The DOI for this manuscript is
DOI:10.2151/jmsj. 2018-019
J-STAGE Advance published date: January 23, 2018

A New Recurrence Formula for Efficient Computation of Spherical Harmonic Transform

Keiichi Ishioka[1]

[1] *Graduate School of Science, Kyoto University*

Corresponding author: Keiichi Ishioka, Graduate School of Science, Kyoto University, Kitashirakawa-Oiwake-cho, Sakyo-ku, Kyoto 606-8502, Japan.
E-mail: ishioka@gfd-dennou.org

Abstract

A new recurrence formula to calculate the associated Legendre functions is proposed for efficient computation of the spherical harmonic transform. This new recurrence formula makes the best use of the fused multiply-add (FMA) operations implemented in modern computers. The computational speeds in calculating the spherical harmonic transform are compared between a numerical code in which the new recurrence formula is implemented and other code using the traditional recurrence formula. This comparison shows that implementation of the new recurrence formula contributes to a faster transform. Furthermore, a scheme to maintain the accuracy of the transform, even when the truncation wavenumber is huge, is also explained.

Keywords: associated Legendre functions, recurrence formula, spherical harmonic transform, spectral method, numerical library

1. Introduction

In many global atmospheric general circulation models, e.g., OpenIFS (ECMWF, 2017), DCPAM (Takahashi, et al., 2016), *etc*, the spectral method with spherical harmonics is used in horizontal discretization. To evaluate the nonlinear terms in these models, the transform method, which was introduced by Orszag (1970), is used for computational efficiency. The transform method requires two kinds of transforms: Grid point data are transformed to coefficients for the spherical harmonics in the forward transform and the reverse is done in the backward transform. We call the two kinds of transforms the spherical harmonic transform hereinafter. Because the spherical harmonic transform must be computed many times per one time step in time integrations of the general circulation models, it is a fundamental tool, but it is desirable to reduce the computational time for the transform. Note that the spherical harmonic transform is also used in other research areas, e.g., cosmic microwave background studies (Reinecke, et al., 2006), planetary dynamos (Wicht and Tilgner, 2010), *etc*.

The spherical harmonic transform consists of the discrete Fourier transform for the longitudinal direction and a transform from coefficients of associated Legendre functions to function values at specified nodes, which we call the associated Legendre function transform hereinafter, for the latitudinal direction. The discrete Fourier transform can be computed with the fast Fourier transform (FFT) algorithm and its computational complexity is $O(M^2 \log_2 M)$, where M is the truncation wavenumber of the spherical harmonic transform. The associated Legendre function transform has a computational complexity of $O(M^3)$, and so it is one of the most time-consuming parts in the computation of dynamics in a spherical spectral model when the horizontal resolution of the model becomes fine. For the associated Legendre function transform, no exact fast algorithm has been discovered. However, several fast algorithms have been proposed to compute

the associated Legendre function transform approximately, e.g., Healy, et al. (2003) and Seljebotn (2012). In particular, in Seljebotn (2012), a fast algorithm the computational complexity of $O(M^2 \log^2 M)$ was proposed and the implementation detail was described. Although such approximate fast algorithms may be promising when M is large because of the asymptotic behavior of the computational complexity, they have disadvantages, for example, that the implementations are very complicated, that they conduct the transform approximately within a given accuracy, and that they require a huge memory area when M is large.

As a strategy to deal with the computational complexity for the associated Legendre function transform other than exploring a fast algorithm, optimizing corresponding numerical codes is a natural choice. Schaeffer (2013) showed that his highly optimizing numerical code for the spherical harmonic transform spent much less computational time than that spent by other competitive codes, which included numerical codes based on fast algorithms. While several tuning techniques have been incorporated into his numerical code, one of the most essential parts is that the associated Legendre functions are computed "on-the-fly" during the transform rather than computed and stored before the transform. The reason why this on-the-fly computation contributes to speed tuning is that it requires much less memory ($O(M^2)$) than that required when computed in advance ($O(M^3)$). Therefore, this takes advantage of the cache memory on modern computers. However, computing the associated Legendre functions on-the-fly has computational complexity of the same order as that for the transform itself ($O(M^3)$). Because the associated Legendre functions are computed by using a recurrence formula, a more efficient recurrence formula requiring less computation would be beneficial.

In this paper, we propose a new efficient recurrence formula for the associated Legendre functions. We show that the recurrence formula is effective especially on recent

computers that have fused multiply-add (FMA) operations. The recurrence formula also has the advantage of maintaining accuracy compared to the commonly used recurrence formula. To demonstrate that the adoption of the new recurrence formula contributes to the transform speed, we develop a numerical code that adopts the proposed recurrence formula and compare its speed with the speed of the code proposed by Schaeffer (2013). In developing the code, care must be taken to maintain the accuracy of the transform when the truncation wavenumber is huge. Accordingly, we also introduce a scheme to maintain the transform accuracy. Furthermore, we explain several tuning techniques incorporated into the developed numerical code.

The remainder of the present paper is organized as follows. After we describe the fundamentals of the spherical harmonic transform in Section 2, we propose a new recurrence formula for the associated Legendre functions and explain why the new formula is efficient compared to the commonly used recurrence formula in Section 3. In Section 4, a scheme for the transform accuracy to be maintained even when the truncation wavenumber is huge is introduced, and several tuning techniques implemented in developing a numerical code for the transform are described in Section 5. In Section 6, the speed and accuracy performance of the developed numerical code are compared with those of another numerical code based on Schaeffer (2013). A summary and a discussion are presented in Section 7.

