数理解析研究所講究録
第1642巻 2009年 184-196

球面上点渦相互作用の高速tree-codeアルゴリズム
Fast tree-code for point-vortex interaction on a sphere

坂上貴之 (Takashi Sakajo),
北海道大学 (Hokkaido University)
科学技術振興機構さきがけ (JST Presto)
E-mail: sakajo@math.sci.hokudai.ac.jp

December 8, 2008

1 イントロダクション

本稿では半径 R の球面上の非粘性・非圧縮流れの時間発展を数値計算するための高速アルゴリズムを提案する．差分法やスペクトル法で球面上の流れの計算を行う場合には，北極や南極でのメッシュに現れるみかけの特異性の影響や，球面調和関数への展開に時間がかかるなど平面の場合にはない特有の問題があるが，本アルゴリズムでは球面上の離散渦法をベースとし，その離散渦間の相互作用を高速に計算することによって，こうした問題を回避することを目的としている．離散渦法は，二次元流れにおいて渦度が流体粒子の軌道に沿って保存される量であるという性質に基づいて開発された数値計算法である．すなわち，初期において渦度が存在しない領域は時間発展を通じて渦無しなので，初期において渦度が存在する領域だけを点渦で近似し，それらの時間発展を計算することによって渦度場全体を近似する方法である．

まず，球面上にある初期渦度存在領域を N 個の小領域に分割し，各小領域内の代表点に点渦を置く．その位置を球面座標系で (θ_m, ϕ_m), $(m = 1, \ldots, N)$ と表し，点渦の強さ Γ_m は各小領域の中に含まれる全循環量で定義する．この時，初期渦度領域は次のような δ 関数の線形結合で近似される．

$$\omega_0(\theta, \phi) \approx \frac{1}{\sin\theta} \sum_{m=1}^{N} \Gamma_m \delta(\theta - \theta_m, \phi - \phi_m).$$

この仮定の下で，球面上のオイラー方程式は $m = 1, 2, \ldots, N$ に対して，以下のような $2N$ 次元の常微分方程式系に帰着される [9].

$$\dot{\theta}_m = -\frac{1}{4\pi R^2} \sum_{j \neq m}^{N} \frac{\Gamma_j \sin\theta_j \sin(\phi_m - \phi_j)}{R^2 + \sigma^2 - \cos\theta_m \cos\theta_j - \sin\theta_m \sin\theta_j \cos(\phi_m - \phi_j)}, \tag{1}$$

$$\dot{\phi}_m = -\frac{1}{4\pi R^2 \sin\theta_m} \sum_{j \neq m}^{N} \frac{\Gamma_j [\cos\theta_m \sin\theta_j \cos(\phi_m - \phi_j) - \sin\theta_m \cos\theta_j]}{R^2 + \sigma^2 - \cos\theta_m \cos\theta_j - \sin\theta_m \sin\theta_j \cos(\phi_m - \phi_j)}. \tag{2}$$

なお，この方程式の中に含まれる σ は，二つの点渦が非常に近い位置にある時に積分 (1) と (2) が特異な振る舞いをするのを防ぐために導入された正則化パラメータである．特に $\sigma = 0$ の時には，この近似方法を *the point vortex method*（点渦法）と呼び，$\sigma \neq 0$ の時を *the vortex blob method* と呼ぶ．以下これらをまとめて離散渦法 (the vortex mehtods) と呼ぶが，この数値計算法の収束性や実際の数値計算例などについての詳細は [2, 7] に詳しい．

渦法の定式化は簡単なものの，実際の数値計算を行う際に大きな困難が伴う．すなわち，与えられた一つの点渦に対して常微分方程式の右辺 (1) と (2) を評価するために $O(N)$ の計算が必要であるため，全点渦に対して計算を行うと $O(N^2)$ 回の計算コストがかかるのである．同様の問題点は N 個の点電荷や重力多体系の問題の数値計算においても見られ，これを解決するために高速 tree-code アルゴリズム (*the fast tree-code algorithm*) [1] や高速多重極展開法 (*the fast multipole method*)[5, 6] などが提案されている．二次元オイラー方程式に対する渦法に対しても Draghicescu[3] らが積分核のテイラー展開に基づく高速 tree-code アルゴリズムを開発しており，二次元渦層の数値計算などに有効に利用されている [4, 10]. また，この方法は三次元の渦法に対しても拡張され，三次元渦層の長時間数値計算にも威力を発揮している [8, 11]. なお，三次元オイラー方程式において渦度の大きさは流体粒子に沿って保存されないので，渦度の引き伸ばしに対応するメカニズムをアルゴリズムに取り入れる必要がある．

ここでは，これらの高速 tree-code アルゴリズムを球面上の点渦間の相互作用の評価に拡張する．この拡張はもちろん原理的には可能である．すなわち，方程式の (1) と (2) の和の中にある分母関数を単純にテイラー展開することで構成できるからである．しかしながら，この分母を高次までテイラー展開することは面倒であり，たとえそれができたとしても，それらを実際に数値計算する場合には非常に時間がかかってしまうために，せっかくの高速アルゴリズムのメリットが全く活かされなくなってしまう．この問題点を克服するために，我々は敢えて問題を三次元空間の中に球面を埋め込んで問題を以下のように再定式化する．球面上にある点渦は $\mathbb{R}^3$ 座標系で表示すると，

$$\boldsymbol{x}_m = (x_m(t), y_m(t), z_m(t)) = (R\sin\theta_m\cos\phi_m, R\sin\theta_m\sin\phi_m, R\cos\theta_m).$$

となり，この時，N 個の点渦の時間発展は以下で与えられる．

$$\dot{\boldsymbol{x}}_m = -\frac{1}{4\pi R}\sum_{j\neq m}^{N}\Gamma_j\frac{\boldsymbol{x}_m\times\boldsymbol{x}_j}{R^2+\sigma^2-\boldsymbol{x}_m\cdot\boldsymbol{x}_j}, \quad m=1,2,\ldots,N. \tag{3}$$

