DRAFT

GSL Technical Report #1

GSL-TR-001-20220827

Implementation of associated Legendre functions in GSL

Patrick Alken

August 27, 2022

Contents

1 Introduction

This document provides details of the implementation of the associated Legendre functions (ALFs) in the GNU Scientific Library (GSL) [Galassi et al, 2009]. We start from the unnormalized ALFs with integer degree and order,

$$P_l^m(x) = (-1)^m \left(1 - x^2\right)^{m/2} \frac{d^m}{dx^m} P_l(x), \qquad 0 \le m \le l \tag{1}$$

where $P_l(x)$ is the Legendre polynomial of degree l and m is called the order. These functions are defined on the interval $x \in [-1, 1]$. The factor $(-1)^m$ is known as the Condon-Shortley phase factor, and is omitted by some authors. GSL provides the option to include or omit this factor. When m is even, the $P_l^m(x)$ are polynomials of degree l and some authors refer to these functions as "associated Legendre polynomials." However they are not polynomials when m is odd.

The ALFs can be extended to negative orders $-l \le m < 0$ through the use of the Rodrigues formula. We simply state the result here,

$$P_l^{-m} = (-1)^m \frac{(l-m)!}{(l+m)!} P_l^m(x), \qquad 0 \le m \le l \tag{2}$$

We note that the factor $(-1)^m$ appearing in Eq. (2) is not the Condon-Shortley phase, and appears regardless of whether the Condon-Shortley phase factor is included in the definition of the ALFs. From a computational standpoint, GSL calculates only ALFs with $m \ge 0$, as the negative order functions can be readily determined from Eq. (2).

2 Normalization

The unnormalized functions defined in Eq. (1) can grow quite large even for modest degrees l, and so many practical applications use instead normalized functions, defined as

$$T_l^m(x) = A_l^m \sqrt{\frac{(l-m)!}{(l+m)!}} P_l^m(x) \tag{3}$$

where A_l^m depends on normalization convention. There are several common conventions which appear in the literature. The following sections describe the conventions implemented in GSL.

2.1 Schmidt quasi-normalization

The Schmidt quasi-normalized ALFs $S_l^m(x)$ are defined with

$$A_l^m = \begin{cases} 1, & m = 0 \\ \sqrt{2}, & m \neq 0 \end{cases} \tag{4}$$

so that

$$S_l^m(x) = \begin{cases} P_l^0(x), & m = 0 \\ \sqrt{2\frac{(l-m)!}{(l+m)!}} P_l^m(x), & m \neq 0 \end{cases} \tag{5}$$

Schmidt quasi-normalized ALFs are often used in the definition of "real-valued" spherical harmonics, namely

$$\hat{S}_l^m(\theta,\phi) = \begin{cases} \cos(m\phi) S_l^m(\cos\theta), & m \geq 0 \\ \sin(|m|\phi) S_l^{|m|}(\cos\theta), & m < 0 \end{cases} \tag{6}$$

These real-valued spherical harmonics have the convenient property that

$$\int \hat{S}_l^m(\theta,\phi)\hat{S}_{l'}^{m'}(\theta,\phi)d\Omega = \frac{4\pi}{2l+1}\delta_{mm'}\delta_{ll'} \tag{7}$$

Winch [2005] provides a detailed exposition of Schmidt quasi-normalized ALFs and real-valued spherical harmonics. The Schmidt quasi-normalized ALFs themselves obey the normalization condition,

$$\int_{-1}^{1} S_l^m(x) S_{l'}^m(x) dx = \begin{cases} \frac{2}{2l+1}\delta_{ll'}, & m = 0 \\ \frac{4}{2l+1}\delta_{ll'}, & m \neq 0 \end{cases} \tag{8}$$

2.2 Spherical harmonic normalization

The spherical harmonic normalized ALFs $Y_l^m(x)$ are defined with

$$A_l^m = \sqrt{\frac{2l+1}{4\pi}} \tag{9}$$

so that

$$Y_l^m(x) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(x) \tag{10}$$

These functions are suitable for use with the usual fully orthogonal complex-valued spherical harmonics,

$$\hat{Y}_l^m(\theta,\phi) = Y_l^m(\cos\theta)e^{im\phi} \tag{11}$$

which satisfy

$$\int \hat{Y}_l^m(\theta,\phi)\hat{Y}_{l'}^{m'*}(\theta,\phi)d\Omega = \delta_{mm'}\delta_{ll'} \tag{12}$$