2. Fundamentals of the spherical harmonic transform

In the spherical spectral method, dependent variables in the governing equations of the model are expanded by using spherical harmonics as follows:

$$f(\lambda, \mu) = \sum_{n=0}^{M} \sum_{m=-n}^{n} s_n^m Y_n^m(\lambda, \mu), \tag{1}$$

where f is a dependent variable, such as temperature, s_n^m is the expansion coefficient, λ is the longitude, $\mu = \sin\phi$, ϕ is the latitude, and M is the truncation wavenumber. The spherical harmonics, Y_n^m, is defined as

$$Y_n^m(\lambda, \mu) = P_n^{|m|}(\mu) e^{im\lambda}. \tag{2}$$

Here, $P_n^m(\mu)$ is the associated Legendre function, which is defined as

$$\begin{aligned} P_n^m(\mu) =& \sqrt{(2n+1)\frac{(n-m)!}{(n+m)!}}\frac{1}{2^n n!} \\ & \times (1-\mu^2)^{m/2}\frac{d^{n+m}}{d\mu^{n+m}}(\mu^2-1)^n \quad (0 \le m \le n). \end{aligned} \tag{3}$$

Note that $P_n^m(\mu)$ is normalized to satisfy the following orthogonality relation:

$$\int_{-1}^{1} P_n^m(\mu) P_{n'}^m(\mu) d\mu = 2\delta_{nn'}. \tag{4}$$

Here, $\delta_{nn'}$ is the Kronecker delta. By this orthogonality, the following "inverse" of (1) holds:

$$s_n^m = \frac{1}{4\pi}\int_0^{2\pi}\int_{-1}^{1} f(\lambda, \mu) Y_n^{-m}(\lambda, \mu) d\mu d\lambda. \tag{5}$$

Because $Y_n^{-m} = (Y_n^m)^*$, where $(\ )^*$ indicates the complex conjugate, $s_n^{-m} = (s_n^m)^*$ must be satisfied if $f(\lambda, \mu)$ is a real function. In the following, it is assumed that this constraint is satisfied.

In most spherical spectral models, the expansion (1) is evaluated on grid points (λ_k, μ_j) $(k = 0, 1, \ldots, K-1;\ j = 1, 2, \ldots, J)$. That is,

$$f(\lambda_k, \mu_j) = \sum_{n=0}^{M}\sum_{m=-n}^{n} s_n^m Y_n^m(\lambda_k, \mu_j). \tag{6}$$

Here, μ_j $(j = 1, 2, \ldots, J)$ are called Gaussian nodes, which are defined as zero points (sorted in ascending order) of $P_J^0(\mu)$, and $\lambda_k = 2\pi k/K$ $(k = 0, 1, \ldots, K-1)$. When both $K > 2M$ and $J > M$ hold, the discrete counterpart of (5), that is, the "inverse" of (6), is

given as

$$s_n^m = \frac{1}{2K} \sum_{k=0}^{K-1} \sum_{j=1}^{J} w_j f(\lambda_k, \mu_j) Y_n^{-m}(\lambda_k, \mu_j). \tag{7}$$

Here, w_j $(j = 1, 2, \ldots, J)$ is called the Gaussian weight and is defined as

$$w_j \equiv \frac{2(2J-1)(1-\mu_j^2)}{\{JP_{J-1}^0(\mu_j)\}^2}. \tag{8}$$

In this paper, we call the transform from s_m^n to $f(\lambda_k, \mu_j)$ as described by (6) the backward transform and the transform from $f(\lambda_k, \mu_j)$ to s_m^n as described by (7) the forward transform.

The backward transform (6) and the forward transform (7) can be divided into two stages, respectively, as follows:

$$g_j^m = \sum_{n=m}^{M} s_n^m P_n^m(\mu_j) \quad (m = 0, 1, \ldots, M;\ j = 1, 2, \ldots, J), \tag{9}$$

$$f(\lambda_k, \mu_j) = g_j^0 + 2\mathrm{Re}\left(\sum_{m=1}^{n} g_j^m e^{im\lambda_k} \right) \quad (k = 0, 1, \ldots, K-1;\ j = 1, 2, \ldots, J), \tag{10}$$

and

$$g_j^m = \frac{1}{K} \sum_{k=0}^{K-1} f(\lambda_k, \mu_j) e^{-im\lambda_k} \quad (m = 0, 1, \ldots, M;\ j = 1, 2, \ldots, J), \tag{11}$$

$$s_n^m = \frac{1}{2} \sum_{j=1}^{J} w_j g_j^m P_n^m(\mu_j) \quad (m = 0, 1, \ldots, M;\ n = m, m+1, \ldots, M). \tag{12}$$

The operation Re( ) indicates that the real part of the argument is taken. While the computations of (10) and (11) can be done with a relatively low cost by using FFT, for which highly optimized libraries are available, e.g., FFTW (Frigo and Johnson, 2005), no exact fast algorithm has been found and hence optimization becomes important for the computations of (9) and (12).

3. A new recurrence formula

In the computation of the associated Legendre function transforms (9) and (12), the simplest implementation is to compute all the $P_n^m(\mu_j)$ needed in advance and store them

in the computer memory to be used repeatedly. With this implementation, however, the amount of data that should be loaded is of the same order as the number of required operations; such an implementation fails to take advantage of the cache memory on modern computers. To overcome this problem, it is natural to try to compute all the associated Legendre functions "on-the-fly", that is, to compute them simultaneously every time with the transform. This is one of the most essential points proposed in Schaeffer (2013), although the idea of computing the associated Legendre function on-the-fly itself has been adopted previously in other studies, such as Ishioka, et al. (2000), Rivier, et al. (2002), and Reinecke (2011). In these on-the-fly computations, the following recurrence formula is adopted:

$$P_{n+1}^m(\mu) = (\mu P_n^m(\mu) - \epsilon_n^m P_{n-1}^m(\mu))/\epsilon_{n+1}^m, \tag{13}$$

where $\epsilon_n^m = \sqrt{(n^2-m^2)/(4n^2-1)}$. This recurrence formula requires three multiplications and one addition per one step even if $(-\epsilon_n^m)$ and $1/\epsilon_{n+1}^m$ are computed and stored in advance. On the other hand, in computing (9) and (12), two multiplications and two additions are required per one n for $m \neq 0$, because both real and imaginary parts of s_n^m and $(w_j g_j^m)$ should be treated. That is, the traditional recurrence formula (13) requires the same number of operations as the transform itself per one n. Note that, a multiplication and an addition have the same computational cost on modern computers. Furthermore, modern computers have FMA operations, and on such computing platforms, a pair of multiplication and addition has the same computational cost as a single multiplication or a single addition. Therefore, on computers having FMA, the traditional recurrence formula (13) requires more computational cost than the transform itself because the recurrence formula requires operations in which the computation equals that of three FMAs while the transform itself requires two FMAs. In the following, we derive a new recurrence formula to reduce the number of required operations.

