ハイパフォーマンス
コンピューティング 71－2
（1998. 5. 15）

球面調和関数変換の高速計算について

須田 礼仁
名古屋大学 工学研究科 計算理工学専攻

概要

球面調和関数変換は球面上の数値計算に欠かせない道具である。球面調和関数変換は Legendre 陪関数変換と Fourier 変換とに分解できるが、Legendre 陪関数変換には FFT のような簡単な高速算法が存在しないため、M 次までの変換に対して通常は計算量とメモリ消費量がともに $O(M^3)$ の直接法が用いられている。本稿では ultraspherical polynomical を経由して Legendre 陪関数変換を行うアルゴリズムを示し、計算量やメモリ消費量について比較検討を行う。理論的にはこの方法では $O(M^2 \log^2 M)$ の計算量で変換を行うことができる。また、実用においても、メモリ消費量と計算量とのバランスにおいて、直接法よりも有利となり得ることがわかった。

High performance computation of spherical harmonic transform

Reiji Suda
Computational Science and Engeneering, Fac. Eng., Nagoya University

Abstract

Spherical harmonic transform is an indispensable tool for numerical computations on spheres, and computed through associated Legendre transform and Fourier transform. While FFT is used for Fourier transform, the direct method, which requires $O(M^3)$ computations and memory, is used for associated Legendre transform. This paper proposes to use ultraspherical polynomial for associated Legendre transform. The proposed method runs theoretically with computational complexity of $O(M^2 \log^2 M)$. Its practicality is in a better tradeoff between computational complexity and memory requirements than the direct method.

§1 はじめに

球面調和関数は、3 次元空間中の 2 次元球面上で関数を近似するのにもっとも適している。一つ次元の低い円においては、三角関数がこれに対応しているが、関数を M 個の三角関数の線形結合で近似 (Fourier 変換) したり、三角関数の線形結合を M 点で評価 (Fourier 逆変換) したりするためには、FFT という高速な計算法があって $O(M \log M)$ で計算できることは周知の通りである。しかし球面調和関数変換/逆変換に対してはこのような簡単な高速計算法が知られておらず、現在でも大抵は $O(M^3)$ のメモリと計算量で計算されている。本稿ではこの球面調和関数変換の計算方法について、計算量とメモリ消費量を中心に考察する。

§1.1 球面調和関数

正規化因子を 4π とした球面調和関数を Y_n^m と書くことにする。すなわち

$$\int_0^{2\pi}\int_{-1}^{1} Y_n^m Y_{n'}^{m'*} = 4\pi\,\delta_{nn'}\,\delta_{mm'}$$

が成り立つ。ここで m のとりうる範囲は $|m| \le n$ であるが、対称性から実際には $0 \le m \le n$ のみを用いて近似することが可能である。従って、以下ではいわゆる M 次の三角形切断をした

$$g = \sum_{m=0}^{M}\sum_{n=m}^{M} g_n^m Y_n^m$$

のような変換を考える。具体的には、展開係数 g_n^m から、経度方向に I 分割、緯度方向に J 分割した格子点での関数値 $g(\lambda_i, \mu_j)$ を求める「評価 (逆変換)」問題と、格子点での値から展開係数を求める「展開 (順変換)」問題とがある。以下では、$I \approx 3M$, $J \approx 3M/2$ と仮定する。

球面上の点の位置を、緯度 ϕ と経度 λ に対して、$\mu = \sin(\phi)$ と λ で表すことにする。すると球面調和関数は

$$Y_n^m(\lambda, \mu) = P_n^m(\mu)\, e^{im\lambda}$$

のように、Legendre 陪関数 P_n^m と三角関数の積に表される。Legendre 陪関数 $P_n^m(x)$ は

$$\begin{aligned} P_m^m(x) &= \sqrt{\frac{(2m+1)!!}{(2m)!!}}\,(1-x^2)^{\frac{m}{2}} \\ P_{m+1}^m(x) &= \sqrt{2m+3}\,x\,P_m^m(x) \\ P_{n+1}^m(x) &= \frac{x P_n^m(x) - \varepsilon_n^m P_{n-1}^m(x)}{\varepsilon_{n+1}^m} \end{aligned} \qquad (1)$$