我々はこの方程式に対して高速 tree-code アルゴリズムを構成する．もとの方程式に比べて計算する変数の数は多くなるが，この方程式の右辺の和に現れる高次のテイラー係数は陽的に計算できるので，高速アルゴリズムは効果的に実行できることが期待できる．

本報告は，Journal of Computational and Applied Mathmatics に掲載された論文 [13] の内容に基づいて，その概要をレビューするものであるが，この日本語版では本アルゴリズムを実装する上のヒントなども詳しく追記する．この方法を使った具体的な渦層の運動などへの応用例については原著論文を参照して欲しい．本報告の構成は以下の通りである．第二章でアルゴリズムの詳細と実装，誤差評価や計算量評価の結果を述べる．第三章ではテスト問題でこれらの効果を報告し，最後の章では，それらのまとめとこのアルゴリズムの回転流体への応用の可能性について議論する．

2 球面版Fast tree-codeアルゴリズム

2.1 階層メッシュの構成

イントロダクションでも述べたように，球面における点渦相互作用の高速 tree-code アルゴリズムのポイントは球面を三次元空間に埋め込んで，あたかも三次元の上での高速算法であるかのように計算を行うところにある．そこで，考える計算領域も球面をすっぽりと含む一辺の長さが $2R(1+\delta)$ の立方体を考える．すなわち，計算領域 $\mathcal{B}$ は次で与えられる．

$$\mathcal{B} = [-(1+\delta)R, (1+\delta)R]^3.$$

ここで $\delta > 0$ は，この立方体が半径 R の円を「すっぽり」含むようにとるために必要なだけなので，非常に小さい値をとっておけばよい．実際には後述するようにアルゴリズムの誤差評価や計算回数の評価をする上で $R < 0.5$ であることを使うので，計算領域を長さ1の立方体にしておいて，球面を適当にリスケールしてこの計算領域におさまるようにする．

この計算領域 $\mathcal{B}$ に対して，次のような再帰的分割を行って，計算領域の中に小さな矩形からなる tree 構造を導入する．まず，与えられた矩形領域 $\tau = [x_1, x_2) \times [y_1, y_2) \times [z_1, z_2)$ に対して，次のような再帰的アルゴリズムを定義する．加えてこのアルゴリズムでは，高速算法に必要なパラメータで矩形領域 τ に付随する以下のものを初期化する．

- τ の中心 $\boldsymbol{y}_\tau$ と半径 $\rho(\tau)$：$\boldsymbol{y}_\tau = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}, \frac{z_1+z_2}{2}\right)$, $\rho(\tau) = \sup_{\boldsymbol{y}\in\tau} |\boldsymbol{y} - \boldsymbol{y}_\tau|$,
- τ の中に含まれる点渦のリスト $L(\tau)$. ただし，このリスト構造は計算領域の tree 構造におけるもっとも小さな矩形に対してのみ定義される．
- τ の中心 $\boldsymbol{y}_\tau$ 近傍におけるテイラー展開の高階微分係数の値を格納するデータ構造．多重指数 $\boldsymbol{k} = (k_1, k_2, k_3)$, $|\boldsymbol{k}| = k_1 + k_2 + k_3 < \lambda$ に対して，$A_\tau^{\boldsymbol{k}}$, $B_\tau^{\boldsymbol{k}}$, $C_\tau^{\boldsymbol{k}}$ の三つを用意する．定義は後ほど与える．

メッシュ分割の再帰アルゴリズムは以下の通り．なお，入力するデータは，矩形 τ, 階層のレベルを表す整数 j, テイラー展開による近似の階数を表す λ，および再帰の終了条件を決める整数 l である．

Algorithm 1. *(計算領域に tree 構造を入れる)*

GenerateMesh(τ, j, l)
  if $j = 3l$ return;
  $\boldsymbol{y}_\tau = (c_{\tau 1}, c_{\tau 2}, c_{\tau 3}) = \left(\frac{1}{2}(x_1+x_2), \frac{1}{2}(y_1+y_2), \frac{1}{2}(z_1+z_2)\right)$;
  Compute the radius of τ, $\rho(\tau)$;
  For all $\boldsymbol{k} = (k_1, k_2, k_3)$, $|\boldsymbol{k}| = k_1 + k_2 + k_3 \le \lambda - 1$, initialize $A_\tau^{\boldsymbol{k}}$, $B_\tau^{\boldsymbol{k}}$, $C_\tau^{\boldsymbol{k}}$;
  Initialize the list of the near-fields, $L(\tau)$;
  if $j \mod 3 = 1$
    $\tau_1 = [x_1, c_{\tau 1}) \times [y_1, y_2) \times [z_1, z_2)$; $\tau_2 = [c_{\tau 1}, x_2) \times [y_1, y_2) \times [z_1, z_2)$;
  else if $j \mod 3 = 2$
    $\tau_1 = [x_1, x_2) \times [y_1, c_{\tau 2}) \times [z_1, z_2)$; $\tau_2 = [x_1, x_2) \times [c_{\tau 2}, y_2) \times [z_1, z_2)$;
  else if $j \mod 3 = 0$
    $\tau_1 = [x_1, x_2) \times [y_1, y_2) \times [z_1, c_{\tau 3})$; $\tau_2 = [x_1, x_2) \times [y_1, y_2) \times [c_{\tau 3}, z_2)$;

```
    end
    For each τ_i, if τ_i ∩ S ≠ ∅
        set τ_i as a child box of τ;
        recursively call GenerateMesh( τ_i, j + 1, l );
        return;
    else
        set ∅ as a child of τ;
        return;
    end
    return;
end
```

End of Algorithm

この再帰アルゴリズムを計算領域 $\tau = \mathcal{B}$ に適用すると，x 成分，y 成分，z 成分の順に二等分された矩形小領域をその tree 構造の子供として持つ再帰的な計算領域の tree 構造が生成される．

