スペクトル法について

辻　裕太

1　前書き

スペクトル法とは何かを一言でいえば、関数展開を途中で打ち切って近似するものである。適用できる境界条件に柔軟性がないが、ひとたび適用出来れば差分法などに比べて圧倒的に高精度の解を与える。この文書では二次元非圧縮粘性流体の方程式を周期境界条件の下で数値計算することを目標にして、必要な事項について解説を行う。

2　解くべき方程式

この節では二次元非圧縮粘性流体の基礎方程式を述べる。導出については適切な流体力学の本を参照すること。$(u(x,y,t), v(x,y,t))$ を (x,y) 方向の流速、ρ は密度 (非圧縮条件から定数)、$p(x,y,t)$ を圧力、$\phi = \frac{p}{\rho}$ として

$$\begin{aligned}
\frac{Du}{Dt} &= -\frac{\partial \phi}{\partial x} + \nu \nabla^2 u, \\
\frac{Du}{Dt} &= -\frac{\partial \phi}{\partial y} + \nu \nabla^2 v, \\
\frac{\partial u}{\partial x} &+ \frac{\partial v}{\partial y} = 0
\end{aligned}$$

となる。ただし、ν は動粘性係数、

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y}$$

$$\nabla^2 = \frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2}$$

としている。滑らかな解 u, v, ϕ が存在するとして、これを以下のように変形する。渦度 ζ を

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

として定義し、基礎方程式の第一式を x 偏微分したものから第二式を y 偏微分したものを引くと、

$$\frac{D\zeta}{Dt} = \nu \nabla^2 \zeta$$

という式が得られる。また、

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

から、適切な領域のもとでは

$$u = -\frac{\partial \psi}{\partial y}, \quad v = \frac{\partial \psi}{\partial x}$$

を満たす関数 ψ が存在する。(流線関数という)

$$\zeta = \nabla^2 \psi$$

という関係式が成立するので、結局基礎方程式は

$$\frac{D(\nabla^2 \psi)}{Dt} = \nu \nabla^2 \left(\nabla^2 \psi \right)$$

となり、ψ のみの方程式に書き直すことが出来る。今回はこのように書き直した方程式を

$$\Omega \coloneqq (0, 2\pi) \times (0, 2\pi)$$

という領域上で考え、境界条件として x, y 両方向の周期境界条件

$$A(x + 2\pi, y, t) = A(x, y, t), \quad A(x, y + 2\pi, t) = A(x, y, t) \quad (x, y) \in \Omega$$

を課し、初期条件として

$$\zeta_0(x, y) = \exp\left(\frac{\cos(x - x_1) + \cos(y - y_1) - 2}{\sigma^2} \right) + \exp\left(\frac{\cos(x - x_2) + \cos(y - y_2) - 2}{\sigma^2} \right) - C$$

としたものをスペクトル法を用いて数値計算する。ただし

$$\sigma = \frac{\pi}{10}, \quad x_1 = y_1 = \frac{4\pi}{5}, \quad x_2 = y_2 = \frac{6\pi}{5}$$

とし、C は

$$\iint_\Omega \zeta_0(x, y) dx dy = 0$$

を満たすように定めた定数として取っている。

3　スペクトル法の離散化

スペクトル法の離散化は、

$$\psi(x, y, t) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} a_{kl}(t) \mathrm{e}^{ikx} \mathrm{e}^{ily}$$

として ψ の二次元フーリエ級数展開を途中で打ち切って行う。方程式の左辺から右辺を引いた

$$\frac{D(\nabla^2 \psi)}{Dt} - \nu \nabla^2 \left(\nabla^2 \psi \right)$$

に上の展開式を代入し、$\mathrm{e}^{-ikx} \mathrm{e}^{-ily}$ をかけて Ω 上で積分して 0 になる事を要請すると、

$$-\left(k^2 + l^2\right) \frac{da_{kl}}{dt} = -\hat{F}_{kl} + \nu \left(k^2 + l^2\right)^2 a_{kl} \quad (-K \leq k \leq K, -L \leq l \leq L)$$