という漸化式で定義することができる。ここで、

$$\varepsilon_n^m = \sqrt{\frac{n^2-m^2}{4n^2-1}}$$

である。

この分解を用いると、

$$\begin{aligned} g^m(\mu_j) &= \sum_{n=m}^{M} g_n^m P_n^m(\mu_j) \\ g(\lambda_i,\mu_j) &= \sum_{m=0}^{M} g^m(\mu_j) e^{im\lambda_i} \end{aligned} \tag{2}$$

のように $O(M^3)$ の計算を 2 ステップ行うことで評価の計算が可能となる。さらに、後半は FFT を用いることにより $O(M^2 \log M)$ の計算量で計算が可能となる。従って、計算量の主要な問題は Legendre 陪関数変換の部分である。メモリ量においても Legendre 陪関数変換が主要な問題となり、すべての $P_n^m(\mu_j)$ を記憶しておくために $O(M^3)$ の記憶領域が必要である。

具体的な計算量とメモリ量は、Legendre 陪関数の対称性

$$P_n^m(-x) = (-1)^{n-m} P_n^m(x)$$

を用いて

$$\begin{aligned} g_0^m &= \sum_{n-m \bmod 2=0} g_n^m P_n^m(\mu_j) \\ g_1^m &= \sum_{n-m \bmod 2=1} g_n^m P_n^m(\mu_j) \\ g^m(\mu_j) &= g_0^m + g_1^m \\ g^m(-\mu_j) &= g_0^m - g_1^m \end{aligned}$$

と計算すれば半分になる。添字の範囲が $0 \le m \le M$ と $m \le n \le M$ であることから、Legendre 変換には $3/4M^3 + o(M^3)$ flop が必要であるということになる。

メモリは $P_n^m(\mu_j)$ を覚えておくだけで $3/8M^3 + o(M^3)$ word 必要である。$O(M^3)$ というメモリはかなり問題で、通常アプリケーションのメモリ消費のほとんどを占める。メモリ消費を抑えるには $P_n^m(\mu_j)$ をいちいち計算し直すことにすればよく、この場合覚えておくのは ε_n^m などの $O(M^2)$ ですむ。$P_n^m(\mu_j)$ を計算し直すための計算量は、(1) の漸化式の 1 ステップが 4 flop であることから、変換の計算量 (1 ステップが 2 flop) の 2 倍となり、変換を合わせた全体では 3 倍の計算量、すなわち $9/4M^3 + o(M^3)$ flop が必要となる。

順変換の場合には Legendre 陪関数の直交関係を用いて

$$\begin{aligned} g_n^m &= \frac{1}{2}\int_{-1}^{1} g^m(x) P_n^m(x) \mathrm{d}x \\ &= \sum_j \frac{w_j\, g^m(\mu_j)}{2} P_n^m(\mu_j) \end{aligned}$$

として計算する。ここで μ_j, w_j は Gauss 積分の分点と重みである。実際にはもちろん $w_j g^m(\mu_j)/2$ は先に計算しておくので、主要な計算の内容は逆変換で用いた行列を転置してベクトルに掛けていることになる。従って、順変換の計算量とメモリ量はともに逆変換と同じである。

§1.2 高速 Legendre 陪関数変換

Legendre 陪関数変換の $O(M^3)$ の計算量は、大抵の場合アプリケーションで最大の計算量である。特に、最近の計算機の能力の向上にともない、M が大きくとられる傾向にあるので、その重要性と問題性が増加している。

これに対し、Legendre 陪関数変換を高速に行う方法がいくつか提案されている。Orszag [2] は各 m に対して $O(M \log^2 M/\log\log M)$ の (全体の計算量は不明)、Mohlenkamp [5] は $O(M^2 \log^2 M)$ の方法を提案している。これらはいずれも Legendre 陪関数の別の関数族による高精度の展開を利用しているらしいが、なかなか素人がまねができるものではないようである。また、普通に行われている規模の問題に対して有効であるかどうかという点についても問題がある。

§2 多項式を経由する Legendre 陪関数変換

前節で見たように、Legendre 陪関数変換には計算量とメモリ消費量という 2 つの問題点がある。関数値 $P_n^m(\mu_j)$ をすべて記憶してある場合には $O(M^3)$ のメモリが必要で、メモリ消費を減らすために漸化式を計算しながら変換を行うと計算量は 3 倍になってしまう。

