, $A(k) = a_k$ と書き直し, $W_N = e^{\frac{2\pi i}{N}}$ と定めています.

さて, (7.13) 式の計算について考えていきましょう. この計算は, $N \times N$ 行列 (W_N^{jk}) に N 次元ベクトル a_k を掛けることに他なりませんから, 必要な計算量は, 積が N^2 回と和が $N(N-1)$ 回で, ともに $\mathcal{O}(N^2)$ 回ということになります. 従って, 例

えば $N = 10000$ の場合を考えると, DFT の計算のために $N^2 = 1$ 億回オーダーの演算が必要になります. 今では, パソコンの性能が向上して, 最速のものになると 1GFlops(1 秒間に 10 億回の浮動小数点演算を実行できる) に迫っていますから, この程度の計算ならパソコンでやれないこともありません. しかし, 電子計算機の草創期ではそんな余裕なことを言っていられませんでした. 1960 年代になってやっとスーパーコンピューターと呼ばれる高速計算機が現れてきましたが, それでもたかだか 1MFlops(1 秒間に 100 万回の浮動小数点演算を実行できる) 程度の能力しかありませんでした. というわけで, 当時は最速の計算機を使っても, 上記の DFT の計算が 100 秒以上かかってしまうという状況であり, N が大きい場合の DFT は気軽には実行できませんでした.

そこで, 「DFT を計算するための何かうまい手法はないか?」というのが時代の要請であり, 「必要は発明の母」の諺どおりに高速フーリエ変換 (Fast Fourier Transform = FFT) が生まれることになりました. ここでは, やや冗長にはなりますが, FFT の「発見者」である Cooley and Tukey (1965) [2] の論文に沿ってその概要を見ていきましょう.

まず, 変換の長さ N が $N = N_1 P$ と 2 つの整数 N_1, P の積になっている場合を考えます. このとき, (7.13) の添字 j, k それぞれを 2 つの添字の組 (j_p, j_1), (k_1, k_p) によって,

$$j = N_1 j_p + j_1, \quad (j_1 = 0, 1, \cdots, N_1 - 1;\ j_p = 0, 1, \cdots, P - 1), \tag{7.14}$$

$$k = P k_1 + k_p, \quad (k_1 = 0, 1, \cdots, N_1 - 1;\ k_p = 0, 1, \cdots, P - 1), \tag{7.15}$$

のように表示することにします. これは, C などのプログラミング言語において, $N_1 \times P$ および $P \times N_1$ の 2 次元配列を使っていると考えると分かりやすいかもしれません. この表示を使うと, (7.13) は以下のように書換えられます:

$$F(j_p, j_1) = \sum_{k_1=0}^{N_1-1} \sum_{k_p=0}^{P-1} W_N^{(N_1 j_p + j_1)(P k_1 + k_p)} A(k_1, k_p). \tag{7.16}$$

ここで,

$$\begin{aligned} W_N^{(N_1 j_p + j_1)(P k_1 + k_p)} &= W_N^{N_1 P j_p k_1 + P j_1 k_1 + (N_1 j_p + j_1) k_p} \\ &= W_N^{N j_p k_1} W_N^{P j_1 k_1} W_N^{(N_1 j_p + j_1) k_p} \\ &= (W_N^N)^{j_p k_1} W_N^{P j_1 k_1} W_N^{(N_1 j_p + j_1) k_p} \\ &= W_N^{P j_1 k_1} W_N^{(N_1 j_p + j_1) k_p} \end{aligned}$$

であることに注意すると, (7.16) は,

$$F(j_p, j_1) = \sum_{k_p=0}^{P-1} W_N^{(N_1 j_p + j_1)k_p} \sum_{k_1=0}^{N_1-1} W_N^{Pj_1k_1} A(k_1, k_p), \tag{7.17}$$

すなわち,

$$F(j_p, j_1) = \sum_{k_p=0}^{P-1} W_N^{(N_1 j_p + j_1)k_p} A_1(j_1, k_p), \quad (j_1 = 0, 1, \cdots, N_1 - 1;\ j_p = 0, 1, \cdots, P - 1), \tag{7.18}$$

$$A_1(j_1, k_p) = \sum_{k_1=0}^{N_1-1} W_N^{Pj_1k_1} A(k_1, k_p), \quad (j_1 = 0, 1, \cdots, N_1 - 1;\ k_p = 0, 1, \cdots, P - 1), \tag{7.19}$$