という係数についての常微分方程式が得られる。あとはこれを適当な方法で解けばよい。ただし $\hat{F}_{kl}$ は非線形項で、

$$\hat{F}_{kl} = \frac{1}{4\pi^2} \iint_\Omega \left(u \frac{\partial \zeta}{\partial x} + v \frac{\partial \zeta}{\partial y} \right) \mathrm{e}^{-ikx} \mathrm{e}^{-ily} dx dy$$

としている。収束の議論については後の節に記す。

4　変換法

前節で定義された微分方程式

$$-\left(k^2+l^2\right)\frac{da_{kl}}{dt}=-\hat{F}_{kl}+\nu\left(k^2+l^2\right)^2 a_{kl}\quad(-K\leq k\leq K,-L\leq l\leq L)$$

を数値計算したい。それ自体は例えば 4 段 4 次 Runge-Kutta 法などで行うことが出来るが、その際に非線形項

$$\hat{F}_{kl}=\frac{1}{4\pi^2}\iint_{\Omega}\left(u\frac{\partial\zeta}{\partial x}+v\frac{\partial\zeta}{\partial y}\right)\mathrm{e}^{-ikx}\mathrm{e}^{-ily}dxdy$$

の評価が問題になる。ここでは積分の式で書いているが、ψ の展開式から

$$\begin{aligned}
u(x,y,t)=-\frac{\partial\psi}{\partial y}&=\sum_{k=-K}^{K}\sum_{l=-L}^{L}-ila_{kl}(t)\mathrm{e}^{ikx}\mathrm{e}^{ily}\\
v(x,y,t)=\frac{\partial\psi}{\partial x}&=\sum_{k=-K}^{K}\sum_{l=-L}^{L}ika_{kl}(t)\mathrm{e}^{ikx}\mathrm{e}^{ily}\\
\frac{\partial\zeta}{\partial x}(x,y,t)&=\sum_{k=-K}^{K}\sum_{l=-L}^{L}-ik(k^2+l^2)a_{kl}(t)\mathrm{e}^{ikx}\mathrm{e}^{ily}\\
\frac{\partial\zeta}{\partial y}(x,y,t)&=\sum_{k=-K}^{K}\sum_{l=-L}^{L}-il(k^2+l^2)a_{kl}(t)\mathrm{e}^{ikx}\mathrm{e}^{ily}
\end{aligned}$$

となるので、一応は次のように積分が計算できる。

$$\begin{aligned}
u\frac{\partial\zeta}{\partial x}&=\left(\sum_{p=-K}^{K}\sum_{q=-L}^{L}-iqa_{pq}\mathrm{e}^{ipx}\mathrm{e}^{iqy}\right)\times\left(\sum_{r=-K}^{K}\sum_{s=-L}^{L}-ir(r^2+s^2)a_{rs}\mathrm{e}^{irx}\mathrm{e}^{isy}\right)\\
&=\sum_{p=-K}^{K}\sum_{q=-L}^{L}\sum_{r=-K}^{K}\sum_{s=-L}^{L}-qr(r^2+s^2)a_{rs}a_{pq}\mathrm{e}^{i(p+r)x}\mathrm{e}^{i(q+s)y}
\end{aligned}$$

となるので、これに $\mathrm{e}^{-ikx}\mathrm{e}^{-ily}$ をかけて Ω 上で積分すると、直交性から

$$\sum_{p=-K}^{K}\sum_{q=-L}^{L}\sum_{r=-K}^{K}\sum_{s=-L}^{L}-qr(r^2+s^2)a_{rs}a_{pq}\delta_{p+r,k}\delta_{q+s,l}$$