次に，論文 [3] と同様に，与えられた球面上の点 $\boldsymbol{x} \in S$ に対する，*far-field*($\mathcal{F}(\boldsymbol{x})$ と書く) と *near-field*($N(\boldsymbol{x})$ と書く) と呼ばれる矩形領域の集合を定義する．

Definition 1. *(far-field と near-field)* 球面上の点 $\boldsymbol{x} \in S$ に対して，矩形集合 $\mathcal{F}(\boldsymbol{x})$ は，その中心 $\boldsymbol{y}_\tau$ が以下の条件を満たす矩形 τ の集合の中の極大集合とする．

$$\rho(\tau) \leq h^\nu \left| R^2 - \boldsymbol{x} \cdot \boldsymbol{y}_\tau \right|, \tag{4}$$

これに対して，$F(\boldsymbol{x}) = \cup\mathcal{F}(\boldsymbol{x})$ と定義する．ここで，$\nu > 0$ は後ほど高速算法の効率と精度をコントロールするために用いられるパラメータである．これに対して，点 $\boldsymbol{x}$ に対する，*near-field* を $N(\boldsymbol{x}) = \Sigma \setminus F(\boldsymbol{x})$ で定義する．

ここで採用されている far-field と near-field の定義は論文 [3] で与えられているものとほとんど同じだが，far-field の判定条件 (4) が異なっている．この条件は後の誤差解析を行う上で便利であるために導入されたものだが，三次元空間における点渦相互作用に対する高速 tree-code アルゴリズムの論文 [3] で与えられた判定条件

$$\rho(\tau) \leq h^\nu \left| \boldsymbol{x} - \boldsymbol{y}_\tau \right|. \tag{5}$$

と本質的には変わらないものである．なぜなら，球面の半径 $R < 0.5$ に対して（そしてこれはスケーリングを適切に行えばいつでもこうできる）$|R^2 - \boldsymbol{x} \cdot \boldsymbol{y}_\tau| \leq |\boldsymbol{x} - \boldsymbol{y}_\tau|$ が任意の $\boldsymbol{y}_\tau \in \mathbb{R}^3$, $|\boldsymbol{x}| = 0.5$ に対して成り立つため，新しい判定条件で far-field と判断されたものは，もとの条件でも far-field となっているからである．

2.2 Far-field における近似計算

高速 tree-code アルゴリズムでは，N 個の点渦 $\boldsymbol{y}_j$ $(j = 1, \ldots, N)$ が点 $\boldsymbol{x}$ の上に誘導する以下の速度場 $\boldsymbol{u}_N(\boldsymbol{x}, t)$ を高速に評価するものである．

$$\boldsymbol{u}_N(\boldsymbol{x}, t) = -\frac{1}{4\pi R} \sum_{j=1}^{N} \Gamma_j \frac{\boldsymbol{x} \times \boldsymbol{y}_j}{R^2 + \sigma^2 - \boldsymbol{x} \cdot \boldsymbol{y}_j} \equiv -\frac{1}{4\pi R} \sum_{j=1}^{N} \Gamma_j \boldsymbol{\gamma}(\boldsymbol{x}, \boldsymbol{y}_j) \mathcal{D}(\boldsymbol{x}, \boldsymbol{y}_j), \tag{6}$$

ここで，$\boldsymbol{\gamma}(\boldsymbol{x},\boldsymbol{y})=\boldsymbol{x}\times\boldsymbol{y}$ および $\mathcal{D}(\boldsymbol{x},\boldsymbol{y})=(R^2+\sigma^2-\boldsymbol{x}\cdot\boldsymbol{y})^{-1}$ で与えられる．$\boldsymbol{x}$ の near-field の中に入っている点渦からの評価は直接計算して求め，far-field からの寄与は far-field に含まれている矩形 τ の中心 $\boldsymbol{y}_\tau$ の周りの $(\lambda-1)$ 次までのテイラー展開で近似する．すなわち，速度場 $\boldsymbol{u}_N$ は以下のように近似される．

$$\boldsymbol{u}_N^{\lambda}(\boldsymbol{x},t)=-\frac{1}{4\pi R}\sum_{\boldsymbol{y}_j\in N(\boldsymbol{x})}\Gamma_j\boldsymbol{\gamma}(\boldsymbol{x},\boldsymbol{y}_j)\mathcal{D}(\boldsymbol{x},\boldsymbol{y}_j)-\frac{1}{4\pi R}\sum_{\tau\in F(\boldsymbol{x})}\boldsymbol{u}_N^{\lambda,\tau}(\boldsymbol{x},t). \tag{7}$$

ただし，

$$\boldsymbol{u}_N^{\lambda,\tau}(\boldsymbol{x},t)=\sum_{\boldsymbol{y}_j\in\tau}\Gamma_j\boldsymbol{\gamma}(\boldsymbol{x},\boldsymbol{y}_j)\sum_{|\boldsymbol{k}|\le\lambda-1}a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}}. \tag{8}$$

である．ここで，$\mathcal{D}$ の微分係数は以下のように陽的に与えられている．

$$a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)=\frac{1}{\boldsymbol{k}!}D_{\boldsymbol{y}}^{\boldsymbol{k}}\mathcal{D}(\boldsymbol{x},\boldsymbol{y})|_{\boldsymbol{y}=\boldsymbol{y}_\tau}=\frac{|\boldsymbol{k}|!}{\boldsymbol{k}!}\left(R^2+\sigma^2-\boldsymbol{x}\cdot\boldsymbol{y}_\tau\right)^{-|\boldsymbol{k}|-1}\boldsymbol{x}^{\boldsymbol{k}}. \tag{9}$$

