ハイパフォーマンス
コンピューティング 83－4
（2000．10．6）

高速球面調和関数変換法の精度と速度

須田礼仁、高見雅保
名古屋大学大学院 工学研究科 計算理工学専攻

概要

球面調和関数変換は通常、切断波数 M に対して計算量が $O(M^3)$ となる直接法で計算されている。これに対し、我々は計算量が $O(M^2 \log M)$ となる高速変換アルゴリズムを提案してきた。今回、我々のアルゴリズムをはじめて実装・評価したので報告する。まず、未解決の課題であった split Legendre function を用いた場合の内挿計算を安定にするための標本点アルゴリズムを提案する。また、このアルゴリズムを用いた我々の実装について報告し、その精度と速度について報告する。我々の実装では、計算は十分に安定で、速度も $M = 511$ で従来の方法よりも高速になったことが確認された。

Performance of Fast Spherical Harmonics Transform

Reiji Suda and Masayasu Takami
Department of Computational Science and Engineering, Nagoya University

Abstract

Spherical harmonics transform is usually computed directly in time $O(M^3)$ for cut-off frequency M. We have proposed a fast transform algorithm which runs in time $O(M^2 \log M)$ accelerated by FMM (Fast Multipole Method) through polynomial interpolation. In this paper, we report the first implementation of our algorithm. We propose an algorithm to select sampling points for dual vectors, which is essential for stable numerical computations. The precision and the speed of our implementation are also reported. The results show high stability, and our algorithm becomes faster than the conventional method for $M = 511$.

1 はじめに

球面調和関数変換は球面上のフーリエ変換に相当するもので、惑星や太陽に関する流体・電磁気現象の数値シミュレーションや、球面上での画像・信号処理に使用される。球面調和関数変換はフーリエ変換とルジャンドル陪関数変換とに分解される。フーリエ変換の方は FFT（高速フーリエ変換）により効率的に計算ができるが、ルジャンドル陪関数変換については従来は直接法が用いられていた。直接法によると、切断波数 M に対して球面調和関数変換の計算量は $O(M^3)$ となり、多くの大規模・高精度アプリケーションでこの計算量が問題となっていた。

これに対して我々は計算量が $O(M^2 \log M)$ となる高速変換アルゴリズムを提案した。アルゴリズムのアイデアについては 1998 年 [1] に、標本点の選択アルゴリズムと、それを用いた分割統治法を伴わない簡易実装の結果については 1999 年 [2] に発表した。また、我々のアルゴリズムの高速性の源になっている FMM（高速多重極子展開法）における領域分割の工夫とそれによる性能の改善については [3] で詳しく説明した。しかし、これらの報告では split Legendre function が実装されていないために分割統治法が使われておらず、計算量の係数は約半分になるものの、オーダーは $O(M^3)$ であった。

今回我々は split Legendre function を用いた dual vector による変換計算を実装し、分割統治法を適用して $O(M^2 \log M)$ の計算量となる高速変換を実現した。本稿では、split Legendre function の導入に伴い必要となる dual vector の内挿計算のための標本点選択アルゴリズムを提案し、高速ルジャンドル陪関数変換アルゴリズムの精度と速度について報告する。なお、関連研究の概要については紙面の都合で省略するので、参考文献を参照されたい。

2 高速変換アルゴリズムのアイデア

この節では、我々の高速変換アルゴリズムのアイデアについて説明する。アルゴリズムの詳細については参考文献に挙げた我々の論文を見られたい。

計算をしたいのはルジャンドル陪関数逆変換

$$s^m(\mu) = \sum_{n=m}^{M} g_n^m P_n^m(\mu)$$

である。順変換はガウス積分と逆変換の転置で実現できるので、逆変換だけを考えれば十分である。ルジャンドル陪関数は

$$P_n^m(\mu) = P_m^m(\mu) q_n^m(\mu)$$

のように、関数 $P_m^m(\mu)$ と多項式との積である。従って、$P_n^m(\mu)$ を $P_m^m(\mu)$ で割ってやれば多項式の計算に帰着させることができる。

しかし、単純にルジャンドル陪関数変換の P_n^m を q_n^m で置き換えてやると、数値的に不安定になりやすい。この不安定性を抑えるためには、計算途中の中間結果の表現に工夫が必要である。我々のアイデアの第 1 は、評価点集合 $\mathcal{M} = \{\mu_j\}$ の中からいくつかの点を選んで「標本点」とし、その上での関数値をベクトル状に並べたものを中間表現として用いることである。