が得られる。ただし

$$\delta_{i,j}=\begin{cases}1 & (i=j)\\ 0 & (i\neq j)\end{cases}$$

である。すると、この定義から $p=k-r,q=l-s$ の場合を除いて $\delta_{p+r,k}=\delta_{q+s,l}=0$ となるので、上の式は

$$\sum_{r=-K}^{K}\sum_{s=-L}^{L}-(l-s)r(r^2+s^2)a_{rs}a_{(k-r)(l-s)}$$

として書き直される。$v\frac{\partial\zeta}{\partial y}$ についても同様である。以上のように $\hat{F}_{kl}$ は計算できるが、このままでは k,l を一つ決めるごとにおおよそ $2K\times 2L$ 回の計算が必要になり、全体では $4K^2\times 4L^2$ 回の計算が必要になる。これは差分法などに比べて計算コストが大きく、このためにスペクトル法は利用されることが少なかった経緯がある。しかし、それを解消するべく高速フーリエ変換を用いた変換法という手法が開発された。それを解説する。改めて

$$\hat{F}_{kl}=\frac{1}{4\pi^2}\iint_{\Omega}\left(u\frac{\partial\zeta}{\partial x}+v\frac{\partial\zeta}{\partial y}\right)\mathrm{e}^{-ikx}\mathrm{e}^{-ily}dxdy$$

を計算したいと考える。右辺の積分が計算出来ればよいのだから、次のように数値積分することを考える (!)
まずは一次元の場合について、

$$\frac{1}{2\pi}\int_0^{2\pi}f(x,t)dx$$

という積分において、被積分関数が

$$f(x,t)=\sum_{k=-K}^{K}\hat{f}_k(t)\exp(ikx)$$

として表示されている場合を考える。この時積分は

$$\sum_{k=-K}^{K}\hat{f}_k(t)\frac{1}{2\pi}\int_0^{2\pi}\exp(ikx)dx$$

とかけるから、分点の数を M 個として積分

$$\frac{1}{2\pi}\int_0^{2\pi}\exp(ikx)dx=\begin{cases}1 & (k=0)\\ 0 & (k\neq 0)\end{cases}$$

の左辺をリーマン和で近似すると

$$\frac{1}{M}\sum_{m=0}^{M}\exp\left(ik\frac{2\pi m}{M}\right)$$

となり、等比数列の和の公式から、$r=\exp(2\pi i\frac{k}{M})$ として

$$\frac{1}{M}\sum_{m=0}^{M}\exp\left(ik\frac{2\pi m}{M}\right)=\begin{cases}1 & (r=1)\\ 0 & (r\neq 0)\end{cases}$$

となる。これが厳密な積分の値に一致するには、全ての $-K\leq k\leq K$ に対して、

$$r=\exp\left(2\pi i\frac{k}{M}\right)\begin{cases}=1 & (k=0)\\ \neq 0 & (k\neq 0)\end{cases}$$

が成立すればよいことが分かる。前者は常に成立する。$M\geq K+1$ として取れば、後者も成立する。このようにして M を取っておけば、

$$\frac{1}{2\pi}\int_0^{2\pi}f(x,t)dx$$

という積分の計算の代わりに、$x_m = \frac{2\pi}{M}$ として

$$\frac{1}{M}\sum_{m=0}^{M} f(x_m, t)$$

を計算すればよいことになる。すなわち、被積分関数のフーリエ展開の波数の最大値よりも大きく分点を取って置けば、数値積分でも厳密に値を求めることが出来ることが分かった。

$$\hat{F}_{kl} = \frac{1}{4\pi^2}\iint_{\Omega}\left(u\frac{\partial\zeta}{\partial x} + v\frac{\partial\zeta}{\partial y}\right)\mathrm{e}^{-ikx}\mathrm{e}^{-ily}dxdy$$

についても同様にして、

$$f(x, y, t) = \left(u\frac{\partial\zeta}{\partial x} + v\frac{\partial\zeta}{\partial y}\right)$$