The spherical harmonic normalized ALFs satisfy the following normalization condition,

$$\int_{-1}^{1} Y_l^m(x) Y_{l'}^m(x) dx = \frac{\delta_{ll'}}{2\pi} \tag{13}$$

2.3 Full normalization

The fully normalized ALFs $N_l^m(x)$ are defined with

$$A_l^m = \sqrt{l + \frac{1}{2}} \tag{14}$$

so that

$$N_l^m(x) = \sqrt{\left(l+\frac{1}{2}\right)\frac{(l-m)!}{(l+m)!}} P_l^m(x) \tag{15}$$

These functions satisfy the condition

$$\int_{-1}^{1} N_l^m(x) N_{l'}^m(x) dx = \delta_{ll'} \tag{16}$$

2.4 4π normalization

The 4π normalized ALFs $R_l^m(x)$ are defined with

$$A_l^m = \begin{cases} \sqrt{2l+1}, & m = 0 \\ \sqrt{2(2l+1)}, & m \neq 0 \end{cases} \tag{17}$$

so that

$$R_l^m(x) = \sqrt{(2-\delta_{m0})\,(2l+1)\,\frac{(l-m)!}{(l+m)!}}\,P_l^m(x) \tag{18}$$

3 Recurrence Relations

The unnormalized ALFs satisfy the following recurrence relations, which form the basis of the algorithm used to compute them,

$$(l-m)\,P_l^m(x) = (2l-1)\,xP_{l-1}^m(x) - (l+m-1)\,P_{l-2}^m(x), \qquad l \geq m+2 \tag{19}$$

$$P_{l+1}^l(x) = (2l+1)\,xP_l^l(x), \qquad l \geq 0 \tag{20}$$

$$P_l^l(x) = \eta_{CS}\,(2l-1)\,uP_{l-1}^{l-1}(x), \qquad l \geq 1 \tag{21}$$

where $u = \sqrt{1-x^2}$ and the parameter η_{CS} is defined as,

$$\eta_{CS} = -1, \qquad \text{Condon-Shortley phase included} \tag{22}$$

$$\eta_{CS} = +1, \qquad \text{Condon-Shortley phase omitted} \tag{23}$$

In terms of an angle $\theta \in [0, \pi]$, the parameters x, u can be assigned as follows,

$$x = \cos\theta \tag{24}$$

$$u = \sin\theta \tag{25}$$

Applying these recurrence relations to a general ALF $T_l^m(x)$ as defined in Eq. (3) yields the following,

$$T_l^m(x) = a_l^m x T_{l-1}^m(x) + b_l^m T_{l-2}^m(x), \qquad l \geq m+2 \tag{26}$$

$$T_{l+1}^l(x) = c_l x T_l^l(x), \qquad l \geq 0 \tag{27}$$

$$T_l^l(x) = d_l u T_{l-1}^{l-1}(x), \qquad l \geq 1 \tag{28}$$

$$a_l^m = \frac{2l-1}{\sqrt{(l+m)(l-m)}}\frac{A_l^m}{A_{l-1}^m}, \qquad l \geq m+2 \tag{29}$$

$$b_l^m = -\sqrt{\frac{(l+m-1)(l-m-1)}{(l+m)(l-m)}}\frac{A_l^m}{A_{l-2}^m}, \qquad l \geq m+2 \tag{30}$$

$$c_l = \sqrt{2l+1}\frac{A_{l+1}^l}{A_l^l}, \qquad l \geq 0 \tag{31}$$

$$d_l = \eta_{CS}\sqrt{\frac{2l-1}{2l}}\frac{A_l^l}{A_{l-1}^{l-1}}, \qquad l \geq 1 \tag{32}$$

Here, $T_l^m(x)$ can be any of $P_l^m(x), S_l^m(x), Y_l^m(x)$, $N_l^m(x)$, or $R_l^m(x)$. Eqs. (26) to (28) form the basis of the algorithm used by GSL to compute the ALFs for all degrees and orders. The coefficients a_l^m, b_l^m, c_l, d_l are precomputed when the user calls the function `gsl_sf_legendre_precompute` and are then used to efficiently evaluate the ALFs for any number of arguments x. Table 1 presents the coefficients for each normalized ALF.

Table 1: Coefficients of recurrence relations for different normalizations of ALFs.