ただし，多重指標 $\boldsymbol{k}=(k_1,k_2,k_3)$ 対して，$\boldsymbol{k}!=k_1!k_2!k_3!$, $|\boldsymbol{k}|=k_1+k_2+k_3$ と定義し，$\boldsymbol{y}=(y_1,y_2,y_3)$ に対して，$D_{\boldsymbol{y}}^{\boldsymbol{k}}=\frac{\partial^{|\boldsymbol{k}|}}{\partial y_1^{k_1}\partial y_2^{k_2}\partial y_3^{k_3}}$ かつ $\boldsymbol{y}^{\boldsymbol{k}}=y_1^{k_1}y_2^{k_2}y_3^{k_3}$ となっている．この微分係数の陽的表現 (9) のおかげでテイラー係数の計算時間は極めて高速に行える．また，一般にこのような表示があることがこれまでの高速アルゴリズムの研究 [3, 10, 8, 11] がうまくいった理由の一つであることにも注意しよう．

ここで実装上の注意であるが，こうした陽的表示があるからといって安心して組み込み数学関数である power 関数などを使って，直接計算しようとしてはいけない．Power 関数は極めてコストの高い関数であるため，これだけで高速化効率が劇的に悪くなってしまうことがある．たとえこのアルゴリズムが後で示すように $O(N^2)$ の計算を $O(N\log N)$ で計算するといっても，実際に我々が計算する N のスケールで直接計算より早いということは約束されておらず，これは一重にプログラムの実装に依存していることを知っておいた方がよい．なお，実際の数値計算ではテイラー係数 $a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)=a_{(k_1,k_2,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau)$ は以下の再帰公式でより簡単に計算できる．これだと power 関数は不要となり高速化効率も向上する．

$$\begin{aligned}
a_{(k_1+1,k_2,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau) &= \frac{|\boldsymbol{k}|+1}{k_1+1}\mathcal{D}(\boldsymbol{x},\boldsymbol{y}_\tau)x_1\,a_{(k_1,k_2,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau),\\
a_{(k_1,k_2+1,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau) &= \frac{|\boldsymbol{k}|+1}{k_2+1}\mathcal{D}(\boldsymbol{x},\boldsymbol{y}_\tau)x_2\,a_{(k_1,k_2,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau),\\
a_{(k_1,k_2,k_3+1)}(\boldsymbol{x},\boldsymbol{y}_\tau) &= \frac{|\boldsymbol{k}|+1}{k_3+1}\mathcal{D}(\boldsymbol{x},\boldsymbol{y}_\tau)x_3\,a_{(k_1,k_2,k_3)}(\boldsymbol{x},\boldsymbol{y}_\tau).
\end{aligned}$$

ただし，$a_{(0,0,0)}(\boldsymbol{x},\boldsymbol{y}_\tau)=\mathcal{D}(\boldsymbol{x},\boldsymbol{y}_\tau)$ ととっておく．

さて，$\boldsymbol{y}_j=(y_{j1},y_{j2},y_{j3})$ と書くことにすると，far-field での近似 (8) は以下のように簡単にできる．

$$\boldsymbol{u}_N^{\lambda,\tau}(\boldsymbol{x},t)=\sum_{|\boldsymbol{k}|\le\lambda-1}a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)\left(x_2C_\tau^{\boldsymbol{k}}-x_3B_\tau^{\boldsymbol{k}},x_3A_\tau^{\boldsymbol{k}}-x_1C_\tau^{\boldsymbol{k}},x_1B_\tau^{\boldsymbol{k}}-x_2A_\tau^{\boldsymbol{k}}\right). \tag{10}$$

ここで，矩形 τ に付随する係数 $A_\tau^{\boldsymbol{k}}$, $B_\tau^{\boldsymbol{k}}$, $C_\tau^{\boldsymbol{k}}$ は以下で与えられるものである．

$$A_\tau^{\boldsymbol{k}}=\sum_{\boldsymbol{y}_j\in\tau}\Gamma_jy_{j1}(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}},B_\tau^{\boldsymbol{k}}=\sum_{\boldsymbol{y}_j\in\tau}\Gamma_jy_{j2}(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}},C_\tau^{\boldsymbol{k}}=\sum_{\boldsymbol{y}_j\in\tau}\Gamma_jy_{j3}(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}}. \tag{11}$$

これらの係数は矩形τに含まれているすべての点渦$\boldsymbol{y}_j$の情報を含んでいるが，これは点渦の分布が決まれば自動的に計算できるため，それらの計算は一回の速度場の評価において，一度だけ行えばよい．このことがアルゴリズムの高速化を生み出す源であることを注意しておく．この部分でも実数のべき乗の計算$(\boldsymbol{y}_j - \boldsymbol{y}_\tau)^{\boldsymbol{k}}$が含まれているために，その実際の計算でも再帰計算を使うべきである．

2.3 高速アルゴリズム

以上の説明に基づいて，高速tree-codeアルゴリズムをここに示す．その前に二つの再帰アルゴリズムを与えておく．まず最初のアルゴリズムでは，位置$\boldsymbol{y}$にある強さΓを持つ点渦を含むΣ内のすべての矩形τに対して係数$A_\tau^{\boldsymbol{k}}$, $B_\tau^{\boldsymbol{k}}$, $C_\tau^{\boldsymbol{k}}$を計算する再帰アルゴリズムである．

Algorithm 2. 係数$A_\tau^{\boldsymbol{k}}$, $B_\tau^{\boldsymbol{k}}$, $C_\tau^{\boldsymbol{k}}$の計算

```
ComputeNodeCoefficients( τ, k, y, Γ )
    if τ = ∅ return;
    if y ∈ τ then for all k, |k| ≤ λ − 1
        add Γ y1 (y − y_τ)^k to A_τ^k;
        add Γ y2 (y − y_τ)^k to B_τ^k;
        add Γ y3 (y − y_τ)^k to C_τ^k;
        if k = 3l then
            add y to the list of the near-field, L(τ);
            return;
        else
            Recursively call ComputeNodeCoefficients( τ_i, k+1, y, Γ ) for all the chil-
            dren of τ, τ1 and τ2;
            return;
        end
    end
end
```