おくと、被積分関数

$$f(x, y, t)\mathrm{e}^{-ikx}\mathrm{e}^{-ily}$$

のフーリエ級数の波数の絶対値は x, y 方向それぞれについて最大で $3K, 3L$ であるから、$M \geq 3K+1, N \geq 3L+1$ としてとれば、$\hat{F}_{kl}$ は積分を計算する代わりに $x_m = \frac{2\pi}{M}, y_n = \frac{2\pi}{N}$ として

$$\frac{1}{M}\frac{1}{N}\sum_{m=0}^{M}\sum_{n=0}^{N} f(x_m, y_n, t)\mathrm{e}^{-ikx_m}\mathrm{e}^{-ily_n}$$

を計算することでも求められる。問題はこれをどうやって計算するかだが、この形の級数は次節で説明する高速フーリエ変換と呼ばれるアルゴリズムを用いて高速 ($2K \times 2L$ よりずっと速く) に計算することが出来る。このような手法を変換法という。

5 高速フーリエ変換

5.1 計算の原理

まずは一次元の場合について、

$$x(\hat{k}) = \sum_{k=0}^{N-1} A(k)\mathrm{e}^{2\pi i\frac{k\hat{k}}{N}} \quad (\hat{k} = 0, \ldots, N-1)$$

の形の離散的複素逆フーリエ変換、すなわち $x(\hat{k}), A(k)$ が共に複素数である場合を考える。正変換はこれの複素共役を取ればよい。$N = PS$ と自然数の積に分解出来るとする。

$$0 \leq \hat{p} \leq P-1 \quad 0 \leq \hat{s} \leq S-1$$

となる $\hat{p}, \hat{q}$ に対して

$$X_S(S\hat{p} + \hat{s}) \coloneqq \mathrm{e}^{2\pi i\frac{\hat{p}\hat{s}}{PS}}\sum_{s=0}^{S} A(Ps + \hat{p})\mathrm{e}^{2\pi i\frac{\hat{s}s}{S}}$$

として X_s を定義すると、$S=1, P=N$ の時は

$$X_1(\hat{p}) = A(\hat{p}) \quad (\hat{p}=0,\ldots,N-1)$$

$S=N, P=1$ の時は

$$X_N(\hat{s}) = \sum_{s=0}^{S-1} A(s)\mathrm{e}^{2\pi i \frac{s\hat{s}}{S}} = x(\hat{s}) \quad (\hat{s}=0,\ldots,S-1)$$

となるので、$X_1(\hat{p}) = A(\hat{p})$ として入力し、$X_N(\hat{s}) = x(\hat{s})$ を求めるアルゴリズムを作ればよい。$N=PS$ かつ $S=QR$ と分解されるとき、$0 \le s, \hat{s} \le S-1$ の範囲にある $s, \hat{s}$ を R, Q で割って

$$\hat{s} = R\hat{q} + \hat{r}, \quad s = Qr + q$$

と書くと、

$$0 \le r, \hat{r} \le R-1, 0 \le q, \hat{q} \le Q-1$$

であり、$s, \hat{s}$ を添え字とする和は

$$\sum_{s=0}^{S-1} a_s = \sum_{r=0}^{R-1}\sum_{q=0}^{Q-1} a_{Rq+r}$$
$$\sum_{\hat{s}=0}^{S-1} a_{\hat{s}} = \sum_{\hat{r}=0}^{R-1}\sum_{\hat{q}=0}^{Q-1} a_{R\hat{q}+\hat{r}}$$