The traditional recurrence formula (13) can be rewritten as follows:

$$\mu P_n^m(\mu) = \epsilon_{n+1}^m P_{n+1}^m(\mu) + \epsilon_n^m P_{n-1}^m(\mu). \tag{14}$$

By multiplying μ on both sides of (14) and applying (14) on the right-hand side recursively, we obtain,

$$\mu^2 P_n^m(\mu) = \epsilon_{n+1}^m \epsilon_{n+2}^m P_{n+2}^m(\mu) + \{(\epsilon_{n+1}^m)^2 + (\epsilon_n^m)^2\} P_n^m(\mu) + \epsilon_n^m \epsilon_{n-1}^m P_{n-2}^m(\mu). \tag{15}$$

In this form of recurrence formula, the associated Legendre functions, which have even numbers in the lower suffix, are treated independently of those with odd suffixes. First, let us consider the Legendre functions $P_n^m(\mu)$ for which $(n-m)$ is an odd number, which means they are odd functions of μ. If we write $n = m+2l+1$ $(l = 0, 1, \ldots)$, the recurrence formula (15) can be rewritten as

$$\begin{aligned} \mu^2 P_{m+2l+1}^m(\mu) =& \epsilon_{m+2l+2}^m \epsilon_{m+2l+3}^m P_{m+2l+3}^m(\mu) \\ &+ \{(\epsilon_{m+2l+2}^m)^2 + (\epsilon_{m+2l+1}^m)^2\} P_{m+2l+1}^m(\mu) \\ &+ \epsilon_{m+2l+1}^m \epsilon_{m+2l}^m P_{m+2l-1}^m(\mu). \end{aligned} \tag{16}$$

Let us introduce α_l^m and $p_l^m(\mu)$ to satisfy,

$$P_{m+2l+1}^m(\mu) = \mu \alpha_l^m p_l^m(\mu). \tag{17}$$

That is, $p_l^m(\mu)$ is an even function of μ. Substituting (17) into (16) yields,

$$\begin{aligned} \mu^2 \alpha_l^m p_l^m(\mu) =& \epsilon_{m+2l+2}^m \epsilon_{m+2l+3}^m \alpha_{l+1}^m p_{l+1}^m(\mu) \\ &+ \{(\epsilon_{m+2l+2}^m)^2 + (\epsilon_{m+2l+1}^m)^2\} \alpha_l^m p_l^m(\mu) \\ &+ \epsilon_{m+2l+1}^m \epsilon_{m+2l}^m \alpha_{l-1}^m p_{l-1}^m(\mu). \end{aligned} \tag{18}$$

To simplify (18), we impose the following condition on α_n^m:

$$\epsilon_{m+2l+2}^m \epsilon_{m+2l+3}^m \alpha_{l+1}^m = -\epsilon_{m+2l+1}^m \epsilon_{m+2l}^m \alpha_{l-1}^m. \tag{19}$$

This means that the coefficients multiplied on $p_{l+1}^m(\mu)$ and $p_{l-1}^m(\mu)$ on the right-hand side of (18) have the same absolute value but opposite signs. From (19), the following equation is derived.

$$\alpha_{l+1}^m \alpha_l^m = -\frac{\epsilon_{m+2l+1}^m \epsilon_{m+2l}^m}{\epsilon_{m+2l+3}^m \epsilon_{m+2l+2}^m} \alpha_l^m \alpha_{l-1}^m. \tag{20}$$

Applying (20) recurrently yields,

$$\alpha_{l+1}^m \alpha_l^m = (-1)^l \frac{\epsilon_{m+3}^m \epsilon_{m+2}^m}{\epsilon_{m+2l+3}^m \epsilon_{m+2l+2}^m} \alpha_1^m \alpha_0^m. \tag{21}$$

While choosing α_0^m and α_1^m is an arbitrary decision, let us choose them as

$$\alpha_0^m = \frac{1}{\epsilon_{m+1}^m},\ \alpha_1^m = \frac{\epsilon_{m+1}^m}{\epsilon_{m+2}^m \epsilon_{m+3}^m}. \tag{22}$$

The reason for this choice is explained later. Substituting (22) into (21) yields,

$$\alpha_{l+1}^m \alpha_l^m = \frac{(-1)^l}{\epsilon_{m+2l+3}^m \epsilon_{m+2l+2}^m}. \tag{23}$$

By multiplying $(-1)^l \alpha_l^m$ on both sides of (18) and then using (23), we finally obtain,

$$p_{l+1}^m(\mu) = \left[(-1)^l (\alpha_l^m)^2 \mu^2 - (-1)^l (\alpha_l^m)^2 \left\{ (\epsilon_{m+2l+2}^m)^2 + (\epsilon_{m+2l+1}^m)^2 \right\} \right] p_l^m(\mu) + p_{l-1}^m(\mu). \tag{24}$$