End of Algorithm

次のアルゴリズムは，計算領域τとtree構造の階層のレベルk，$\boldsymbol{y}$にある点渦が与えられた時に，$\boldsymbol{y}$の上での速度場を再帰的に計算するものである．

Algorithm 3. *(点$\boldsymbol{y}$における速度場の計算)*

```
ComputeFarNearField( τ, k, y )
    if τ = ∅ return;
    if ρ(τ) < h^ν |R^2 − y · y_τ| then
        Compute the far-field approximation according to (10);
        return;
    else
        if k = 3l then
```

```
            Compute the contribution from the points in the near-field list L(τ) di-
            rectly;
            return;
        else
            Recursively call ComputeFarNearField( τ_i, k+1, y ) for all the children of
            τ, τ_1 and τ_2;
            return;
        end
    end
end
```

End of Algorithm

上の二つの再帰アルゴリズムを使えば高速 tree-code は以下のように簡単に書き下すことができる．入力は点渦の数 n, テイラー近似の次数 λ, far-field の条件を定める実数 ν, 点渦の強さ Γ_j と位置 $\boldsymbol{y}_j$ $(j=1,\ldots,n)$ である．出力はすべての点渦 $\boldsymbol{y}_j$ $(j=1,\ldots,n)$ の上における速度場 $\boldsymbol{u}_h^\lambda(\boldsymbol{x},t)$ である．

Algorithm 4. *(球面の点渦相互作用に対する高速 tree-code アルゴリズム)*

```
Stage 0 (計算領域 B に tree 構造を入れる．これは一回だけ呼び出せばよい)
    GenerateMesh( B, 0 );
end
Stage 1 (係数 A_τ^k, B_τ^k, C_τ^k の計算)
    For j = 1,...,n, call ComputeNodeCoefficients( B, 0, y_j, Γ_j );
end
Stage 2 (速度場 (6) の計算)
    For j = 1,...,n, call ComputeFarNearField( B, 0, y_j );
end
```

End of Algorithm

なお，これらのアルゴリズムは far-field の判定条件とテイラー展開による近似から生成される係数 A_τ^k, B_τ^k, C_τ^k の定義が異なるだけで，三次元点渦相互作用に対する高速 tree-code アルゴリズムと本質的に同じである．そのため計算回数は論文 [3] と同様に $O(N\lambda^3\log N)$ となっている．特にテイラー近似の次数を $\lambda = O(\log N)$ のように選んでおけば $O(N(\log N)^4)$ となっている．

2.4 誤差解析

高速アルゴリズムの近似誤差について述べる．ここでは $\sigma = 0$ の場合について調べるが $\sigma \neq 0$ の時も同様にできる．まず定数 C を以下のように定義する．

$$C = \max_{1\le j\le N} \frac{m(\mathcal{A})}{4\pi R}\omega_0(\theta_j,\phi_j) = N \max_{1\le j\le N} \frac{\Gamma_j}{4\pi R}.$$

この時，誤差は以下のようになる．

$$\begin{aligned}\left|\boldsymbol{u}_N(\boldsymbol{x},t)-\boldsymbol{u}_N^{\lambda}(\boldsymbol{x},t)\right| &= \left|\frac{1}{4\pi R}\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}\Gamma_j\gamma(\boldsymbol{x},\boldsymbol{y}_j)\sum_{|\boldsymbol{k}|=\lambda}a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}}\right| \\ &\le \frac{C}{N}\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}|\gamma|\sum_{|\boldsymbol{k}|=\lambda}|a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)||(\boldsymbol{y}_j-\boldsymbol{y}_\tau)^{\boldsymbol{k}}|. \qquad (12)\end{aligned}$$

ここで，$|\boldsymbol{x}|=|\boldsymbol{y}_j|=R$ より $|\gamma(\boldsymbol{x},\boldsymbol{y}_j)|=|\boldsymbol{x}\times\boldsymbol{y}_j|\le R^2$ であり，$\boldsymbol{y}_j\in\tau$ に対して，$|\boldsymbol{y}_j-\boldsymbol{y}_\tau|\le\rho(\tau)$ であること，さらには $|\boldsymbol{k}|=\lambda$ に対するテイラー係数 $a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)$ が以下のように評価されることに注意する．

$$|a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)|\le\frac{\lambda!}{\boldsymbol{k}!}|R^2-\boldsymbol{x}\cdot\boldsymbol{y}_\tau|^{-\lambda-1}R^{\lambda+1}\le\frac{\lambda!}{\boldsymbol{k}!}\rho^{-\lambda-1}(\tau)h^{\nu(\lambda+1)}R^{\lambda+1}.$$

また，

$$(a+b+c)^{\lambda}=\sum_{|\boldsymbol{k}|=\lambda}\frac{\lambda!}{\boldsymbol{k}!}a^{k_1}b^{k_2}c^{k_3}$$

なので，$\sum_{|\boldsymbol{k}|=\lambda}\lambda!/\boldsymbol{k}!=3^{\lambda}$ となる．加えて任意の τ に対して $\rho(\tau)\ge\frac{\sqrt{3}}{2}h$ かつ $\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}=\sum_{\tau\in F(\boldsymbol{x})}n_\tau\le N$，（$n_\tau$ は矩形領域 $\tau\in F(\boldsymbol{x})$ に含まれる点渦の数）であることなどをふまえて，以下を得る．

$$\begin{aligned}\left|\boldsymbol{u}_N(\boldsymbol{x},t)-\boldsymbol{u}_N^{\lambda}(\boldsymbol{x},t)\right| &\le \frac{CR^2}{N}\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}\sum_{|\boldsymbol{k}|=\lambda}|a_{\boldsymbol{k}}(\boldsymbol{x},\boldsymbol{y}_\tau)|\rho^{\lambda}(\tau) \\ &\le \frac{CR^{\lambda+3}}{N}\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}\sum_{|\boldsymbol{k}|=\lambda}\frac{\lambda!}{\boldsymbol{k}!}\rho^{-1}(\tau)h^{\nu(\lambda+1)} \\ &\le \frac{CR^{\lambda+3}}{N}3^{\lambda}\frac{2}{\sqrt{3}}h^{-1}\sum_{\tau\in F(\boldsymbol{x})}\sum_{\boldsymbol{y}_j\in\tau}h^{\nu(\lambda+1)} \\ &\le C'R^{\lambda+3}3^{\lambda}h^{\nu(\lambda+1)-1}\le C''h^{\nu\lambda-1}. \qquad (13)\end{aligned}$$