Normalization	Notation	a_l^m	b_l^m	c_l	d_l
Unnormalized	$P_l^m(x)$	$\frac{2l-1}{l-m}$	$-\frac{l+m-1}{l-m}$	$2l+1$	$\eta_{CS}\,(2l-1)$
Schmidt	$S_l^m(x)$	$\frac{2l-1}{\sqrt{(l+m)(l-m)}}$	$-\sqrt{\frac{(l+m-1)(l-m-1)}{(l+m)(l-m)}}$	$\sqrt{2l+1}$	$\begin{cases} \eta_{CS} & l=1 \\ \eta_{CS}\sqrt{1-\frac{1}{2l}} & l>1 \end{cases}$
Spherical Harmonic	$Y_l^m(x)$	$\sqrt{\frac{(2l+1)(2l-1)}{(l+m)(l-m)}}$	$-\sqrt{\frac{(l+m-1)(l-m-1)(2l+1)}{(l+m)(l-m)(2l-3)}}$	$\sqrt{2l+3}$	$\eta_{CS}\sqrt{1+\frac{1}{2l}}$
Full	$N_l^m(x)$	$\sqrt{\frac{(2l+1)(2l-1)}{(l+m)(l-m)}}$	$-\sqrt{\frac{(l+m-1)(l-m-1)(2l+1)}{(l+m)(l-m)(2l-3)}}$	$\sqrt{2l+3}$	$\eta_{CS}\sqrt{1+\frac{1}{2l}}$
4π	$R_l^m(x)$	$\sqrt{\frac{(2l+1)(2l-1)}{(l+m)(l-m)}}$	$-\sqrt{\frac{(l+m-1)(l-m-1)(2l+1)}{(l+m)(l-m)(2l-3)}}$	$\sqrt{2l+3}$	$\begin{cases} \eta_{CS}\sqrt{3} & l=1 \\ \eta_{CS}\sqrt{1+\frac{1}{2l}} & l>1 \end{cases}$

We note that Eq. (28) contains an instability which could cause underflow when evaluating ALFs for large degrees near the poles where u is small, due to the recursive multiplication by u. GSL applies the method described in Holmes and Featherstone [2002] to rescale u as the recursion progresses to avoid the instability. This allows the accurate calculation of normalized ALFs up to degree and order approximately 2700 in double precision.

4 Recurrence Relations for Derivatives

Derivatives of associated Legendre functions are often required in applications, such as the spherical harmonic expansions of potential fields. Derivatives of ALFs can be computed with respect to the input argument x, but also with respect to a colatitude variable θ, where $x = \cos\theta$,

$$\frac{d^k}{dx^k} P_l^m(x) \tag{33}$$

or

$$\frac{d^k}{d\theta^k} P_l^m(\cos\theta) \tag{34}$$

The first type of derivative with respect to x can contain singularities at the end points $x = \pm 1$. For example,

$$\frac{d}{dx} P_1^1(x) = -\frac{x}{\sqrt{1-x^2}} \tag{35}$$

which is undefined at the end points. However, the alternative derivative with respect to θ is defined at the end points for all degrees and orders. Bosch [2000] derives the following

recurrence relations for any order derivative with respect to θ:

$$\frac{d^k}{d\theta^k} P_0^0(\cos\theta) = 0 \tag{36}$$

$$\frac{d^k}{d\theta^k} P_l^0(\cos\theta) = -\eta_{CS} \frac{d^{k-1}}{d\theta^{k-1}} P_l^1(\cos\theta), \qquad l \geq 1 \tag{37}$$

$$\frac{d^k}{d\theta^k} P_l^m(\cos\theta) = \frac{1}{2}\eta_{CS} \left[(l+m)(l-m+1) \frac{d^{k-1}}{d\theta^{k-1}} P_l^{m-1}(\cos\theta) - \frac{d^{k-1}}{d\theta^{k-1}} P_l^{m+1}(\cos\theta) \right], \qquad l \geq 1, m > 0 \tag{38}$$