Furthermore, if we introduce a_l^m and b_l^m as

$$a_l^m = (-1)^l (\alpha_l^m)^2, \quad b_l^m = -a_l^m \left\{ (\epsilon_{m+2l+2}^m)^2 + (\epsilon_{m+2l+1}^m)^2 \right\}, \tag{25}$$

we can rewrite (24) as

$$p_{l+1}^m(\mu) = (a_l^m \mu^2 + b_l^m) p_l^m(\mu) + p_{l-1}^m(\mu). \tag{26}$$

By using the new recurrence formula (26), the lower suffix l of $p_l^m(\mu_j)$ can be increased by one with two FMA operations if μ_j^2, a_l^m, and b_l^m are computed and stored in advance; To store them, only $O(M^2 + J)$ memory area is required. That is, one FMA operation is required per one increase in the corresponding suffix n of P_n^m and this FMA operation

cost is one-third the cost of the traditional recurrence formula (13). One might think that the efficiency of the new recurrence formula (26) would be lost if it must also be used in the case that $n-m$ is even. However, this problem can be circumvented as follows. The backward transform (9) is rewritten as follows:

$$g_j^m = \sum_{l=0}^{\left[\frac{M-m}{2}\right]} s_{m+2l}^m P_{m+2l}^m(\mu_j) + \sum_{l=0}^{\left[\frac{M-m-1}{2}\right]} s_{m+2l+1}^m P_{m+2l+1}^m(\mu_j). \tag{27}$$

Here, [ ] indicates the floor function. On the right-hand side of (27), the first and the second terms correspond to the symmetric and the anti-symmetric parts about $\mu = 0$, respectively. By using (14), we can rewrite the first term as follows:

$$\begin{aligned}\sum_{l=0}^{\left[\frac{M-m}{2}\right]} s_{m+2l}^m P_{m+2l}^m(\mu_j) &= \frac{1}{\mu_j} \sum_{l=0}^{\left[\frac{M-m}{2}\right]} s_{m+2l}^m \{\epsilon_{m+2l+1}^m P_{m+2l+1}^m(\mu_j) + \epsilon_{m+2l}^m P_{m+2l-1}^m(\mu_j)\} \\ &= \frac{1}{\mu_j} \sum_{l=0}^{\left[\frac{M-m}{2}\right]} s_{m+2l}^m \epsilon_{m+2l+1}^m P_{m+2l+1}^m(\mu_j) \\ &\quad + \frac{1}{\mu_j} \sum_{l=0}^{\left[\frac{M-m}{2}\right]-1} s_{m+2l+2}^m \epsilon_{m+2l+2}^m P_{m+2l+1}^m(\mu_j).\end{aligned} \tag{28}$$

Note that, in the derivation of (28), we used the fact that (14) holds even when $n = m$ by defining $P_{m-1}^m(\mu) = 0$. Substituting (28) into (27) and using (17), we finally obtain,

$$g_j^m = \sum_{l=0}^{\left[\frac{M-m}{2}\right]} (t_l^m \alpha_l^m) p_l^m(\mu_j) + \mu_j \sum_{l=0}^{\left[\frac{M-m-1}{2}\right]} (s_{m+2l+1}^m \alpha_l^m) p_l^m(\mu_j). \tag{29}$$

Here,

$$t_l^m = \begin{cases} s_{m+2l}^m \epsilon_{m+2l+1}^m + s_{m+2l+2}^m \epsilon_{m+2l+2}^m & (l = 0, 1, \ldots, [\frac{M-m}{2}] - 1) \\ s_{m+2l}^m \epsilon_{m+2l+1}^m & (l = [\frac{M-m}{2}]). \end{cases} \tag{30}$$

Therefore, the backward associated Legendre function transform can be computed by using (29) with $p_l^m(\mu_j)$ $(l = 0, 1, \ldots, [\frac{M-m}{2}])$. Furthermore, in (29), the first term and the second term must be computed only for a hemisphere, because $p_l^m(\mu)$ is an even function of μ. That is, if we assume that J is an even number, g_j^m $(j = 1, 2, \ldots, J)$ can be computed

for each m $(m = 0, 1, \ldots, M)$ as

$$(g_j^m)_s = \sum_{l=0}^{\left[\frac{M-m}{2}\right]} (t_l^m \alpha_l^m) p_l^m(\mu_j) \quad (j = 1, 2, \ldots, J/2), \tag{31}$$

$$(g_j^m)_a = \mu_j \sum_{l=0}^{\left[\frac{M-m-1}{2}\right]} (s_{m+2l+1}^m \alpha_l^m) p_l^m(\mu_j) \quad (j = 1, 2, \ldots, J/2), \tag{32}$$

$$g_j^m = (g_j^m)_s + (g_j^m)_a, \quad g_{J-j+1}^m = (g_j^m)_s - (g_j^m)_a \quad (j = 1, 2, \ldots, J/2), \tag{33}$$

because $\mu_{J+1-j} = -\mu_j$. Here, $(g_j^m)_s$ and $(g_j^m)_a$ are the symmetric and the antisymmetric parts of g_j^m, respectively. In computing (31) and (32), the coefficients $(t_l^m \alpha_l^m)$ and $(s_{m+2l+1}^m \alpha_l^m)$ should be computed prior to the summation. The computational complexity for this change of coefficients is only $O(M^2)$, which is negligible compared to that of the transform itself $(O(M^3))$. Furthermore, (31) and (32) can be computed simultaneously, because $p_l^m(\mu_j)$ is shared between them.