標本点以外の評価点における関数値は、中間表現に含まれる関数値を内挿することによって得られるが、関数値を P_m^m で一旦割ってやることにより、よく知られた多項式の内挿公式を計算に用いることができる。この多項式内挿に FMM（高速多重極子展開法）を用いることにより、N 個の標本点での関数値から M 個の内挿点での関数値を求める内挿の計算量が $O(N+M)$ になる。これを利用するのが我々のアイデアの第 2 である。

内挿によって関数値を得るために必要となる標本点の数は、多項式の次数によって決まる。ルジャンドル陪関数変換はいくつかのルジャンドル陪関数の線形結合を評価することに他ならないので、M 個の関数の線形結合を表現するためには M 個の点で関数値がわかっていればよい。この M 個の関数値を求めるために、線形結合を $M/2$ 個ずつの 2 つの部分線形結合に分割し、それぞれ $M/2$ 個の点での関数値を求めておいて、それぞれの残りの点での関数値を内挿によって求めることができる。このとき FMM を用いれば内挿の計算量は $O(M)$ となる。これを再帰的に繰り返してやると、$O(\log M)$ 段で $O(1)$ の計算量の計算に帰着されるから、$O(M \log M)$ でルジャンドル陪関数変換ができることになる。すべての位数 m について計算すると合計の計算量が $O(M^2 \log M)$ となる。このように分割統治法を用いるのが我々のアイデアの第 3 である。

しかし、分割統治法によって生じる部分線形結合は P_m^m で割っても高い次数の多項式になってしまい、内挿の高速性が失われてしまう。ここで、ルジャンドル陪関数を次のように 2 つの **split Legendre function** の和に分離することができることに注目する。

$$P_n^m(x) = P_{n,\nu}^{m,0}(x) + P_{n,\nu}^{m,1}(x)$$

ただし、split Legendre function $P_{n,\nu}^{m,l}(x)$ は

$$P_{n,\nu}^{m,l}(x) = q_{n,\nu}^{m,l}(x) P_{\nu+l}^m(x)$$

のように分解できる。ここで $q_{n,\nu}^{m,l}(x)$ は $n-\nu+l-1=0$ の時は 0、それ以外のときには次数が $|n-\nu+l-1|-1$ の多項式である。この分離はルジャンドル陪関数の漸化式から簡単に導かれるが、これを用いて部分線形結合を

$$\begin{aligned} s_\nu(\mu) &= s_\nu^0(\mu) + s_\nu^1(\mu) \\ s_\nu^l(\mu) &= \sum_n g_n P_{n,\nu}^l(\mu) \end{aligned}$$

と分離してやると、分離された部分線形結合 $s_\nu^l(\mu)$ はそれぞれ $P_\nu(\mu)$ で割ってやれば低い次数の多項式になる。このようにして FMM による内挿の高速性を確保するのが、我々のアイデアの第 4 である。以下、分離された部分線形結合の標本点上の値を並べたものを **dual vector** と呼ぶことにする。

このようにして、分割統治法において split Legendre function を導入して FMM による高速多項式内挿法を利用できるようにすることにより、ルジャンドル陪関数変換の高速変換アルゴリズムを構成することができる。このアルゴリズムの数値的な安定性について調べてみると、内挿計算の数値的な安定性がもっとも重要であることがわかる。この内挿計算の数値的な安定性を、標本点を適切に選択してやることにより実現するというのが我々のアイデアの第 5 である。Split Legendre function を用いていない場合 (**single vector**) の標本点の選択の方法については、[2] ですでに報告しているが、次の節では dual vector の場合の標本点選択について論じる。

3 標本点選択アルゴリズム

Dual vector の内挿計算は

$$\begin{aligned} s_\nu^l(y_i) &= \sum_j \Theta_{ij}^l s_\nu^l(x_j) \\ \Theta_{ij}^l &= \frac{P_{\nu+l}^m(y_i)\omega_j(y_i)}{P_{\nu+l}^m(x_j)\omega_j(x_j)} \\ \omega_j(x) &= \prod_{k\neq j}(x-x_k) \end{aligned} \tag{1}$$

という形になる。ここで式 (1) の左辺 $s_\nu^l(y_i)$ は決まった関数であるから、その値は標本点 x_i の選択により変化することはありえない。従って、もし Θ_{ij}^l が大きな数の場合には、左辺の値を不変にするために計算中に大きい値同士が相殺されなければならない。これは桁落ち、すなわち精度の低下を意味する。従って、数値的な安定性を実現するためには Θ_{ij}^l はできるだけ小さな値に抑えておくべきである。