本節では、ultraspherical polynomial を中継する方法について述べ、その計算量とメモリ消費量について論ずる。理論的な計算量は $O(M^2 \log^2 M)$ となるが、実用的にもメモリ消費量と計算量とのバランスにおいて直接法よりも有利となり得ることが示される。

§2.1 多項式 Q_n^m の定義と基本的な性質

Legendre 陪関数 P_n^m に対し、次で定義される多項式 Q_n^m を考える。

$$Q_n^m = P_n^m / P_m^m$$

これは ultraspherical polynomial [1 の §4.7] と呼ばれるもので、Legendre 陪関数と同じ漸化式

$$Q^m_{n+1} = \frac{xQ^m_n - \varepsilon^m_n Q^m_{n-1}}{\varepsilon^m_{n+1}} \tag{3}$$

を満たす $(n-m)$ 次の多項式である。また、Legendre 陪関数の直交関係に従って、重みつきの直交関係

$$\int_{-1}^{1} Q^m_n(x)Q^m_{n'}(x)(1-x^2)^m \mathrm{d}x = 0 \qquad \text{for } n \neq n' \tag{4}$$

を満たす。また、$P^0_0 = 1$ であるから $Q^0_n = P^0_n$、すなわち Legendre 多項式である。また、Legendre 陪関数と同様に

$$Q^m_n(-x) = (-1)^{m+n} Q^m_n(x)$$

という対称性を持っている。

球面調和関数変換の計算は P^m_n の代わりに Q^m_n を用いて次のように行うことができる。すなわち (2) の第 1 式を

$$\tilde{g}^m(\mu_j) = \sum_{n=m}^{M} g^m_n Q^m_n(\mu_j)$$
$$g^m(\mu_j) = \tilde{g}^m(\mu_j) P^m_m(\mu_j)$$

と計算するのである。計算量は前半は $O(M^3)$ で後半は $O(M^2)$ である。従って、前半のステップの計算について主に考察すればよい。順変換でもほとんど同様である。

§2.2 計算方法と計算量

Q^m_n は $(n-m)$ 次の多項式であるから、$R^m = \sum g^m_n Q^m_n$ も $(M-m)$ 次の多項式である。従って、上記の変換の「前半のステップ」はさらに以下のように、「多項式を求める」と「多項式を評価する」という 2 つのステップに分解することができる。

$$R^m(x) = \sum_{n=m}^{M} g^m_n Q^m_n(x) \tag{5}$$

$$\tilde{g}^m(\mu_j) = R^m(\mu_j) \tag{6}$$

上の 2 つのステップ (5), (6) は次のように計算することができる。多項式 R^m を

$$R^m(x) = \sum_k \alpha_{mk} Z_k(x) \tag{7}$$

のように、基底多項式 $Z_k(x)$ の線形結合の形に書くことにする。係数の α_{mk} は漸化式 (3) を用いて計算できる。A, B, C を基底多項式の線形結合で表現された多項式 (定数の場合を含む) とすると、漸化式から、

$$AQ^m_{n+1} + BQ^m_n + CQ^m_{n-1}$$
$$= \left(B + \frac{x}{\varepsilon^m_{n+1}}A\right) Q^m_n + \left(C - \frac{\varepsilon^m_n}{\varepsilon^m_{n+1}}A\right) Q^m_{n-1} \tag{8}$$

のように 3 項のものを 2 項にすることができる。これを n が大きい方から小さい方にむかって繰り返し適用すれば最終的には $Q^m_m = 1$ に注意して

$$\begin{aligned} R^m &= \sum_n g^m_n Q^m_n \\ &= \cdots \\ &= DQ^m_{m+1} + EQ^m_m \\ &= E + \frac{x}{\varepsilon^m_{m+1}} D \end{aligned}$$

のように R^m を得ることができる。

計算量は基底多項式によって変化する。基底多項式として冪級数 $Z_k(x) = x^k$ をとると、(8) の flop 数は A の次数の 3 倍となる。基底多項式を漸化式

$$xZ_k(x) = \theta_{k+1}Z_{k+1}(x) + \nu_k Z_{k-1}(x) \tag{9}$$

を満たす直交多項式とする場合には A の次数の 6 倍の flop 数が必要となる。すべての m に対して $R^m(x)$ を求めるには、冪級数の場合は $1/2M^3 + o(M^3)$ flop, 直交多項式の場合は $M^3 + o(M^3)$ flop が必要となる。