The forward transform (12) can be computed in a similar way. That is, for each m $(m = 0, 1, \ldots, M)$,

$$(\widetilde{g_j^m})_s = \frac{1}{2} w_j (g_j^m + g_{J-j+1}^m), \quad (\widetilde{g_j^m})_a = \frac{1}{2} w_j \mu_j (g_j^m - g_{J-j+1}^m) \quad (j = 1, 2, \ldots, J/2), \tag{34}$$

$$s_{m+2l+1}^m = \alpha_l^m \sum_{j=1}^{J/2} (\widetilde{g_j^m})_a p_l^m(\mu_j) \quad (l = 0, 1, [\tfrac{M-m-1}{2}]), \tag{35}$$

$$u_l^m = \alpha_l^m \sum_{j=1}^{J/2} (\widetilde{g_j^m})_s p_l^m(\mu_j) \quad (l = 0, 1, [\tfrac{M-m}{2}]), \tag{36}$$

$$s_{m+2l}^m = \begin{cases} u_l^m \epsilon_{m+2l+1}^m + u_{l-1}^m \epsilon_{m+2l}^m & (l = 1, 2, \ldots, [\tfrac{M-m}{2}]) \\ u_l^m \epsilon_{m+2l+1}^m & (l = 0). \end{cases} \tag{37}$$

As described above, the computational procedure of the backward and the forward associated Legendre function transforms with the new recurrence formula is completed. The coefficients α_l^m $(l = 0, 1, \ldots)$ can be computed recursively by using (21) from the starting

point given by (22). To compute $p_l^m(\mu)$ $(l = 0, 1, \ldots)$ by using (26), the starting points $p_0^m(\mu)$ and $p_1^m(\mu)$ must be given. From definition (3), the following holds:

$$P_m^m(\mu) = \frac{\sqrt{(2m+1)!}}{2^m m!}(1-\mu^2)^{\frac{m}{2}},\ P_{m+1}^m(\mu) = \sqrt{2m+3}\mu P_m^m(\mu). \tag{38}$$

Therefore, by considering (17) and (22), $p_0^m(\mu)$ is given as

$$p_0^m(\mu) = \frac{1}{\alpha_0^m}\frac{P_{m+1}^m(\mu)}{\mu} = \epsilon_{m+1}^m \frac{P_{m+1}^m(\mu)}{\mu} = P_m^m(\mu). \tag{39}$$

Note that the value of α_0^m is chosen in (22) to simplify the formula (39), as shown above. By considering that (26) holds even when $l = 0$ by setting $p_{-1}^m(\mu) = 0$, $p_1^m(\mu)$ is given as

$$\begin{aligned} p_1^m(\mu) &= \left[(\alpha_0^m)^2\mu^2 - (\alpha_0^m)^2\left\{(\epsilon_{m+2}^m)^2 + (\epsilon_{m+1}^m)^2\right\}\right] p_0^m(\mu) \\ &= (2m+3)\left(\mu^2 - \frac{3}{2m+5}\right) p_0^m(\mu). \end{aligned} \tag{40}$$

4. Maintaining accuracy

As described in Enomoto (2015), using the traditional recurrence formula (13) naively leads to an accuracy problem when the truncation wavenumber is large, such as $M \gtrapprox 1000$. This is because the absolute values of the starting points for (13), $P_m^m(\mu)$ and $P_{m+1}^m(\mu)$ given in (38), become too small to be expressed in finite digit precision arithmetic and this causes an underflow when m is large and $|\mu| \approx 1$. This problem also appears even when the new recurrence formula (26) is used. To overcome this problem, we adopt the following strategy, while there exist other strategies, e.g. "enhanced exponent" approach (Reinecke, 2011).

First, we design the transform algorithms to require on-the-fly computation of $p_l^m(\mu_j)$ only when underflow does not occur. In (31), (32), (35), and (36), the computations for $l = 2\gamma$ and $l = 2\gamma + 1$ $(\gamma = 0, 1, \ldots)$ are paired with each other so that $p_{2\gamma+2}^m(\mu_j)$ and $p_{2\gamma+3}^m(\mu_j)$ for the next step can be computed from $p_{2\gamma}^m(\mu_j)$ and $p_{2\gamma+1}^m(\mu_j)$ by using (26). In

this procedure, we introduce j_γ^m ($\gamma = 0, 1, \ldots$) so that $j_\gamma^m \leq J/2$ is the maximum integer that satisfies the following:

$$|p_{2\gamma'}^m(\mu_j)| < \xi \quad \text{and} \quad |p_{2\gamma'+1}^m(\mu_j)| < \xi \quad (j = 1, 2, \ldots, j_\gamma^m - 1;\ \gamma' = 0, 1, \ldots, \gamma). \tag{41}$$

Here, $\xi > 0$ is a prescribed threshold value, which we set as $\xi = 10^{-20}$ with a margin in IEEE754 double precision arithmetic to determine which operations can be omitted. That is, for a γ, computations that include $p_{2\gamma}^m(\mu_j)$ or $p_{2\gamma+1}^m(\mu_j)$ ($j < j_\gamma^m$) are omitted. For example, in (36), the computation for a γ is done practically as

$$u_{2\gamma}^m = \alpha_{2\gamma}^m \sum_{j=j_\gamma^m}^{J/2} (\widetilde{g_j^m})_s p_{2\gamma}^m(\mu_j), \quad u_{2\gamma+1}^m = \alpha_{2\gamma+1}^m \sum_{j=j_\gamma^m}^{J/2} (\widetilde{g_j^m})_s p_{2\gamma+1}^m(\mu_j), \tag{42}$$