のように書き直すことが出来る。これを用いると、

$$\begin{aligned}
X_S(S\hat{p}+\hat{s}) &= \mathrm{e}^{2\pi i \frac{\hat{p}\hat{s}}{PS}} \sum_{s=0}^{S} A(Ps+\hat{p})\mathrm{e}^{2\pi i \frac{\hat{s}s}{S}} \\
&= \mathrm{e}^{2\pi i \frac{\hat{p}(R\hat{q}+\hat{r})}{PQR}} \sum_{q=0}^{Q-1}\sum_{r=0}^{R-1} A(P(Qr+q)+\hat{p})\mathrm{e}^{2\pi i \frac{(R\hat{q}+\hat{r})(Qr+q)}{QR}} \\
&= \mathrm{e}^{2\pi i \frac{\hat{p}\hat{q}}{PQ}} \mathrm{e}^{2\pi i \frac{\hat{p}\hat{r}}{PQR}} \sum_{q=0}^{Q-1}\sum_{r=0}^{R-1} A(PQr+Pq+\hat{p})\mathrm{e}^{2\pi i \frac{\hat{q}q}{Q}} \mathrm{e}^{2\pi i \frac{\hat{r}r}{R}} \mathrm{e}^{2\pi i \frac{\hat{q}\hat{r}}{QR}} \\
&= \mathrm{e}^{2\pi i \frac{\hat{p}\hat{q}}{PQ}} \sum_{q=0}^{Q-1} \mathrm{e}^{2\pi i \frac{\hat{p}\hat{r}}{PQR}} \mathrm{e}^{2\pi i \frac{\hat{q}q}{Q}} \mathrm{e}^{2\pi i \frac{\hat{q}\hat{r}}{QR}} \sum_{r=0}^{R-1} A(PQr+Pq+\hat{p})\mathrm{e}^{2\pi i \frac{\hat{r}r}{R}} \\
&= \mathrm{e}^{2\pi i \frac{\hat{p}\hat{q}}{PQ}} \sum_{q=0}^{Q-1} \mathrm{e}^{2\pi i \frac{\hat{q}q}{Q}} \mathrm{e}^{2\pi i \frac{(Pq+\hat{p})\hat{r}}{PQR}} \sum_{r=0}^{R-1} A(PQr+Pq+\hat{p})\mathrm{e}^{2\pi i \frac{\hat{r}r}{R}} \\
&= \mathrm{e}^{2\pi i \frac{\hat{p}\hat{q}}{PQ}} \sum_{q=0}^{Q-1} \mathrm{e}^{2\pi i \frac{\hat{q}q}{Q}} X_R(R(Pq+\hat{p})+\hat{r})
\end{aligned}$$

となるので、X_S の S を R から QR に増やす漸化式

$$X_{QR}(QR\hat{p}+R\hat{q}+\hat{r}) = \mathrm{e}^{2\pi i \frac{\hat{p}\hat{q}}{PQ}} \sum_{q=0}^{Q-1} \mathrm{e}^{2\pi i \frac{\hat{q}q}{Q}} X_R(PRq+R\hat{p}+\hat{r})$$
$$(0 \le \hat{p} \le P-1, \quad 0 \le \hat{q} \le Q-1, \quad 0 \le \hat{r} \le R-1)$$

が得られた。特に $N = 2^l$ と書けるような場合は、

$$X_1 \to X_2 \to X_4 \to X_8 \to \ldots \to X_{2^l}$$

となり、各ステップの計算が $2N$、ステップ数は $\log_2 N = l$ となるので、全体の計算回数は $2N\log_2 N$ となる。これは普通に和を計算したときの $N \times N$ よりも遥かに小さくなる。

5.2 プログラム例

$N = 2^l$ のときのプログラム例を示す。この場合は常に $Q = 2$ である。また、$l = 6$ としている。

```
/*この部分は次回以降のプログラム例では省略する*/
#include<iostream>
#include<math.h>
#include<stdio.h>
#include<fstream>
#include<cstdlib>
#include<complex>

using namespace std;

/*初期条件の複素配列と返却用配列を受け取って逆 DFT を返す関数*/
/*入力の長さは 64 項としている*/
/*正変換は得られた結果の共役を取ればよい*/
void INV_DFFT_64(complex<double>* a,complex<double>* X){
const int N = 64;
double pi = acos(-1.0);//円周率
complex<double> i_cpx(0.0,1.0);//虚数単位

/*配列の初期化*/
for(int i=0;i<N;i++){
X[i] = a[i];
}

/*必要な変数*/
int l = 6;
int P = N;
int Q = 2;
int R;
/*N=2^l と書けるとき*/
/*計算に使う配列を動的に確保*/
```

```
complex<double> *XR = new complex<double>[N];
complex<double> *XQR = new complex<double>[N];

/*1 次元配列で書く*/
for(int j=1;j<=l;j++){
P = P / 2;
R = N / (P * Q);

/*XR は一つ前のステップで得られた配列で初期化*/
for(int i=0;i<N;i++){
XR[i] = X[i];
}

/*XR から XQR を計算する*/
for(int p=0;p<P;p++){
for(int r=0;r<R;r++){
XQR[Q*R*p + r] = XR[R*p+r] + XR[P*R+R*p+r];
XQR[Q*R*p + R + r] = exp(2.0*pi*i_cpx*double(p)/(2.0*double(P)))*(XR[R*p+r] - XR[P*R+R*p+r]);
}
}

/*得られた配列 XQR をもとに X を更新*/
for(int i=0;i<N;i++){
X[i] = XQR[i];
}
}
/*動的確保したら必ず開放する!!*/
delete[] XR;
delete[] XQR;
}
```