These formulas are valid for any $k \geq 1$ and for all $\theta \in [0, \pi]$ including the end points. The corresponding formulas for a normalized ALF $T_l^m(\cos\theta)$ are

$$\frac{d^k}{d\theta^k} T_0^0(\cos\theta) = 0 \tag{39}$$

$$\frac{d^k}{d\theta^k} T_l^0(\cos\theta) = -\eta_{CS} \frac{A_l^0}{A_l^1} \sqrt{l(l+1)} \frac{d^{k-1}}{d\theta^{k-1}} T_l^1(\cos\theta), \qquad l \geq 1 \tag{40}$$

$$\frac{d^k}{d\theta^k} T_l^m(\cos\theta) = \frac{1}{2}\eta_{CS} \left[\sqrt{(l+m)(l-m+1)} \frac{A_l^m}{A_l^{m-1}} \frac{d^{k-1}}{d\theta^{k-1}} T_l^{m-1}(\cos\theta) - \sqrt{(l+m+1)(l-m)} \frac{A_l^m}{A_l^{m+1}} \frac{d^{k-1}}{d\theta^{k-1}} T_l^{m+1}(\cos\theta) \right], \qquad l \geq 1, m > 0 \tag{41}$$

These equations can be applied recursively to find any derivative order k. GSL provides routines to calculate up to the second derivative ($k = 2$). First, the ALFs are computed using the recurrence relations described in Sec. 3. The first derivatives with respect to θ are then calculated using the relations above. Optionally, the second derivatives with respect to θ are calculated by applying the above relations again to the computed first derivatives. Eqs. (40)-(41) contain coefficients which are square roots of integers. In order to optimize the calculation, these square root factors are calculated in the precomputing step.

5 Implementation Details

The recurrence relation Eq. (26) comprises the main computational effort of calculating the associated Legendre functions. The output arrays of the associated Legendre calculations in GSL version 2.7 and prior were indexed according to

$$\mathcal{I}_l(l, m) = \frac{l(l+1)}{2} + m, \qquad 0 \leq l \leq L, 0 \leq m \leq l \tag{42}$$

where L is the maximum degree and order input to the routines. This corresponds to the following memory layout,

$$m \quad \overbrace{0}^{l=0} \quad \overbrace{0 \quad 1}^{l=1} \quad \overbrace{0 \quad 1 \quad 2}^{l=2} \quad \cdots \quad \overbrace{0 \quad 1 \quad \cdots \quad L}^{l=L}$$

So inside a given l block, consecutive elements correspond to different m. However a careful analysis of Eq. (26) indicates that the recurrence proceeds by holding m fixed and increasing

l to calculate the next term. Therefore, using the $\mathcal{I}_l(l,m)$ indexing for the coefficients a_l^m and b_l^m during this recurrence would require accessing memory in non-sequential order, potentially resulting in costly cache misses. A more cache efficient scheme would organize the arrays holding the a_l^m and b_l^m coefficients so that consecutive memory locations are accessed at each step of the recurrence. To do this, GSL version 2.8 and later defines the alternate indexing scheme,

$$\mathcal{I}_m(l,m,L) = mL - \frac{m(m-1)}{2} + l, \qquad 0 \le l \le L, 0 \le m \le l \tag{43}$$

which corresponds to the memory layout

$$l \quad \overbrace{0 \quad 1 \quad 2 \quad \cdots \quad L}^{m=0} \quad \overbrace{1 \quad 2 \quad \cdots \quad L}^{m=1} \quad \overbrace{2 \quad \cdots \quad L}^{m=2} \quad \cdots \quad \overbrace{L}^{m=L}$$

This memory layout was inspired by the SHTns software package [Schaeffer, 2013]. Here we group entries according to order m, and in each m block, consecutive elements correspond to increasing l. This memory scheme is ideal for the recurrence Eq. (26) since the terms a_l^m, b_l^m will be needed during one iteration, while the terms a_{l+1}^m, b_{l+1}^m will be needed on the next. We go one step further and store the elements a_l^m, b_l^m in a single array as follows,

$$\overbrace{a_0^0 \quad b_0^0 \quad a_1^0 \quad b_1^0 \quad \cdots \quad a_L^0 \quad b_L^0}^{m=0} \quad \overbrace{a_1^1 \quad b_1^1 \quad a_2^1 \quad b_2^1 \quad \cdots \quad a_L^1 \quad b_L^1}^{m=1} \quad \cdots \quad \overbrace{a_L^L \quad b_L^L}^{m=L}$$

Therefore, on a given iteration of the recurrence, the required coefficients a_l^m, b_l^m will be stored in consecutive memory locations, allowing a cache-efficient implementation of Eq. (26).