この Θ_{ij}^l は、標本点集合 $\{x_j\}$ と内挿点 y_i で決まる。我々の問題の場合、最終的には与えられた評価点集合 $\mathcal{M}=\{\mu_i\}$ の各点で関数を評価したい。このため、内挿点集合 $\{y_i\}$ は評価点集合から標本点集合を取り除いた差集合 $\mathcal{M}-\{x_j\}$ でなければならない。このことから、計算量を最小にするには、標本点集合を評価点集合の部分集合とすればよいことがわかる。従って、標本点の数が N 個必要な場合、評価点集合 $\mathcal{M}$ のすべての N 元からなる部分集合を考えて、それぞれを標本点集合とした場合の係数 Θ_{ij}^l を計算して、結果が最適なものを選ぶということをすれば最良の標本点集合が得られる。しかし、このような組み合わせを数え上げる方法では計算量がかかりすぎて実用にならない。もっと簡単かつ効果的なアルゴリズムが必要である。

我々の以前の論文 [2] では、split をしていない single vector の場合について、次のような標本点選択アルゴリズムを提案した。入力は評価点集合 $\mathcal{M}$ と標本点の個数 N で、出力は標本点集合 $\{x_j\}_{j=1}^N$ である。

1. 初期化：$\mu\in\mathcal{M}$ に対して $w_1(\mu)=P_m^m(\mu)$ とする。
2. $j=1$ とする。
3. $|w_j(\mu)|$ を最大にする $\mu\in\mathcal{M}$ を j 番目の標本点 x_j とする。
4. $\mu\in\mathcal{M}$ に対し $w_{j+1}(\mu)=(\mu-x_j)w_j(\mu)$ とする。
5. $j<N$ なら j を 1 増やしてステップ 3 に戻る。

ここで $w_j(\mu)$ は $P_m^m(\mu)\omega(\mu)$ (但し $\omega(x)=\prod(x-x_j)$) を最初の $j-1$ 個の標本点について部分的に評価したものと考えることができる。これを Θ_{ij} の近似値とみなして、できるだけ小さくするような点を新しい標本点として選ぶのである。このアルゴリズムはかなり粗い近似と単純な貪欲最適化を行っているのであるが、内挿の結果は [2] で報告してあるように極めて安定である。

これを dual vector に適用する場合には、Θ^0_{ij} と Θ^1_{ij} を同時に小さく抑えなければならない。分離した 2 つの線形結合 ($l = 0, 1$) に対してそれぞれ標本点集合を選び、2 つの標本点集合の和集合を標本点集合とすれば上記のアルゴリズムがそのまま使えるが、標本点の数が増えてしまうため計算量の点で不利である。Θ^0 と Θ^1 を同時に最小にすることはできるとは限らないので、ふさわしい目標は両方合わせたすべての要素の中で絶対値がもっとも大きいもの（つまり最悪のもの）をできるだけ小さくするという、minimax 条件になる。そこで我々はこの考えに基づいて以下のような標本点選択アルゴリズムを考えた。

1. 初期化：$\mu \in \mathcal{M}$ に対し、$w^0_1(\mu) = P^m_\nu(\mu)$ および $w^1_1(\mu) = P^m_{\nu+1}(\mu)$ とする。
2. $j = 1$ とする。
3. j 番目の標本点 $x_j \in \mathcal{M}$ は、

$$\min\left\{\frac{|w^0_j(x_j)|}{\max\{|w^0_j(\mu)|\}}, \frac{|w^1_j(x_j)|}{\max\{|w^1_j(\mu)|\}}\right\}$$

   を最大にするものに選ぶ。
4. $\mu \in \mathcal{M}$ に対し、$w^0_{j+1}(\mu) = (\mu - x_j)w^0_j(\mu)$ および $w^1_{j+1}(\mu) = (\mu - x_j)w^1_j(\mu)$ とする。
5. $j < N$ ならば j を 1 増やしてステップ 3 に戻る。

このアルゴリズムは single vector の場合のアルゴリズムを素直に拡張したものである。このアルゴリズムでは 2 つの内挿を同時に安定化させようとしているので、必ずしも十分な安定性が確保できるとは限らない。しかし次節で報告するように、このアルゴリズムを用いて dual vector の内挿を含む高速変換を実装してみたところ、数値的な安定性は十分であった。