and the computation of $p_{2\gamma+2}^m(\mu_j)$ and $p_{2\gamma+3}^m(\mu_j)$ by using (26) is done only for $j = j_\gamma^m, \ldots, J/2$. If $j_{\gamma+1}^m \leq j_\gamma^m - 1$, the values of $p_{2\gamma+2}^m(\mu_j)$ and $p_{2\gamma+3}^m(\mu_j)$ ($j = j_{\gamma+1}^m, \ldots, j_\gamma^m - 1$) are also required for the next-step computation. These values are computed and stored in a memory area in advance. Hence, the values of $p_{2\gamma}^m(\mu_j)$ and $p_{2\gamma+1}^m(\mu_j)$ ($\gamma = 0, 1, \ldots$) should be stored in memory only for $j = j_\gamma^m, \ldots, j_{\gamma-1}^m - 1$ if $j_\gamma^m \leq j_{\gamma-1}^m - 1$. For the case of $\gamma = 0$, we define $j_{-1}^m = J/2 + 1$ for convenience. The memory area required to store these values is only of $O(J)$ for each m. Furthermore, the absolute values of these starting points for the on-the-fly computation are large enough to not cause the underflow, due to the design described above. Note that the omission of the computations near the pole described above leads to reduction of the computations required for the spherical harmonic transform and such implementations were also proposed in previous works, such as Juang (2004) and Schaeffer (2013). This kind of idea to reduce the computations corresponding to the polar regions dates back to Hortal and Simmons (1991), where the reduced Gaussian grid proposed by Kurihara (1965) was used to reduce the computational cost for time-integration of spherical spectral models.

Second, to avoid the underflow in the computation of the starting points defined above,

we use the following procedure. From (38), (39) and (40),

$$p_0^m(\mu_j) = \frac{\sqrt{(2m+1)!}}{2^m m!}(1-\mu_j^2)^{\frac{m}{2}}, \quad p_1^m(\mu_j) = (2m+3)\left(\mu_j^2 - \frac{3}{2m+5}\right)p_0^m(\mu_j). \tag{43}$$

When m is large and $|\mu_j| \approx 1$, the absolute value of $p_0^m(\mu_j)$ becomes very small, and so leads to the underflow in the finite digit precision arithmetic. Instead, we introduce $q_0^m(\mu_j)$ and $q_1^m(\mu_j)$ as

$$q_0^m(\mu_j) = 1, \quad q_1^m(\mu_j) = (2m+3)\left(\mu_j^2 - \frac{3}{2m+5}\right). \tag{44}$$

Then, $q_{2\gamma}^m(\mu_j)$ and $q_{2\gamma+1}^m(\mu_j)$ $(\gamma = 1, 2, \ldots)$ are computed based on the recurrence formula (26) as follows:

$$\begin{aligned}
\tilde{q}_{2\gamma}^m(\mu_j) &= (a_{2\gamma-1}^m \mu_j^2 + b_{2\gamma-1}^m) q_{2\gamma-1}^m(\mu_j) + q_{2\gamma-2}^m(\mu_j), \\
\tilde{q}_{2\gamma+1}^m(\mu_j) &= (a_{2\gamma}^m \mu_j^2 + b_{2\gamma}^m) \tilde{q}_{2\gamma}^m(\mu_j) + q_{2\gamma-1}^m(\mu_j), \\
q_{2\gamma}^m(\mu_j) &= \beta_\gamma^m \tilde{q}_{2\gamma}^m(\mu_j), \\
q_{2\gamma+1}^m(\mu_j) &= \beta_\gamma^m \tilde{q}_{2\gamma+1}^m(\mu_j).
\end{aligned} \tag{45}$$

To avoid the overflow in the computation of (45)e, β_γ^m $(\gamma = 1, 2, \ldots)$ is a scaling factor introduced as follows: :

$$\beta_\gamma^m = \begin{cases} \frac{1}{\eta} & (|\tilde{q}_{2\gamma}^m(\mu_j)| > \eta \quad \text{or} \quad |\tilde{q}_{2\gamma+1}^m(\mu_j)| > \eta) \\ 1 & (\text{else}). \end{cases} \tag{46}$$

In addition, we define $\beta_0^m = 1$ for convenience. We set $\eta = 10^{270}$ with a margin in IEEE754 double precision arithmetic. Because $\log(p_0^m(\mu_j))$ can be computed without the risk of underflow as

$$\log(p_0^m(\mu_j)) = \frac{1}{2}\sum_{m'=1}^{m} \log\frac{2m'+1}{2m'} + \frac{m}{2}\log(1-\mu_j^2), \tag{47}$$

the values of $p_{2\gamma}^m(\mu_j)$ and $p_{2\gamma+1}^m(\mu_j)$ $(\gamma = 0, 1, \ldots)$ are obtained as

$$\begin{aligned}
p_{2\gamma}^m(\mu_j) &= \mathrm{sgn}(q_{2\gamma}^m(\mu_j)) \exp\left\{\log|q_{2\gamma}^m(\mu_j)| - \sum_{\gamma'=0}^{\gamma}\log\beta_{\gamma'}^m + \log(p_0^m(\mu_j))\right\}, \\
p_{2\gamma+1}^m(\mu_j) &= \mathrm{sgn}(q_{2\gamma+1}^m(\mu_j)) \exp\left\{\log|q_{2\gamma+1}^m(\mu_j)| - \sum_{\gamma'=0}^{\gamma}\log\beta_{\gamma'}^m + \log(p_0^m(\mu_j))\right\}.
\end{aligned} \tag{48}$$

Here, sgn( ) is the sign function, and $\sum_{\gamma'=0}^{\gamma} \log \beta_{\gamma'}^m$ should be evaluated recursively as

$$\sum_{\gamma'=0}^{\gamma} \log \beta_{\gamma'}^m = \log \beta_{\gamma}^m + \sum_{\gamma'=0}^{\gamma-1} \log \beta_{\gamma'}^m. \tag{49}$$

Although the use of the logarithmic and the exponential functions might seem costly in the computations (48), their use is not problematic because these computations must be done only once in advance to provide the starting points for the on-the-fly recurrence.

Third, while it is not directly related to avoiding the underflow, the computations that should be done in advance but are not required to be done on-the-fly can be done in a higher precision arithmetic. For example, computations of α_l^m are done with IEEE754 quadruple precision arithmetic to keep the accuracy as high as possible. The Gaussian nodes and the Gaussian weights are also computed with IEEE754 quadruple precision arithmetic by using a simple Newtonian iteration in μ and by using (8), respectively, which provides sufficient precision for the spherical harmonic transform within IEEE754 double precision arithmetic.