この計算量はもっと直接的な方法を用いれば減らすことができる。

$$Q^m_n(x) = \sum_k \beta^m_{nk} Z_k(x)$$

と仮定する。次数と偶奇性から、右辺の総和の項数はおよそ $(n-m)/2$ となる。明らかに $\alpha_{mk} = \sum_n \beta^m_{nk} g^m_n$ であるから、多項式を求めるまでは $1/6M^3 + o(M^3)$ flop で計算できる。但し、この方法では β^m_{nk} をすべて覚えておくための $1/12M^3 + o(M^3)$ のメモリが必要となる。

後半 (6) の計算は線形結合 (7) をそのまま計算すればよい。計算量は対称性を利用して $3/4M^3 + o(M^3)$ flop となる。この計算は行列・行列積となっている。なぜなら、$R^m(x) = \sum_k \alpha_{mk} Z_k(x)$ であるから、行列 A を $a_{mk} = \alpha_{mk}$ とし、行列 Z を $z_{kj} = Z_k(\mu_j)$ としておくと、$R^m(\mu_j) = \sum_k a_{mk} z_{kj}$ となるからである。従って、これに対して Strassen の工夫を何段か適用して多少係数を小さくすることは可能であろう。

順変換は逆変換とは行列が転置になるだけであるからほとんど同じである。計算は

$$\tilde{\gamma}_j^m = \frac{w_j g^m(\mu_j)}{2} P_m^m(\mu_j)$$
$$g_n^m = \sum_j \tilde{\gamma}_j^m Q_n^m(\mu_j)$$

となるが、計算量が $O(M^3)$ である後半の計算を考えればよい。冪級数の場合には逆変換の α_{mk} に代って

$$\zeta_{mk} = \sum_j \tilde{\gamma}_j^m \mu_j^k$$

を中継する。ζ_{mk} 自体はこの式にしたがって計算すれば $3/4M^3 + o(M^3)$ の計算量と $O(M^2)$ のメモリで計算できる。これも計算としては行列・行列積となっているので、Strassen のアルゴリズムを適用できる。そのあと、ζ_{mk} を拡張する形で

$$\eta_{nk}^m = \sum_j \tilde{\gamma}_j^m \mu_j^k Q_n^m(\mu_j)$$

を定義すると、漸化式から

$$\eta_{mk}^m = \zeta_{mk}$$
$$\eta_{m+1k}^m = \zeta_{mk-1}/\varepsilon_{m+1}^m$$
$$\eta_{n+1k}^m = \frac{1}{\varepsilon_{n+1}^m}\eta_{nk+1}^m - \frac{\varepsilon_n^m}{\varepsilon_{n+1}^m}\eta_{n-1k}^m$$

のような関係が得られることがわかる。目的とする g_n^m は η_{n0}^m にほかならない。計算量は逆変換の場合と同じく $1/2M^3 + o(M^3)$ となっている。

漸化式 (9) を満たす基底多項式を用いる場合もほとんど同様で、

$$\zeta_{mk} = \sum_j \tilde{\gamma}_j^m Z_k^m(\mu_j)$$
$$\eta_{nk}^m = \sum_j \tilde{\gamma}_j^m Z_k^m(\mu_j) Q_n^m(\mu_j)$$

のように定義すると、簡単な計算から

$$\eta_{n+1k}^m = \frac{\theta_{k+1}}{\varepsilon_{n+1}^m}\eta_{nk+1}^m + \frac{\nu_k}{\varepsilon_{n+1}^m}\eta_{nk-1}^m - \frac{\varepsilon_n^m}{\varepsilon_{n+1}^m}\eta_{n-1k}^m$$

などという漸化式が得られる。計算量はやはり逆変換と同じく $M^3 + o(M^3)$ flop である。また、逆変換の場合と同様に $O(M^3)$ のメモリを使えば $1/6M^3 + o(M^3)$ の計算量で計算することもできる。