4 実装と評価

この節では高速変換アルゴリズムの実装の結果について報告する。実装に用いたプログラミング言語は C で、コンパイラと最適化オプションは gcc -02 である。このプログラムでは FMM の展開次数 (K) はマクロ定数として定義されている。実行は SUN Enterprise 450 (300 MHz) を用いた。4 CPU あるが、メモリアクセスの負担を軽くするために 1 CPU だけで実行している。実装はかなり原始的なもので、直接法・高速法ともに性能のチューニングをしていない。このため絶対性能としてはかなり低いものである。

ルジャンドル陪関数 $P^m_n(x)$ は $n \leq 2m$ で $x \approx 1$ において急激に 0 に接近し、次数が大きいとアンダーフローを起こしてしまう。このとき漸化式で関数値が計算できなくなるので、これを避けるために指数部を拡張したデータ構造を導入した。この拡張指数部表現のために前処理に必要な時間が相当長くなってしまっているが、実際の変換には通常の浮動小数点数を用いており、アルゴリズムの高速性には影響しない。

このように関数値が極端に小さくなる場合、直接法においてはそれに関する変換計算を省略している。この工夫は計算量を減らす効果もあるが、組立除法による直接変換法の計算を安定化させるのがその主目的である。この工夫を伴う直接法を、ここでは「簡略化直接法」と呼ぶことにする。これに対し、すべての計算を実行する仮想的な直接法を「完全な直接法」と呼ぶ。完全な直接法は計算途中でオーバーフローを起こしてしまうために実際には実行できないが、その計算時間は簡略化直接法の所要時間と計算量の比から推定できる。なお、簡略化直接法の計算省略のための閾値パラメタを高速アルゴリズムの精度に合わせて設定しているため、高速アルゴリズムの精度とともに直接法の計算時間もわずかに変化する。簡略化直接法の計算量は、完全な直接法の計算量のおよそ 2/3 となっている。

高速変換法においては、split Legendre function のための分離点 ν を、次数 n の範囲の下端に設定した。これは計算量的には有利ではないが、数値的な安定性がもっとも優れている。また、分割統治法のための分割点は、次数 n の範囲の中央に設定した。これは自然だが、必ずしも計算量的には最適とは限らない。

表 1 は、実装した高速変換アルゴリズムの精度と速度を示したものである。表の左端にある K と M はそれぞれ FMM の展開次数と切断波数である。「誤差」は変換結果の相対誤差であるが、次のように評価し

表 1: 高速アルゴリズムの精度と速度

K: FMM の展開次数、N: 切断波数、誤差: 最大値ノルムによる相対誤差、T_f: 高速アルゴリズムの所要時間、T_d: 簡略化直接法の所要時間、T_0: 完全な直接法の予測所要時間、T_d/T_f: 簡略化直接法に対する高速化率、T_0/T_f: 完全な直接法に対する高速化率。所要時間の単位は秒。

K	M	誤差	T_f	T_d	T_0	T_d/T_f	T_0/T_f
10	341	5.13E-06	0.832	0.840	1.143	1.01	1.37
14	341	1.59E-08	0.848	0.881	1.168	1.04	1.38
18	341	1.59E-10	0.862	0.887	1.166	1.03	1.35
22	341	2.74E-13	0.896	0.875	1.133	0.98	1.27
10	511	6.14E-06	2.506	2.843	3.968	1.13	1.58
14	511	1.52E-08	2.734	3.004	4.079	1.10	1.49
18	511	8.61E-11	2.861	3.037	4.076	1.06	1.42
22	511	4.48E-13	3.052	3.035	4.061	0.99	1.33
10	682	6.53E-06	5.379	7.098	9.889	1.32	1.84
14	682	7.17E-08	6.198	7.050	9.796	1.14	1.58
18	682	1.03E-09	6.489	7.660	10.389	1.18	1.60
22	682	1.91E-12	6.477	6.928	9.449	1.07	1.46
10	1023	1.50E-05	16.063	25.033	35.329	1.56	2.20
14	1023	1.47E-07	18.440	26.637	36.883	1.44	2.00
18	1023	1.57E-09	19.661	25.941	35.863	1.32	1.82
22	1023	1.36E-11	21.583	26.827	36.499	1.24	1.69
10	1365	8.89E-06	33.921	62.418	88.677	1.84	2.61
14	1365	7.29E-08	38.810	64.743	91.548	1.67	2.36
18	1365	1.01E-09	43.691	63.689	88.830	1.46	2.03
22	1365	1.59E-11	47.668	63.979	88.343	1.34	1.85
10	2047	3.37E-05	98.062	233.450	333.790	2.38	3.40
14	2047	8.38E-07	115.169	235.493	333.215	2.04	2.89
18	2047	1.25E-09	124.836	232.700	325.763	1.86	2.61
22	2047	2.54E-11	139.115	237.069	333.282	1.70	2.40
10	2730	1.24E-05	202.077	567.975	816.783	2.81	4.04
14	2730	1.48E-07	242.160	566.347	808.492	2.34	3.34
18	2730	1.60E-09	271.347	573.762	817.067	2.11	3.01
22	2730	2.57E-11	296.666	587.312	825.074	1.98	2.78
10	4095	2.92E-05	567.141	2078.464	2983.363	3.66	5.26
14	4095	1.75E-07	675.425	2131.294	3051.445	3.16	4.52
18	4095	2.25E-09	774.313	2231.551	3146.774	2.88	4.06
22	4095	7.39E-11	869.345	2069.444	2957.820	2.38	3.40