ここで，C' や C'' は適当な定数である．したがって，$\nu=O(3/\lambda)$ 程度に選んでおけば，近似誤差は

$$\left|\boldsymbol{u}_N(\boldsymbol{x},t)-\boldsymbol{u}_N^{\lambda}(\boldsymbol{x},t)\right|\le C''h^2. \qquad (14)$$

のようになる．この誤差は二次元点渦相互作用の高速 tree-code アルゴリズム [3] と同じである．

3 アルゴリズムのテスト結果

前セクションの解析結果を確かめるために，以下のようなテスト問題を用意する．正則化パラメータを $\sigma=0.05$ とし，速度場の評価を実際に計算してみる．まず半径が $R=0.5$ の

球面上の M 本の緯線の上に均等配分された，単位強さ持つ点渦を考える．この時，その配置は次のように与えられる．今，i 番目の緯度の z 座標を

$$z^{(i)} = R - \frac{i}{M+1}, \quad i = 1, \ldots, M,$$

として，点渦の座標は次のように与えられる．

$$\boldsymbol{x}_j^{(i)} = \left(\sqrt{R^2 - (z^{(i)})^2} \cos 2\pi j/N, \sqrt{R^2 - (z^{(i)})^2} \sin 2\pi j/N, z^{(i)} \right), \quad j = 1, \ldots, N'. \quad (15)$$

全部で点渦の数は $N = MN'$ 個あることに注意する．この配置は球面全体に点が分布しているので高速アルゴリズムのテストとしては最適なものである．

Far-field を決定する条件 (4) に現れるパラメータ ν, 計算領域に導入する階層メッシュ構造の深さ l などの高速アルゴリズムに必要なパラメータの大きさは $\nu = \frac{1}{n}$, $l = n$ となるように選んでおく．ただし, n は $N = 2^n$. で決まる値である．数値計算はオプテロン 275 プロセッサで実行している. テイラー展開の近似次数 λ は 4 から 10 まで変更するものとし，計算された結果の誤差を見るために，与えられた点渦の位置を $\boldsymbol{x}_i$ $(i = 1, \ldots, N)$ としたときに，直接算法で計算された速度場 $\boldsymbol{u}_N^{\lambda}(\boldsymbol{x}_i)$ と高速アルゴリズムで計算された速度場 $\boldsymbol{u}_N(\boldsymbol{x}_i)$ に対する L^2 相対誤差 $E_N^{(2)}$ と最大相対誤差 $E_N^{(\infty)}$ を以下のように与える．

$$E_N^{(2)} = \frac{\left(\sum_{i=1}^{N} \left| \boldsymbol{u}_N(\boldsymbol{x}_i) - \boldsymbol{u}_N^{\lambda}(\boldsymbol{x}_i) \right|^2 \right)^{\frac{1}{2}}}{\left(\sum_{i=1}^{N} \left| \boldsymbol{u}_N(\boldsymbol{x}_i) \right|^2 \right)^{\frac{1}{2}}}, \qquad E_N^{(\infty)} = \frac{\max\limits_{1 \le i \le N} \left| \boldsymbol{u}_N(\boldsymbol{x}_i) - \boldsymbol{u}_N^{\lambda}(\boldsymbol{x}_i) \right|}{\max\limits_{1 \le i \le N} \left| \boldsymbol{u}_N(\boldsymbol{x}_i) \right|}.$$

表 1 に様々な N と λ に対する，L^2 相対誤差と最大値誤差を与えている．それによると，両相対誤差は N を固定し，近似次数 λ を大きくすると近似の桁があがっているという顕著な減少を見せている．一方で，λ を固定した場合は，N をいくら大きくしてもその誤差のオーダーはほとんど変わらない．

表 2 には直接算法と高速アルゴリズムに対する速度場の計算時間を示している．直接算法の計算時間は $O(N^2)$ のオーダーで計算時間が増加しているのに対して，高速アルゴリズムに対しては緩やかな増加が見られる．といっても，N が小さい時では，高速アルゴリズムと直接算法の計算時間は同じ程度か，あるいはテイラー展開の近似次数によっては遅くなっているので，高速アルゴリズムが実時間で実用的になるのは N が大きいときであることに注意する．この実時間で高速アルゴリズムの効果が期待できる N の大きさは，高速アルゴリズムの実装の方法に大きく依存する．前に説明したように組み込み power 関数などを使った場合は N が 16384 点であっても高速算法より直接算法の方が早い時間で計算を完了してしまう．このアルゴリズムの高速化効率について，二次元点渦法に対する高速 tree-code アルゴリズム [10] の結果と比較してみる．二次元の場合は $N = 65536$, $\lambda = 8$, $\nu = 0.05$ に対して，近似誤差 3.7×10^{-5}，その高速化は直接計算の約 50 倍を達成している．一方，今回の球面の高速 tree-code では $N = 65536$, $\lambda = 6$, $\nu = 0.00625$ のパラメータ設定に対して誤差が 4.1×10^{-5} であり，その高速化効率は 2.94 にとどまっている．したがって，球面版の高速 tree-code はその高速化効率において二次元版のアルゴリズムほどではないことがわかるが，これは球面の方程式を三次元空間の中で定式化して，それに対してテイラー展開