6 計算量の削減

複素数から複素数への変換は先のように書けるが、実数値関数の離散フーリエ変換を求める、あるいは離散逆フーリエ変換でフーリエ係数から実数値関数の値を求めるような場合は、入力か出力の一方は実数である。そのまま上記の定義式で計算しようとすると、配列の虚部の部分が無駄になってしまうが、このような場合は計算量を削減できることが知られている。これについて説明する。以下、N は常に偶数であるとする。

6.1 正変換

$$x(j) \in \mathbb{R} \quad (j = 0, \ldots, N-1)$$

として、離散フーリエ正変換

$$A(k) = \frac{1}{N} \sum_{j=0}^{N-1} x(j) \exp\left(-2\pi i \frac{jk}{N}\right) \quad (k = 0, 1, \ldots, N-1)$$

を計算する。この長さ N のフーリエ変換を以下のように長さ $N/2$ の複素フーリエ変換に押し込むことが出来る。まず $x(j) \in \mathbb{R}$ だから、$a \in \mathbb{C}$ の共役を a^* と書くことにすると、

$$\begin{aligned} A(N-k) &= \frac{1}{N} \sum_{j=0}^{N-1} x(j) \exp\left(-2\pi i \frac{j(N-k)}{N}\right) \\ &= \frac{1}{N} \sum_{j=0}^{N-1} x(j) \exp\left(2\pi i \frac{jk}{N}\right) \\ &= A^*(k) \quad (k = 0, 1, \ldots, N-1) \end{aligned}$$

という関係式が成立し、

$$A(k) \quad (k = 0, \ldots, N/2)$$

だけを求めておけば残りの成分は共役を取ることで求めることが出来る。j の偶奇で場合分けをすると、

$$\begin{aligned} A(k) &= \frac{1}{N} \sum_{j=0}^{N-1} x(j) \exp\left(-2\pi i \frac{jk}{N}\right) \\ &= \frac{1}{N} \sum_{j=0}^{N/2-1} x(2j) \exp\left(-2\pi i \frac{2jk}{N}\right) + \frac{1}{N} \sum_{j=0}^{N/2-1} x(2j+1) \exp\left(-2\pi i \frac{(2j+1)k}{N}\right) \\ &= \frac{1}{N} \sum_{j=0}^{N/2-1} x(2j) \exp\left(-2\pi i \frac{jk}{N/2}\right) + \frac{1}{N} \sum_{j=0}^{N/2-1} x(2j+1) \exp\left(-2\pi i \frac{jk}{N/2}\right) \exp\left(-2\pi i \frac{k}{N}\right) \\ &= \frac{1}{N} \left\{ B_0^*(k) + B_1^*(k) \exp\left(-2\pi i \frac{k}{N}\right) \right\} \end{aligned}$$

ここで

$$\begin{aligned} B_0(k) &\coloneqq \sum_{j=0}^{N/2-1} x(2j) \exp\left(2\pi i \frac{jk}{N/2}\right), \\ B_1(k) &\coloneqq \sum_{j=0}^{N/2-1} x(2j+1) \exp\left(2\pi i \frac{jk}{N/2}\right) \end{aligned}$$