We note that GSL v2.8 and later uses the $\mathcal{I}_m(l,m,L)$ indexing scheme by default, but provides the $\mathcal{I}_l(l,m)$ scheme for backward compatibility.

6 Verification

There are a few ways to verify the calculation of the associated Legendre functions and their derivatives.

6.1 Known identities

A useful method for testing the accuracy of high degree and order ALFs is to use the following identity for the Schmidt quasi-normalized ALFs,

$$\sum_{m=0}^{l} \left(S_l^m(x)\right)^2 = 1 \tag{44}$$

We define the following quantity for testing purposes,

$$\epsilon_l(x) = \sum_{m=0}^{l} \left(S_l^m(x)\right)^2 - 1 \tag{45}$$

Figure 1 (left) shows the value of $\epsilon_{2700}(x)$ for 2000 randomly chosen points x on $[-1,1]$. The right panel shows a heat map of $\log_{10}(\epsilon_l(x))$ as a function of degree l and input x.

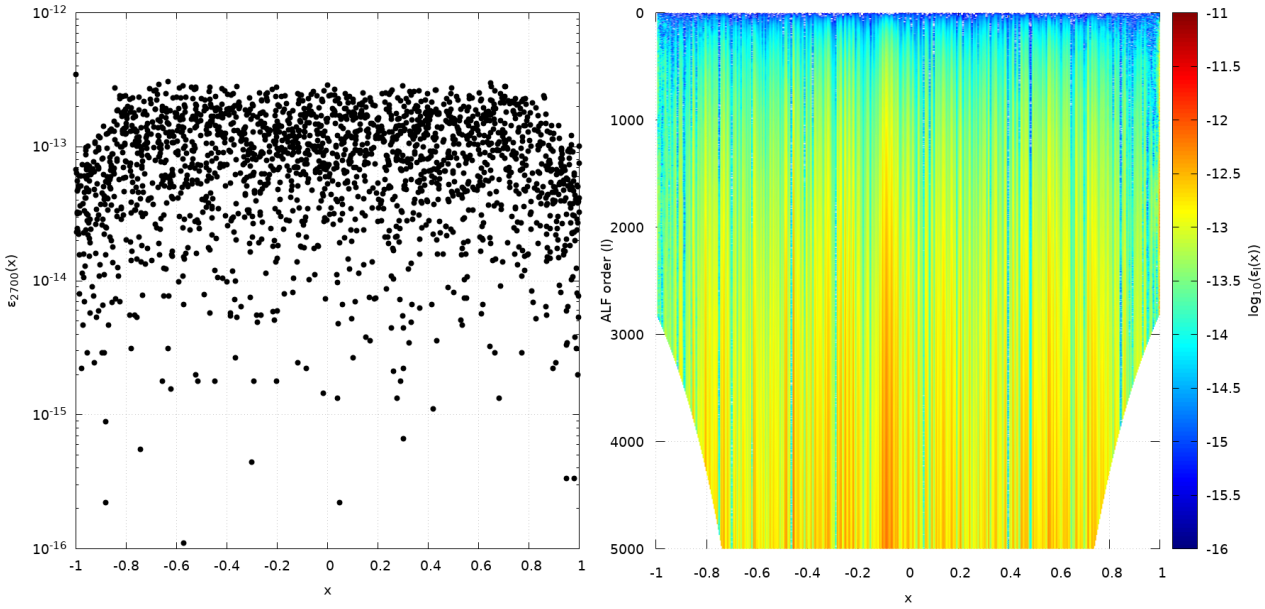

Figure 1: Left: $\epsilon_{2700}(x)$ for 2000 randomly distributed points on $[-1, 1]$. Right: Map of $\log_{10}(\epsilon_l(x))$ as a function of ALF degree l and x. Schmidt normalization is used.

6.2 Comparison with arbitrary precision libraries

Here, we compare the GSL ALF calculation against the arbitrary precision Python library, mpmath, using a working precision of 50 digits. In the case of ALFs, arbitrary precision libraries avoid the underflow problem at high orders, and can serve as a truth reference for the double precision GSL calculations. Figure 2 presents the logarithm of the relative error between GSL and mpmath, using spherical harmonic normalized ALFs up to degree and order 3000, for three different input values, $x = \cos\theta$. The left panel shows $\theta = 60°$, the middle panel $\theta = 40°$, and the right panel $\theta = 25°$. The whitespace under the black diagonal line indicates that the resulting ALF cannot be represented in double precision, and the GSL calculation underflowed. We see that the underflow problem increases as we move closer to the pole region at high degrees and orders. The blue regions indicate that GSL matches the arbitrary precision result to around 12-16 significant digits. The red portion of the right panel indicates a loss of many significant digits in the GSL result, starting around degree 2000 and order 1500 for $\theta = 25°$.