(a) N	$\lambda=4$	$\lambda=6$	$\lambda=8$	$\lambda=10$
4096	3.25e-4	1.84e-5	1.18e-6	8.50e-8
16384	4.31e-4	2.37e-5	1.49e-6	1.07e-8
65536	4.51e-4	2.50e-5	1.58e-6	1.15e-7
262144	4.74e-4	2.65e-5	1.72e-6	1.27e-7
1048576	4.89e-4	2.74e-5	1.80e-6	1.36e-7
(b) N	$\lambda=4$	$\lambda=6$	$\lambda=8$	$\lambda=10$
4096	4.23e-4	3.32e-5	1.35e-6	8.35e-8
16384	6.91e-4	3.60e-5	2.09e-6	1.35e-8
65536	7.78e-4	4.07e-5	2.35e-6	1.48e-7
262144	8.05e-4	4.17e-5	2.41e-6	1.53e-7
1048576	8.17e-4	4.20e-5	2.41e-6	1.53e-7

Table 1: 点渦の配置 (15) に対して直接算法と高速 tree-code アルゴリズムで計算された速度場 (3) に対する (a) L^2 相対誤差 $E_N^{(2)}$ と (b) 最大相対誤差 $E_N^{(\infty)}$.

N	$\lambda=4$	$\lambda=6$	$\lambda=8$	$\lambda=10$	direct	ν
4096	0.7	1.6	3.1	6.5	0.6	$\nu=0.083$
16384	4.9	11.9	20.3	34.7	10.2	$\nu=0.071$
65536	40.3	71.8	98.0	156.2	164.3	$\nu=0.0625$
262144	310.5	433.2	576.5	908.8	2629.3	$\nu=0.055$
1048576	2034.4	2694.5	4018.2	6050.5	42060.5	$\nu=0.05$

Table 2: 点渦の配置 (15) に対する高速 tree-code アルゴリズムと直接算法による速度場の計算にかかった時間（単位は秒）.

の近似を行ったために計算すべきテイラー係数の数は $O(\lambda^3)$ となり，これが二次元の場合 $O(\lambda^2)$ に比べて非常に大きいことに起因している.

最後にこれらの数値計算結果を使って，前セクションで示された誤差解析や計算量のオーダーの見積もりが達成されていることを見る．これらの解析では $\lambda\nu$ の値がだいたい同じであるとの仮定が用いられているので，ここでも表 3 に示されたデータから $\lambda\nu \sim 0.5$ となっているデータを抜き出して図 1 に示す．これによると最大誤差は $O(h^2) \sim O(1/N)$ で収束し，計算時間は $O(N(\log N)^4)$ で増加している．これらは理論的解析を裏付けるものである.

4 まとめ

本報告では，論文 [13] に基づいて，球面上に N 個の点渦が与えられた時にその点における他の点渦からの誘導速度を $O(N\log N)$ で近似する高速 tree-code アルゴリズムの紹介，その誤差解析とその数値テスト結果を与えた．基本的なアイデアは球面上の方程式を三次元座標系で定式化し，それに対して三次元の高速 tree-code アルゴリズムを構成するという方法で行うところにある．数値計算の実装によって，その高速化が実感できる N の大きさは

N	4096	65536	1048576
λ	6	8	10
ν	0.083	0.0625	0.05
$E_N^{(\infty)}$	3.32e-5	2.35e-6	1.53e-7
Time (s)	1.6	98.0	6050.5