としている。後は $x(j)$ から B_0, B_1 を求めればよい。そのために

$$y(j) = x(2j) + ix(2j+1) \quad (j = 0, 1, \ldots, N/2-1)$$

として長さ $N/2$ の複素データを作っておき、長さ $N/2$ の $y(j)$ の複素フーリエ逆変換を $C(k)$ とすると、

$$C(k) \coloneqq \sum_{j=0}^{N/2-1} y(j) \exp\left(2\pi i \frac{jk}{N/2}\right) = B_0(k) + iB_1(k),$$
$$C(N/2-k) = \sum_{j=0}^{N/2-1} y(j) \exp\left(-2\pi i \frac{jk}{N/2}\right) = B_0^*(k) + iB_1^*(k) \quad (k = 1, \ldots, N/2-1)$$

となる。これらを連立させて B_0, B_1 について解くと、

$$B_0^*(k) = \frac{1}{2}\{C(N/2-k) + C^*(k)\}$$
$$B_1^*(k) = \frac{1}{2i}\{C(N/2-k) - C^*(k)\} \quad (k = 1, \ldots, N/2-1)$$

となる。$k = 0$ の時は、

$$B_0(0) = \mathrm{Re}(C(0))$$
$$B_1(0) = \mathrm{Im}(C(0))$$

となる。以上から、$k = 1, \ldots, N/2-1$ のときは

$$A(k) = \frac{1}{2N}\left[\{C(N/2-k) + C^*(k)\} + \frac{1}{i}\exp\{-2\pi i \frac{k}{N}\}\{C(N/2-k) - C^*(k)\}\right]$$

$k = 0, N/2$ のときは

$$A(0) = \frac{1}{N}\{\mathrm{Re}(C(0)) + \mathrm{Im}(C(0))\}$$
$$A(N/2) = \frac{1}{N}\{\mathrm{Re}(C(0)) - \mathrm{Im}(C(0))\}$$

として求めることが出来る。$k = N/2+1, \ldots, N$ はこれの共役を取ればよい。プログラム例 (C++) を以下に記す。

```
/*実数の配列を入力すると長さが半分の複素 FFT を利用して正フーリエ変換を計算*/
/*ただし配列の長さは偶数に限る*/
/*今回は入力の長さは 64*/
void REGULAR_DFFT(double* x,complex<double>* A){
const int N = 64;//入力した配列の長さ
complex<double> i(0.0,1.0);//虚数単位
double pi = acos(-1.0);//円周率
complex<double> e = -2.0*pi*i;//計算用定数
/*計算用配列を動的確保*/
complex<double> *y = new complex<double>[N/2];
complex<double> *C = new complex<double>[N/2];

/*入力データを長さ半分の複素データに入れる*/
for(int j=0;j<N/2;j++){
```

```
y[j] = complex<double>(x[2*j],x[2*j+1]);
}

/*y[j] から複素フーリエ逆変換*/
/*FFT を利用して戻り値を保存*/
/*逆変換の関数を次の名前で書いておく*/
INV_DFFT_32(y,C);

/*A の独立な要素を C から計算*/
for(int k=1;k<N/2;k++){
A[k] = ((C[N/2-k]+conj(C[k])) - i*exp(e*complex<double>(k)/complex<double>(N))*
(C[N/2-k] - conj(C[k])))/complex<double>(2*N);
}

/*こちら側は前半部分の共役でよい*/
for(int k=1;k<N/2;k++){
A[N-k] = conj(A[k]);
}

/*残りの 0,N/2 は C0 から計算する*/
A[0] = (real(C[0])+imag(C[0]))/double(N);
A[N/2] = (real(C[0])-imag(C[0]))/double(N);

/*動的確保した配列は開放する*/
delete[] C;
delete[] y;
}
```

6.2 逆変換

今度は $A(k)$ を入力として逆変換

$$x(j) = \sum_{k=0}^{N-1} A(k) \exp\left(2\pi i \frac{jk}{N}\right) \quad (j = 0, \ldots, N-1)$$