§2.3 冪級数を用いた計算の理論的な計算量

この節では冪級数を用いた逆変換のアルゴリズムが $O(M^2 \log^2 M)$ の計算量を持つことを示す。順変換については現在のところ示されていないが、その他の点についてはすべて順変換と逆変換は並行に話が進むので、おそらくこの点についても同じであると思われる。

前節の (8) 付近の説明から明らかなように、漸化式を繰り返し適用することにより、一般的に $n > n_0 + 1$ に対して

$$\begin{aligned} Q_n^m(x) =& \rho[n, n_0+1](x)\, Q_{n_0+1}^m(x) \\ &+ \sigma[n, n_0](x)\, Q_{n_0}^m \end{aligned} \tag{10}$$

と書くことができる。ρ と σ は多項式で、その次数は ρ が $(n - n_0 - 1)$ 次、σ が $(n - n_0 - 2)$ 次である。関数の偶奇性から、ρ と σ は一方が偶関数、もう一方が奇関数となるはずなので、非零の項は半分しかない。

この ρ と σ を用いて、以下のように分割統治法を行う。

$$R^m = \sum_{n=m}^{M} g_n^m Q_n^m$$

を $m_0 = (M+m)/2$ で 2 つに分けて

$$R^m = \sum_{n=m}^{m_0-1} g_n^m Q_n^m + \sum_{n=m_0}^{M} g_n^m Q_n^m$$

とすれば、それぞれ

$$\begin{aligned} \sum_{n=m}^{m_0-1} g_n^m Q_n^m &= r_{m+1} Q_{m+1}^m + s_m Q_m^m \\ \sum_{n=m_0}^{M} g_n^m Q_n^m &= r_{m_0+1} Q_{m_0+1}^m + s_{m_0} Q_{m_0}^m \end{aligned} \tag{11}$$

と書けるはずであるから、これから

$$\begin{aligned} R^m =& (r_{m+1} + \rho[m_0+1, m+1]\, r_{m_0+1} \\ &+ \rho[m_0, m+1]\, s_{m_0}) Q_{m+1}^m \\ &+ (s_m + \sigma[m_0+1, m]\, r_{m_0+1} \\ &+ \sigma[m_0, m]\, s_{m_0}) Q_m^m \end{aligned} \tag{12}$$

となり、R^m を計算することができる。この (12) の主要な計算は多項式同士の積であるが、それぞれ T 次の多項式であるとすると、これは FFT を用いれば $O(T \log T)$ で求められる。次数 T は $m = 0$ で最大でほぼ $M/2$ 次であるから、(12) の計算量は $O(M \log M)$ ということになる。

さらに (11) のそれぞれの式について分割してやることが可能である。この場合は (12) に対応する式が 2 つできるわけであるが、出てくる多項式の次数が半分になるので、合計の計算量はやはり $O(M\log M)$ である。

この分割を $O(\log M)$ レベル行うと多項式の次数は定数となり、(12) にあたる式は $O(M)$ 個となり、その計算量の合計は $O(M)$ である。結局、それぞれのレベルでの計算量は $O(M\log M)$ で、$O(\log M)$ レベルあるので、R^m が求まるまでの計算量は $O(M\log^2 M)$ となる。これを各 m に対して計算するので、全体としては $O(M^2\log^2 M)$ である。

上記のアルゴリズムの高速化の鍵は、多項式の乗算が FFT を用いて $O(T\log T)$ でできることにあった。しかし、実際には FFT を用いて速度の向上が得られるのは非常に大きな問題に限られる。多項式の積に対して、もっと小さい問題に対しても有効な、計算量が $O(T^{\log_2 3}) \approx O(T^{1.58})$ となるアルゴリズムが知られている。これを使うことにすれば、計算量は $O(M^{2.58}\log M)$ となる。

後半部分は多項式 R^m を μ_j で評価するという問題である。これに対しては、多項式の高速除算を用いた $O(M\log^2 M)$ の方法が知られている。これを用いれば $O(M^2\log^2 M)$ の計算量になるが、この方法はさらに大規模な M でなければ有効でない。なお、Z として Legendre 多項式を用いた場合には場合には、その評価のために Alpert ら [4] による $O(M^2\log M)$ の方法が知られている。