5. Implementation and optimization

Based on the new recurrence formula proposed in the previous sections, we provide an implementation of the spherical harmonic transform, (6) and (7), as Fortran77 subroutines in open-source software ISPACK (Ishioka, 2017). In the implementation, several subroutines are written in an assembly language to fully utilize not only FMA operations but also single instruction multiple data (SIMD) operations in Intel CPUs. That is, in the computations of (31), (32), (35) and (36), the loops for suffix j are implemented so that SIMD operations are used. FFT is done by using an originally developed compact FFT library, which shows a comparable performance to FFTW. Furthermore, to utilize recent many-core CPUs, the backward and the forward associated Legendre function

transforms for $m = 0, 1, \ldots, M$ are computed in parallel by using OpenMP and specifying the dynamic scheduling option.

6. Speed and accuracy comparison

To evaluate the performance of the numerical library ISPACK, we compare its speed and accuracy with those of SHTns (Schaeffer, 2017), which was developed on the basis of Schaeffer (2013). The comparison methodology, proposed in Schaeffer (2013), is as follows. In the spherical harmonic transform (6) and (7), we first set the value of each of the real and the imaginary parts of each s_n^m $(n = 0, 1, \ldots, M; m = 0, 1, \ldots, n)$ to a uniform random number in $[-1, 1]$. Next, we compute the backward transform (6). Then, the forward transform (7) is computed from $f(\lambda_k, \mu_j)$ and we write the obtained result as $(s_n^m)'$ because the input of the backward transform is not completely equal to the output of the forward transform in finite digit precision arithmetic. The speed of each of the libraries is evaluated by measuring the elapsed time required for each transform. The accuracy of each of the libraries is also evaluated by calculating the L_∞-error and the L_2-error, which are defined, respectively, as

$$\epsilon_{\max} = \max_{n,m} |(s_n^m)' - s_n^m| \tag{50}$$

and

$$\epsilon_{\mathrm{rms}} = \sqrt{\frac{2}{(M+1)(M+2)} \sum_{n=0}^{M} \sum_{m=0}^{n} |(s_n^m)' - s_n^m|^2}. \tag{51}$$

First, we compare the speed. The computational platform is a Linux (Debian 9.1) server that has two Xeon E5-2699v4 CPUs. Each of the CPUs has 22 cores and the total number of cores of the server is 44. The Fortran compiler and the compiling option for ISPACK are "gfortran 6.3.0" and "-O3 -march=native -fopenmp", respectively. The C compiler and the compiling option for SHTns are "gcc 6.3.0" and "-

O3 -march=native -ffast-math", respectively. The tested truncation wavenumbers are $M = 1023, 2047, 4095, 8191$, and 16383. The numbers of the longitudinal and the latitudinal nodes are set as $I = 2(M + 1)$ and $J = M + 1$, respectively. Table 1 shows the comparison of the elapsed times by the number of OpenMP threads set as 44. While the elapsed times increase as M increases approximately in proportion to M^3 for both SHTns and ISPACK, the elapsed times for ISPACK are shorter than those for SHTns in each case of M for both the forward and the backward transforms. In particular, the superiority of ISPACK to SHTns in elapsed times is more significant in larger M cases. The superiority is thought to originate from the reduction of FMA operations for the recurrence computations in ISPACK, because SHTns adopts a similar on-the-fly computation strategy of the associated Legendre functions as is done in ISPACK, but the recurrence formula used is the standard one shown as (13). Furthermore, the elapsed time for ISPACK is shorter for the forward transform than that for the backward transform for each M, although the computational complexity is the same between the forward and the backward transforms. The difference is due to the implementation of ISPACK, in which the memory access for the forward transform is less than that for the backward transform.

Second, we compare the accuracy. Table 2 compares the L_∞-error and the L_2-error for SHTns with those for ISPACK for $M = 1023, 2047, 4095, 8191$, and 16383. Although these error estimates depend on the chosen random numbers and they become large as M increases, the L_∞-error and the L_2-error for ISPACK are smaller than those for SHTns for each M. This is thought to be due to the reduction of the number of computations required for the recurrence formula adopted in ISPACK.

Table 1

Table 2

7. Summary and discussion

In the present paper, we presented a new recurrence formula to calculate the associated Legendre functions to efficiently compute the spherical harmonic transform. The new recurrence formula was derived to take advantage of the fused multiply-add operation implemented on modern computers. We compared the computational speed and the accuracy of a numerical code, ISPACK, which is based on the new recurrence formula for the spherical harmonic transform, with those of another numerical code, SHTns, which was developed based on Schaeffer (2013) and showed that ISPACK was superior to SHTns in both speed and accuracy. We did not compare ISPACK with any fast algorithms. However, Schaeffer (2013), SHTns was compared with a fast algorithm (Healy, et al., 2003) and SHTns was shown to be several times faster than the fast algorithm. Furthermore, there it was shown that SHTns was about four times faster than libpsht (Reinecke, 2011) at $M = 4095$ while Seljebotn (2012) showed that his newer fast algorithm, Wavemoth, was about six times faster than libpsht in the case corresponding to $M = 4095$. On the other hand, as shown in Table 1, ISPACK is about 1.5 times faster than SHTns at $M = 4095$, considering the average elapsed time for the backward and forward transforms. Therefore, ISPACK is estimated to have comparable speed as Wavemoth at $M = 4095$ although this estimate is a rough one because the computing platforms used by these papers are all different.

A faster numerical code for the spherical harmonic transform is desired not only for constructing global atmospheric general circulation models using a spherical spectral method, as mentioned in Section 1, but also for analysis of global data even when the data is generated by a finite-difference model or by observation. This is because the spherical harmonics are eigenfunctions of the horizontal Laplacian on a sphere and the spherical harmonic expansions are useful for analyzing global atmospheric dynamics.