と 2 段階に分解されます. さて, このように分解すると, 演算回数がどう変化するか数えてみましょう. まず, (7.19) には積が $N_1^2 \times P$ 回, 和が $N_1(N_1 - 1) \times P$ 回必要で, (7.18) については, 積が $P^2 \times N_1$ 回, 和が $P(P - 1) \times N_1$ 回となりますから, 合計で, 積が $N_1P(N_1 + P) = N(N_1 + P)$ 回, 和が $N_1P(N_1 + P - 2) = N(N_1 + P - 2)$ 回必要ということになります. 従って, $\mathcal{O}(N^2)$ の計算量が $\mathcal{O}(N(N_1 + P))$ になったことになります. 最初に $N = 10000$ の場合の計算量を例にとりましたが, この場合, $N_1 = P = 100$ ととってあげると, $N^2 = 1$ 億回の計算量が $N(N_1 + P) = 200$ 万回と, $1/50$ に減じられることになります.

これでも十分にすばらしい改良なのですが, さらに, $P = N_2Q$ と P が 2 つの整数の積に分解できる場合には, j_p, k_p それぞれを 2 つの添字の組 (j_q, j_2), (k_2, k_q) によって,

$$j_p = N_2 j_q + j_2, \quad (j_2 = 0, 1, \cdots, N_2 - 1;\ j_q = 0, 1, \cdots, Q - 1), \tag{7.20}$$

$$k_p = Qk_2 + k_q, \quad (k_2 = 0, 1, \cdots, N_2 - 1;\ k_q = 0, 1, \cdots, Q - 1), \tag{7.21}$$

と表すことにより, (7.18) の計算は以下のように変形できます (以下, 添字の範囲に

ついては省略しています):

$$
\begin{aligned}
F(j_q, j_2, j_1) &= \sum_{k_q=0}^{Q-1} \sum_{k_2=0}^{N_2-1} W_N^{(N_1(N_2 j_q + j_2)+j_1)(Qk_2+k_q)} A_1(j_1, k_2, k_q) && (7.22)\\
&= \sum_{k_q=0}^{Q-1} W_N^{(N_1(N_2 j_q + j_2)+j_1)k_q} \sum_{k_2=0}^{N_2-1} W_N^{Q(N_1(N_2 j_q + j_2)+j_1)k_2} A_1(j_1, k_2, k_q) && (7.23)\\
&= \sum_{k_q=0}^{Q-1} W_N^{(N_1(N_2 j_q + j_2)+j_1)k_q} A_2(j_1, j_2, k_q). && (7.24)
\end{aligned}
$$

ここに, $W_N^{QN_1N_2} = W_N^N = 1$ であることに注意して,

$$
A_2(j_1, j_2, k_q) = \sum_{k_2=0}^{N_2-1} W_N^{Q(N_1 j_2 + j_1)k_2} A_1(j_1, k_2, k_q) \tag{7.25}
$$

と定めました. このように分解すると, 計算量は $\mathcal{O}(N(N_1 + N_2 + Q))$ に下げられることになります. さらに $Q = N_3 R$ と表されるときに (7.24)を分解すると $\cdots$ と, この分解を繰り返していくことを考えると, 結局, $N = N_1 N_2 \cdots N_m$ のとき, $F(j)$ は以下のような漸化式的手順で計算できることになります.

i) まず, 以下のように初期値 A_1 を A から定めます:

$$
\begin{gathered}
Q_1 = N_1; \quad P_1 = N/Q_1,\\
A_1(j_1, k_p) = \sum_{k_1=0}^{N_1-1} W_N^{P_1 j_1 k_1} A(k_1, k_p),\\
(j_1 = 0, 1, \cdots, N_1 - 1;\ k_p = 0, 1, \cdots, P_1 - 1).
\end{gathered} \tag{7.26}
$$

ii) あとは $l = 2, 3, \cdots, m$ について, 漸化式的に順に A_l を求めていきます:

$$
\begin{gathered}
Q_l = N_1 N_2 \cdots N_l; \quad P_l = N/Q_l,\\
A_l(j_1, j_2, \cdots, j_{l-1}, j_l, k_p) = \sum_{k_l=0}^{N_l-1} W_N^{P_l(Q_{l-1} j_l + Q_{l-2} j_{l-1} + \cdots + Q_1 j_2 + j_1)k_l} A_{l-1}(j_1, j_2, \cdots, j_{l-1}, k_l, k_p)\\
(j_r = 0, 1, \cdots, N_r - 1\ (r = 1, 2, \cdots, l);\ k_p = 0, 1, \cdots, P_l - 1).
\end{gathered} \tag{7.27}
$$

iii) 以上を繰り返していくと最終的に A_m が求まるので, それから F を添字の並べ替えによって以下のように求めます:

$$
\begin{gathered}
F(j_m, j_{m-1}, \cdots, j_2, j_1) = A_m(j_1, j_2, \cdots, j_{m-1}, j_m),\\
(j_r = 0, 1, \cdots, N_r - 1\ (r = 1, 2, \cdots, m)).
\end{gathered} \tag{7.28}
$$

これで, 入力 A から出力 F を得ることができました. 要される計算量は, 数えてみればすぐ分かりますように, $\mathcal{O}(N(N_1 + N_2 + \cdots + N_m))$ にまで減らされることになります. 特に, N がある整数 r の冪によって, $N = r^m$ と書ける場合には, $N_1 = N_2 = \cdots = N_m = r$ ととると, $\mathcal{O}(Nrm) = \mathcal{O}(Nr\log_r N)$ と表せます. 最初の $N = 10000$ の場合について, $r = 10$ ととると, 計算量は, $\mathcal{O}(Nr\log_r N) = \mathcal{O}(10^4 \times 10 \times 4) = \mathcal{O}(400000)$, となり, 結局 DFT を直接計算するのに必要な 1 億回の計算量が, 40 万回となり, 1/250 にまで減じられることになります[2].

以上のアルゴリズムによって, $\mathcal{O}(N^2)$ の計算量が必要だった DFT が $\mathcal{O}(N\log N)$ の計算量でできることが分かります. このアルゴリズムは Cooley and Tukey (1965) [2] によって発表され[3], Fast Fourier Transform = FFT と呼ばれています. このアルゴリズムを使うことによる計算量の減少は劇的ですが, アルゴリズム自体は非常に簡明なので 1965 年まで発見されなかったということを不思議に思われる方もいらっしゃるかもしれません. この疑問はもっともで, 最初に紹介した部分で「発見者」とかぎ括弧を付けたのもそのためです. 実際, Cooley and Tukey (1965) [2] 以前にも同様のアルゴリズムを独立に発見し, 実際, プログラミングまで行っていた人もいたのですが, FFT を実際に世に広めたのはこの 5 ページの簡潔な論文でしたので, 今では FFT の発見者の栄誉は Cooley and Tukey (1965) [2] に与えられています. このあたりの FFT の歴史については, Brigham "The Fast Fourier Transform" (1974) (邦訳: 宮川・今井訳「高速フーリエ変換」(1978)) [1] の第 1 章の末尾に詳しくまとめられています.

[2] もっとも, この場合 $10000 = (2\times 5)^4 = 2^4 \times 5^4$ とより細かく分解した方が, 計算量が $\mathcal{O}(10^4 \times (2\times 4 + 5\times 4)) = \mathcal{O}(280000)$ と, さらに少なくなります.

[3] Cooley and Tukey (1965) [2] では, 具体的な表式としては $N = 2^m$ の場合のみが示されますので, N が上記のように一般の N_r で分解される場合についての表式は, その若干の拡張である Bergland (1967) によっています.

関連図書

[1] Brigham, E. Oran, 1974: *The Fast Fourier Transform*, Prentice-Hall. (邦訳: 宮川洋・今井秀樹 訳, 1978: 高速フーリエ変換, 科学技術出版社, 262pp.)

[2] Cooley, J. W. and J. W. Tukey, 1965: An Algorithm for the Machine Calculation of Complex Fourier Series, *Math. Comp.*, **19**, 297 - 301.

[3] 日野幹雄, 1977: スペクトル解析, 朝倉書店, 300pp.

[4] 石岡圭一, 1998: FFT – 高速アルゴリズムの発見 –. 数学セミナー, **37** (1998年12月号), 日本評論社, 34 - 39.