Table 3: 最大相対誤差と計算時間．$\lambda\nu$ が一定になるようにして表 1 と表 2 から抜き出したもの．

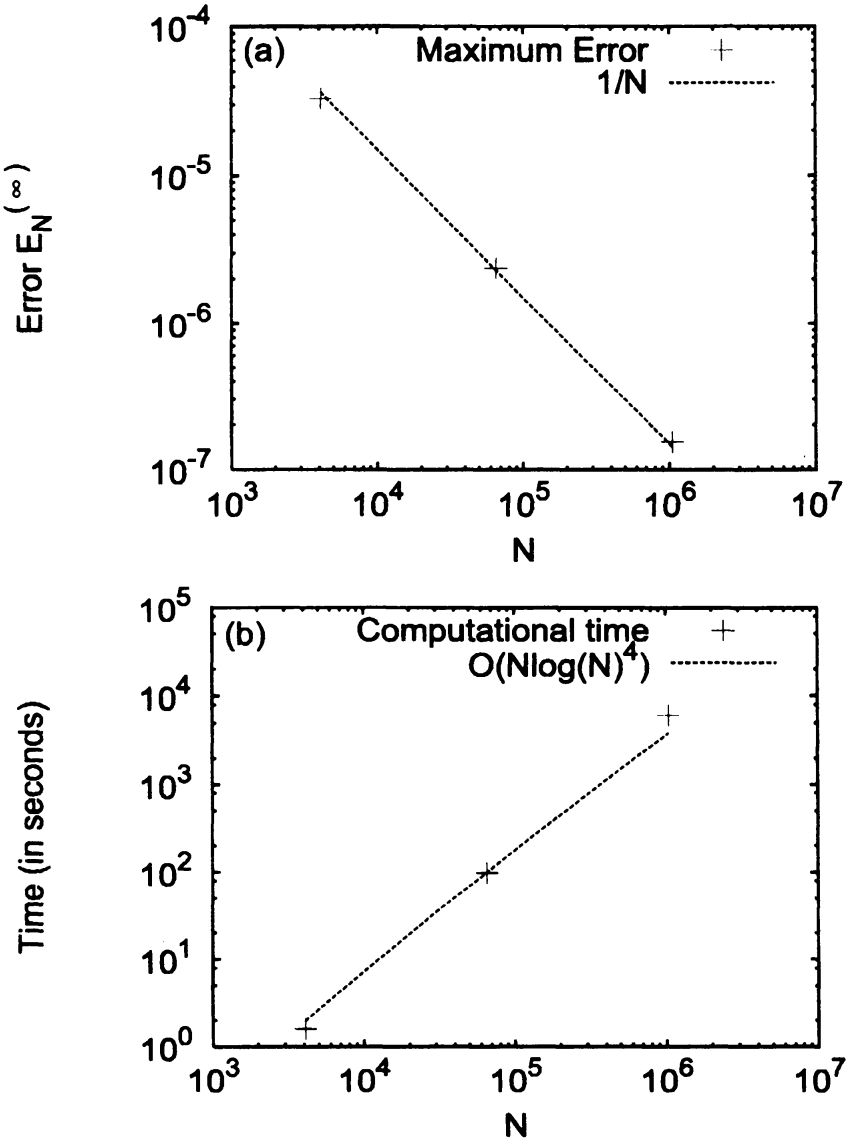

Figure 1: 点渦の数 N に対する，最大相対誤差 $E_N^{(\infty)}$ と計算時間．パラメータの値は表 3 の通り．

異なるが，うまく実装すれば $N = 16384$ 点程度で十分早く，N がもっと大きくなればそれだけ高速化効率があがり，さらに近似誤差もどんどんよくなっていくアルゴリズムとなっている．実際の数値計算では丸め誤差があるので，十分大きな N に対しては直接算法とほとんどかわらない結果を計算してくれるという点でも効果的なアルゴリズムである．また球面スペクトル法に対する高速ルジャンドル展開に比べても高速化効率は高く，今後の利用が期待できるアルゴリズムでもある．さらに，点渦法は点渦間の距離にのみ依存する相互作用だけに注目したラグランジュ的方法なので，オイラー的に球面メッシュを入れて考えた時に見える北極や南極における見かけの特異性は存在しないことも注意しておく必要がある．

さて，我々が今考えている非回転球面上の離散渦法の枠組みであれば，各点渦の運動だけを数値的に追跡すればよいので，各点渦の上での速度場を計算するのに高速アルゴリズムを使ったが，回転する球面上での渦運動を考える時は，渦度はもはや流体粒子の軌道に沿って保存される量でないために，このアルゴリズムはそのままでは適用ができない．回転球面では渦度に流体粒子の位置情報を加えた「ポテンシャル渦度」と呼ばれる量が流体粒子の軌道に沿って保存されているので，もし初期に渦度がない領域でも，ほんの少し時間が経過すると，流体粒子が球面での位置を変えるとそれに応じて渦度が新たに生成してしまうことになる．したがって，初期渦度存在領域だけに点渦を配置するという離散渦法は使えないし，その結果この高速アルゴリズムを使うことも難しそうに見える．しかし一方で，この高速算法は与えられた N 点の渦点に対して，任意の球面上の N 点 $\boldsymbol{x}_i \in S$ $(i = 1, \ldots, N)$ における速度場を同様に計算してくれるアルゴリズムであると見なせば，Semi-Lagrange 的アイデアで回転球面上における高速アルゴリズムとして機能する可能性がある．実際，球面を適当にメッシュを入れて，その上に点渦を置き，高速算法で速度場を計算，それに従って点渦を少し動かした後に，その点渦のもつ循環の大きさをポテンシャル渦度保存則から決定し，その配置に対してメッシュの上で速度場を高速算法で計算すれば，メッシュの上に（新しい循環の大きさを持った）点渦を構成できる．このステップを繰り返すことで球面メッシュ上における渦度の時間発展を近似することが可能になる．このような方法がきちんと回転球面のオイラー方程式の解を近似するかなどについては，まだ未知の部分がさらなる研究が必要である．

謝辞

本研究は日本学術振興会科研費，若手研究 (A) #17684002 2007 の支援を受けて行われた研究である．また本論文作成時に，著者は科学技術振興機構さきがけからの支援を受けている．さらに，本報告は英国シェフィールド大学に滞在中に大木谷耕司教授の支援，および同大学応用数学科から提供された快適な研究環境のもとで作成された．ここに感謝の意を表する．

References

[1] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm", Nature, vol. 324, pp. 446–449, 1986

[2] G.-H. Cottet and P. D. Koumoutsakos, "Vortex methods, theory and practice", Cambridge Univ. Press, 1994

[3] C. I. Draghicescu, "An efficient implementation of particle methods for the incompressible Euler equations", SIAM J. Numer. Anal., vol. 31 No. 4, pp. 1090-1108, 1994

[4] C. I. Draghicescu and M. Draghicescu, "A fast algorithm for vortex blob interactions", J. Comput. Phys., vol. 116, pp. 69–78, 1995

[5] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations", J. Comput. Phys., vol. 73, pp. 325–348, 1987

[6] L. Greengard and V. Rokhlin, "Rapid evaluations of potential fields in three dimensions", Springer Lecture Notes in Mathematics, vol. 1360, Springer, Berlin, pp. 121–141, 1988

[7] R. Krasny, "Desingularization of periodic vortex sheet roll-up", J. Comput. Phys, vol. 65, pp. 292–313, 1986.

[8] K. Lindsay and R. Krasny, "A particle method and adaptive treecode for vortex motion in three-dimensional flow", J. Comput. Phys., vol. 172, pp. 879–907, 2001

[9] P. K. Newton, "The N-vortex problem, analytical techniques", Springer-Verlag, 2001

[10] T. Sakajo and H. Okamoto, "An application of Draghicescu's fast summation method to vortex sheet motion", J. Phys. Soc. Japan, vol 67, No. 2, pp.462–470, 1998

[11] T. Sakajo, "Numerical computation of a three-dimensional vortex sheet in a swirl flow", Fluid Dyn. Res., vol. 28, pp.423–448, 2001

[12] T. Sakajo, "Motion of a vortex sheet on a sphere with pole vortices", Phys. Fluids, vol. 16, pp. 717–727, 2004

[13] T. Sakajo, "A fast tree-code algorithm for the vortex method on a sphere", J. Comp. Appl. Math. 2008, doi: 10.1016/j.cam.2008.07.021