一方、先にのべたように、後半の計算は行列·行列積の形になっている。Strassen の方法を用いればこれは $O(M^{2.807})$ で計算でき、$O(M^3)$ よりも高速である。Pan のアルゴリズムによれば $O(M^{2.795})$ でも計算できることになる。

以上のように、Q_n^m を経由する方法の理論的な計算量は $O(M^2\log^2 M)$ で、これまでに知られている高速計算法に引けを取らない。また、$o(M^3)$ となるような他の計算方法も存在する。しかし、実際的な問題のサイズにおいて、これが有効になるかどうかは疑問である。

§2.4 多項式を中継する方法の計算誤差

多項式を途中で経由する方法では、多項式をどのように表現するかで計算の安定性が非常に異なる。以下の実験では、g_n^m を球面上の流体解析プログラム [3] を参考にして与え、$P_n^m(\mu_j)$ を直接用いる方法 (直接法) と多項式を経由する方法の 2 通りで逆変換の計算をさせた。その結果得られた結果を 2 本のベクトルと見なして単純に 2 ノルムを取り、直接法をベースとして相対誤差を評価した。多項式の表現として冪級数を用いた場合には次のようになる。

J	16	32	64	128	256
M	10	21	42	85	170
誤差	2e-14	8e-12	1e-7	6e+1	1e+16

このように、問題規模が大きくなると、急に精度が悪化して、実際に興味のあるサイズでは全然役に立たない。得られた冪級数の係数 α_{mk} を見てみると、非常に大きな値が多くあり、多項式の評価の際にあちこちで桁落ちが生じて誤差を生んでいるらしいことがわかる。

一方、基底多項式として Legendre 多項式をとると、次のように安定して計算ができている。

J	256	512	1024
M	170	340	680
誤差	6e-12	1e-11	1e-11

上記の結果は漸化式を用いて計算しているが、行列表現を用いるとこのようには安定に計算されなかった。ここでも漸化式というもののもつ不思議な安定性の一端が見えている。

§2.5 誤差の改良

前節で見たように、冪級数を用いると数値的に不安定で実用上支障がある。また、Legendre 多項式を基底多項式として用いた場合も、他の問題に対しても安定かどうか、また順変換も同様に安定かどうかという問題がある。

このような誤差の問題を解決する簡単な方法がある。R^m の次数が高くなると誤差が大きくなっているので、強制的に R^m の次数を抑えるのである。適当な定数 S を設定し、計算の途中で多項式の次数が S となったところで漸化式の適用を打ち切り、そこで多項式の評価に移るのである。先の式を流用すれば、(11) を求めた後 (12) を行わず、(11) の多項式に μ_j を代入して計算するということである。順変換の計算は、η_{nk}^m の k を S 以下に制限し、必要な n に対する η_{nk}^m を定義式から直接計算すればよい。下の表は $J=256$ の場合の問題に対して冪級数の次数に制限を加えた場合の誤差と S との関係を示しており、それなりの精度で計算ができることがわかる。

S	5	10	21	42	85
誤差	6e-16	5e-15	5e-13	9e-8	2e-1

計算量は、前半の多項式を生成する部分では $3/2M^2S+o(M^2S)$ flop となってオーダーが落ちるが、後半の評価の部分では (11) に見るように多項式が 2 つずつ組になっているため 2 倍の $3/2M^3+o(M^3)$ flop となる。

必要なメモリ量は、P_n^m を記憶している場合、直接法の $2/S$ 倍になる。これでは $O(M^3)$ だが、

表 1. 各種の計算方法による計算量・メモリ量・安定性

中間表現	計算方法	計算量	メモリ	安定性	コメント
直接	行列	$3/4M^3$	$3/8M^3$	良	メモリに余裕があれば
直接	漸化式	$9/4M^3$	$O(M^2)$	良	メモリを節約、計算も安定
冪級数	漸化式	$5/4M^3$	$O(M^2)$	最悪	数値的に不安定で使えない
Legendre 多項式	漸化式	$7/4M^3$	$O(M^2)$	まし	誤差が我慢できれば
多項式	行列	$11/12M^3$	$1/12M^3$	だめ？	今後の検討課題
次数制限冪級数	漸化式	$(1+S/M)3/2M^3$	$3/(4S)M^3$	S 次第	場合によってはよい選択かも