Hence, we believe that the new recurrence formula proposed in the present study will contribute to not only constructing a global spectral model but also a global atmospheric data analysis. In particular, when the horizontal resolution of global data is very fine, an on-the-fly computation strategy of associated Legendre functions is inevitable because of the limitations of computer memory. Thus, the new recurrence formula has the potential to become a fundamental technique. Furthermore, owing to the less memory usage, the on-the-fly computation strategy combined with the proposed new recurrence formula could enable use of a higher resolution global atmospheric general circulation model or a larger ensemble size on a wide range of machines from workstations to supercomputer systems.

Acknowledgments

We thank two anonymous reviewers for their helpful comments. This work was supported by JSPS KAKENHI (S) Grant Number 24224011, JSPS KAKENHI (A) Grant Number 17H01159, and MEXT Post-K Exploratory Challenge 3.

References

ECMWF, 2017: OpenIFS. https://software.ecmwf.int/wiki/display/OIFS/OpenIFS+Home

Enomoto, T., 2015: Comparison of computational methods of associated Legendre functions. *SOLA*, **11**, 144–149.

Frigo, M. and S. G. Johnson, 2005: The Design and Implementation of FFTW3. *Proceedings of the IEEE*, **93**, 216–231.

Healy, D. M., Rockmore, D. N., Kostelec, P. J., and Moore, S., 2003: FFTs for the 2-sphere—improvements and variations. *J. Fourier analysis and applications*, **9**, 341–385.

Hortal, M. and Simmons, A. J., 1991: Use of reduced Gaussian grids in spectral models. *Mon. Wea. Rev.*, **119**, 1057–1074.

Ishioka, K., 2017: ispack-2.1.3. http://www.gfd-dennou.org/arch/ishioka/ispack/

Ishioka, K., Yamada, M., Hayashi, Y.-Y., and Yoden, S., 2000: Technical approach for the design of a high-resolution spectral model on a sphere: Application to decaying turbulence. *Nonlinear Processes in Geophysics*, **7**, 105–110.

Juang, H. M. H., 2004: A reduced spectral transform for the NCEP seasonal forecast global spectral atmospheric model. *Mon. Wea. Rev.*, **132**, 1019–1035.

Kurihara, Y.,1965: On the use of implicit and iterative methods for the time integration of the wave equation. *Mon. Wea. Rev.*, **93**, 33–46.

Orszag, S. A., 1970: Transform method for the calculation of vector-coupled sums: Application to the spectral form of the vorticity equation. *J. Atmos. Sci.*, **27**, 890–895.

Reinecke, M., 2011: Libpsht – algorithms for efficient spherical harmonic transforms. *Astronomy & Astrophysics*, **526**, A108.

Reinecke, M. K. Dolag, R. Hell, M. Bartelmann, and T. A. Enßlin, 2006: A simulation pipeline for the Planck mission. *Astronomy & Astrophysics*, **445**, 373.

Rivier, L., R. Loft, and L. M. Polvani, 2002: An efficient spectral dynamical core for distributed memory computers. *Mon. Wea. Rev.* **130**, 1384–1396.

Schaeffer, N., 2013: Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations. *Geochemistry, Geophysics, Geosystems*, **14**, 751–758.

Schaeffer, N., 2017: SHTns-2.8. https://bitbucket.org/nschaeff/shtns/

Seljebotn, D. S., 2012: Wavemoth–fast spherical harmonic transforms by butterfly matrix compression. *The Astrophysical Journal Supplement Series*, **199**, 5.

Takahashi, Y. O., H. Kashimura, S. Takehiro, M. Ishiwatari, S. Noda, M. Odaka, T. Horinouchi, Y. Morikawa, Y.-Y. Hayashi, DCPAM Development Group, 2016: DCPAM: planetary atmosphere model. http://www.gfd-dennou.org/library/dcpam/, GFD Dennou Club.

Wicht, J. and A. Tilgner, 2010: Theory and Modeling of Planetary Dynamos. *Space Sci. Rev.*, **152**, 501–542.

List of Tables

Table 1. Comparison of the elapsed times (s) for backward (bwd) and forward (fwd) transforms between SHTns and ISPACK in the cases of $M = 1023, 2047, 4095, 8191,$ and 16383.

	$M = 1023$		$M = 2047$		$M = 4095$		$M = 8191$		$M = 16383$	
	bwd	fwd	bwd	fwd	bwd	fwd	bwd	fwd	bwd	fwd
SHTns	0.0022	0.0024	0.015	0.015	0.11	0.12	0.87	0.94	7.0	7.4
ISPACK	0.0020	0.0015	0.013	0.010	0.087	0.065	0.63	0.45	4.8	3.4

Table 2. Comparison of the L_∞ errors ($\epsilon_{\max}$) and the L_2 errors (ϵ_{rms}) between SHTns and ISPACK in the cases of $M = 1023, 2047, 4095, 8191$, and 16383.

		$M = 1023$	$M = 2047$	$M = 4095$	$M = 8191$	$M = 16383$
$\epsilon_{\max}$	SHTns	7.9×10^{-13}	3.4×10^{-12}	9.4×10^{-12}	5.5×10^{-11}	6.4×10^{-11}
	ISPACK	6.8×10^{-13}	1.2×10^{-12}	5.5×10^{-12}	1.6×10^{-11}	3.9×10^{-11}
ϵ_{rms}	SHTns	5.4×10^{-14}	1.3×10^{-13}	2.6×10^{-13}	7.0×10^{-13}	1.0×10^{-12}
	ISPACK	4.6×10^{-14}	9.4×10^{-14}	2.0×10^{-13}	4.5×10^{-13}	8.3×10^{-13}