た。各要素が [0, 1) の一様分布をするランダムな入力ベクトルを作成し、直接法と高速変換法で逆変換を行い、出力ベクトルの相対誤差を最大値ノルムで評価する。これを 10 回繰り返し、相対誤差の最大値を求める。さらに位数 m を変化させ、最大値を求めた結果が表の「誤差」の項に示されている。計算時間は実験の所要時間を節約するために次のように測定した。位数 m を $m = 0, N/20, N/10, \ldots, 19N/20$ の 20 通りに変化させ、それぞれの変換にかかる所要時間を求め、その合計を 20 倍することにより、すべての位数に対する計算時間を予測する。このようにしなければならない主要な理由は、現在の実装では前処理に時間がかかりすぎているという点なのであるが、この時間はおもに簡略化直接法の前処理に消費されており、この点で改良が必要である。なお、このようなサンプリングによる所要時間への影響を評価したところ、およそ 5 % で、計測誤差とほぼ同程度で済んでいる。

今回の実装においては $M = 341$ ではほとんど速度向上が確認されなかったが、$M = 511$ からは徐々に高速性を発揮して、$M = 4065$ では完全な直接法の 5 倍以上のスピードが出た。非常に大規模な問題とはいえ、ルジャンドル陪関数変換においてこれだけの性能向上を実現したのは初めてであろう。$M = 2730$ の所要時間と $M = 4095$ の所要時間の比はおよそ 2.8 倍で、これは $1.5^2 = 2.25$ よりもやや大きいが、$1.5^3 = 3.375$ よりも明らかに小さく、$O(M^3)$ を下回る計算量が実現されていることが確認できる。計算は数値的に安定で、相対誤差は十分に制御できている。K を固定して M を大きくすると相対誤差が少しずつ大きくなっているように見えているが、この原因は現在調査中である。

5 おわりに

本稿では、我々が提案している高速球面調和関数変換法のために必要となる、dual vector のための標本点選択アルゴリズムと、それを用いた高速変換の最初の実装の精度と性能について報告した。我々のアルゴリズムにより数値的に安定性な高速球面調和関数変換が実現できることが確認できた。

アルゴリズムの提案から実装まで 2 年もかかったのは、アルゴリズムが複雑であるということとともに、誤差と数値的な安定性の考察に時間がかかったからである。今回、我々の提案する標本点選択アルゴリズムにより安定な計算が実現できたが、誤差はスケーリングや用いるノルムによって大幅に異なってくる。このため、誤差の発生と伝播を詳細に追跡して今回得られた誤差の結果を解析し、さらに誤差の制御を行うための解析と実験を進めて行く必要がある。また、今回は性能のチューニングは何もしていないために絶対性能がかなり低くなっている。FMM の近似方法の改良、メモリ使用量の解析と削減、前処理時間の短縮と合わせて、実用化に向けてさらに研究を進めて行く必要がある。

謝辞

あらゆる意味で私達の研究を支えてくださっている杉原正顯教授に深く感謝いたします。また、有益なコメントを下さっている未来開拓のプロジェクトの皆様、HPC 研究会の参加者の皆様にも感謝いたします。この研究の一部は、学振（未来開拓）、豊田理化学研究所、および文部省科研費の支援を受けています。

参考文献

[1] 須田礼仁、「高速球面調和関数変換法」、情報処理学会研究報告 98-HPC-73、1998 年 10 月, pp. 37–42.

[2] 高見雅保、須田礼仁、杉原正顯、「FMM による Legendre 陪関数変換の高速化」、情報処理学会研究報告 99-HPC-78、1999 年 10 月、pp. 1–6.

[3] 高見雅保、須田礼仁、「Legendre 陪関数変換の高速計算に伴う FMM の改良」、豊田研究報告、第 53 巻、2000 年 5 月、pp. 77–84.