P_n^m の再計算には計算量が $3M^3+o(M^3)$ flop 必要である。m 方向の何らかの漸化式が使えれば、再計算の高速化が可能となるであろう。

§3 まとめ

考察の結果をまとめると、以下のようになる。

(i) もっとも単純な方法は P_n^m の線形結合をそのまま計算する方法で、$P_n^m(\mu_j)$ をメモリに格納しておくと $3/4M^3+o(M^3)$ flop の計算量と $O(M^3)$ のメモリが必要である。

(ii) この $P_n^m(\mu_j)$ をいちいち計算し直すと、メモリ量は $O(M^2)$ で済むようになるが、計算量は $9/4M^3+o(M^3)$ となる。

(iii) 途中で多項式の冪級数形を経由する方法では、メモリ量が $O(M^2)$ で計算量が $5/4M^3+o(M^3)$、またはメモリが $O(M^3)$ で計算量が $11/12M^3+o(M^3)$ となる。理論的には $O(M^2\log^2 M)$ の計算方法も存在する。しかし、この方法は数値的には不安定である。

(iv) 途中で出てくる多項式を Legendre 多項式の線形結合で表現すると計算量が $7/4M^3+o(M^3)$ でメモリ量が $O(M^2)$ の方法が実現できる。しかし、現在のところどれほどの問題まで Legendre 多項式で安定に展開できるかは不明である。メモリを $O(M^3)$ 許せば、計算量は $11/12M^3+o(M^3)$ となるが、数値的に不安定であるらしい。

(v) 多項式の次数を一定値 S で抑えると、比較的安定に計算が進む。計算量は $3/2M^3+o(M^3)$ であるが、メモリは $O(M^3/S)$ となってしまう。

以上を総合すると、メモリが十分にあるのであれば、単純な計算方法が精度の点でも計算量の点でも優れている。$O(M^2)$ のメモリですむ計算方法としては、Legendre 多項式の線形結合を経由する方法が $7/4M^3+o(M^3)$ flop でよいようである。高い精度を要求する場合にはそれでも直接法で漸化式を用いるほかない。すこしメモリを使ってよければ次数制限つきの冪級数もよい選択肢となり得る。多項式を経由する方法はいずれも行列・行列積を使用するので、Strassen の工夫を数段用いることで、若干係数を減らすことが可能であると予想される。

今回は触れることができなかったが、高性能 RISC プロセッサや並列計算環境での性能について検討する必要がある。RISC については、$O(M^3)$ のメモリを必要とする行列・ベクトル積を用いる計算方法はキャッシュブロック化が出来ないため、性能が低下する恐れがある。次数制限のある冪級数を経由する場合は、S のサイズの計算を繰り返していることになっているので、S が適当な大きさであれば、工夫次第では性能が出せると思われる。

しかし、なぜ Legendre 多項式による展開を経由すると安定なのかは現在のところ不明である。3 項漸化式を満たす多項式系であれば Z_k^m として何を選んでも計算量は変わらないので、さらに安定性のよいものがあれば、そのほうが望ましい。また、漸化式の係数が特別な値であれば計算量を減らすことができるかも知れない。

また、多項式 f の表現方法として、$(x, f(x))$ のペアを n 個準備する方法もある。しかし、この方法は計算量、メモリ量ともあまり芳しくないようである。安定性については現在のところ不明である。

参考文献

[1] G. Szegö, *Orthogonal polynomials*, AMS, Colloquium publications Vol. XXIII (1959).

[2] S. A. Orszag, "Fast Eigenfunction Transforms," *Science and Computers*, Vol. 10, (1986) pp. 23–30.

[3] 余田成男・山田道夫・石岡圭一、「スペクトル法による球面上の流体方程式の数値解法」、京都大学大型計算機センター広報 Vol. 23, No. 5, (1990) pp. 283–290.

[4] B. Alpert and V. Rokhlin, "A fast algorithm for the evaluation of Legendre expansions," *SIAM J. Sci. Stat. Comp.*, Vol. 12, No. 1, (1991) pp. 158–179.

[5] M. J. Mohlenkamp, "A Fast Transform for Spherical Harmonics," *Ph.D Dissertation*, Yale University, (1997).