を計算したい。正変換の場合と同様にして、

$$x(j) \in \mathbb{R}$$

という条件を付けると、

$$A(N-k) = A^*(k)$$

となる。k についての和を $N/2$ より小さい部分と大きい部分に分けると、

$$x(j) = A(0) + (-1)^j A(N/2) + \sum_{k=1}^{N/2-1} A(k) \exp\left(2\pi i \frac{jk}{N}\right) + \sum_{k=1}^{N/2-1} A(N-k) \exp\left(2\pi i \frac{j(N-k)}{N}\right)$$
$$= A(0) + (-1)^j A(N/2) + \sum_{k=1}^{N/2-1} A(k) \exp\left(2\pi i \frac{jk}{N}\right) + \sum_{k=1}^{N/2-1} A^*(k) \exp\left(-2\pi i \frac{jk}{N}\right)$$

最後の項は $A(k)$ の条件から次のように変形することが出来る。

$$\sum_{k=1}^{N/2-1} A^*(k) \exp\left(-2\pi i \frac{jk}{N}\right) = \sum_{k'=1}^{N/2-1} A^*(N/2-k') \exp\left(-2\pi i \frac{j(N/2-k')}{N}\right)$$
$$= \sum_{k'=1}^{N/2-1} A^*(N/2-k')(-1)^j \exp\left(-2\pi i \frac{jk'}{N}\right)$$

これを用いて j の偶奇で場合分けすると

$$x(2j) = A(0) + A(N/2) + \sum_{k=1}^{N/2-1} A(k) \exp\left(2\pi i \frac{jk}{N/2}\right) + \sum_{k=1}^{N/2-1} A^*(N/2-k) \exp\left(2\pi i \frac{jk}{N/2}\right)$$
$$x(2j+1) = A(0) - A(N/2) + \sum_{k=1}^{N/2-1} A(k) \exp\left(2\pi i \frac{jk}{N/2}\right) \exp\left(2\pi i \frac{k}{N}\right) - \sum_{k=1}^{N/2-1} A^*(N/2-k) \exp\left(2\pi i \frac{jk}{N/2}\right) \exp\left(2\pi i \frac{k}{N}\right)$$

となり、

$$x(2j) + x(2j+1) = \sum_{k=0}^{N/2-1} D(k) \exp\left(2\pi i \frac{jk}{N/2}\right) \quad (j = 0, 1, \ldots, N/2-1)$$

と書ける。ただし

$$D(k) \coloneqq \{A(k) + A^*(N/2-k)\} + i \exp\left(2\pi i \frac{k}{N}\right) \{A(k) - A^*(N/2-k)\} \quad (k = 0, 1, \ldots, N/2-1)$$

としている。すなわち、$D(k)$ という長さ $N/2$ の複素配列を作り、その FFT の結果の実部、虚部を取ればよい。プログラム例を下に記す。

```
/*実 FFT 逆変換を長さが半分の複素 FFT を利用して計算する*/
/*入力は 64 項とする*/
void REAL_INV_DFFT(complex<double>* A,double* x){
const int N = 64;//入力データの長さ
complex<double> i(0.0,1.0);//虚数単位
double pi = acos(-1.0);//円周率
/*計算用配列を動的確保*/
complex<double> *D = new complex<double>[N/2];
complex<double> *X = new complex<double>[N/2];

/*入力データを長さ半分の複素配列に入れる*/
```

```
for(int k=0;k<N/2;k++){
D[k] = A[k] + conj(A[N/2-k]) + i*exp(2.0*pi*i*complex<double>(k)/complex<double>(N))*(A[k]-conj(A[N/
}

/*D から FFT したものを保存*/
INV_DFFT_32(D,X);

for(int k=0;k<N/2;k++){
x[2*k] = real(X[k]);
x[2*k+1] = imag(X[k]);
}

/*動的確保した配列は開放する*/
delete[] D;
delete[] X;
}
```