7 Benchmarks

Here, we compare the performance of the GSL implementation of ALFs against two other libraries. The first, known as SHTOOLS [Wieczorek and Meschede, 2018], provides numerous normalization conventions and uses L-major indexing in the output arrays. The second, called SHTns [Schaeffer, 2013], provides highly optimized ALFs with the spherical harmonic normalization, and uses M-major indexing in its output arrays. All benchmarks were performed on an Intel Xeon CPU E5-2620 v3 @ 2.40GHz with 64GB RAM. Figure 3 presents the result of calculating ALFs using each library interface for a fixed input point $x = -0.75$. The top panel displays the wall clock time for each implementation as a function of ALF degree up

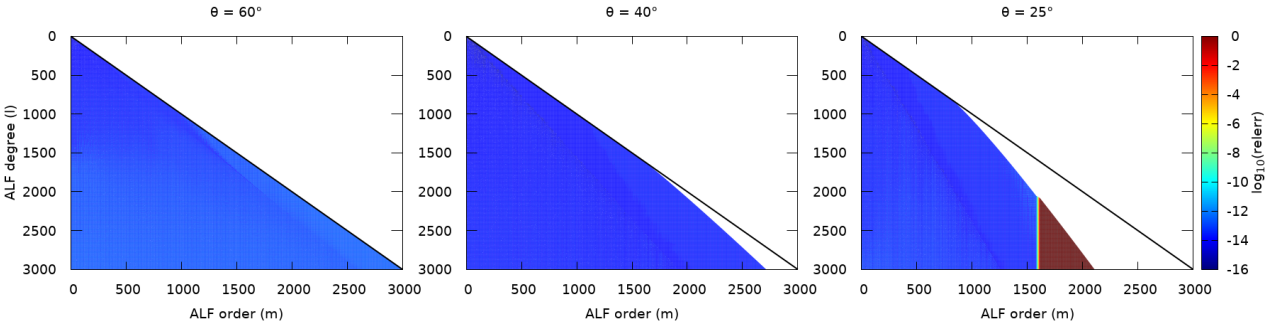

Figure 2: Log of relative error between GSL and mpmath calculations plotted versus ALF degree and order for $\theta = 60°$ (left), $\theta = 40°$ (middle), $\theta = 25°$ (right). This figure uses the spherical harmonic normalization.

to $L = 1500$. The SHTns and GSL implementions (with M-major indexing) perform best at nearly all degrees, with SHTns performing slightly faster than GSL. The bottom panel shows the wall clock time compared with SHTns (e.g. the ratio between the GSL/SHTOOLS time and SHTns). The green curve shows that GSL (M-major indexing) performs nearly as well as SHTns at all degrees up to 1500. The blue curve shows that GSL (L-major indexing) is competitive with SHTns up to about degree 1100 at which point it begins to significantly underperform. The SHTOOLS library underperforms SHTns at all degrees, which we attribute to its cache-inefficient implementation. The source code used for these benchmarks is listed in Appendix A.

8 List of Acronyms

Acronym	Meaning
ALF	Associated Legendre Function
GSL	GNU Scientific Library

Bibliography

Bosch, W., "On the computation of derivatives of Legendre functions,", *Phys. Chem. Earth*, 25, 9–11, pg. 655–659, 2000.

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., and Rossi, F., "GNU Scientific Library Reference Manual," 3rd edition, *Network Theory Ltd*, 2009.

Holmes, S., Featherstone, W., "A unified approach to the Clenshaw summation and the recursive computation of very high degree and order normalised associated Legendre functions," *Journal of Geodesy*, 279-299, 2002.

Schaeffer, N., "Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations", *Geochem. Geophys. Geosys.*, 14, 3, 751–758, doi:10.1002/ggge.20071, 2013.

Wieczorek, M. A., Meschede M., "Tools for working with spherical harmonics", *Geochem. Geophys. Geosys.*, 19, 2574–2592, doi:10.1029/2018GC007529, 2018.

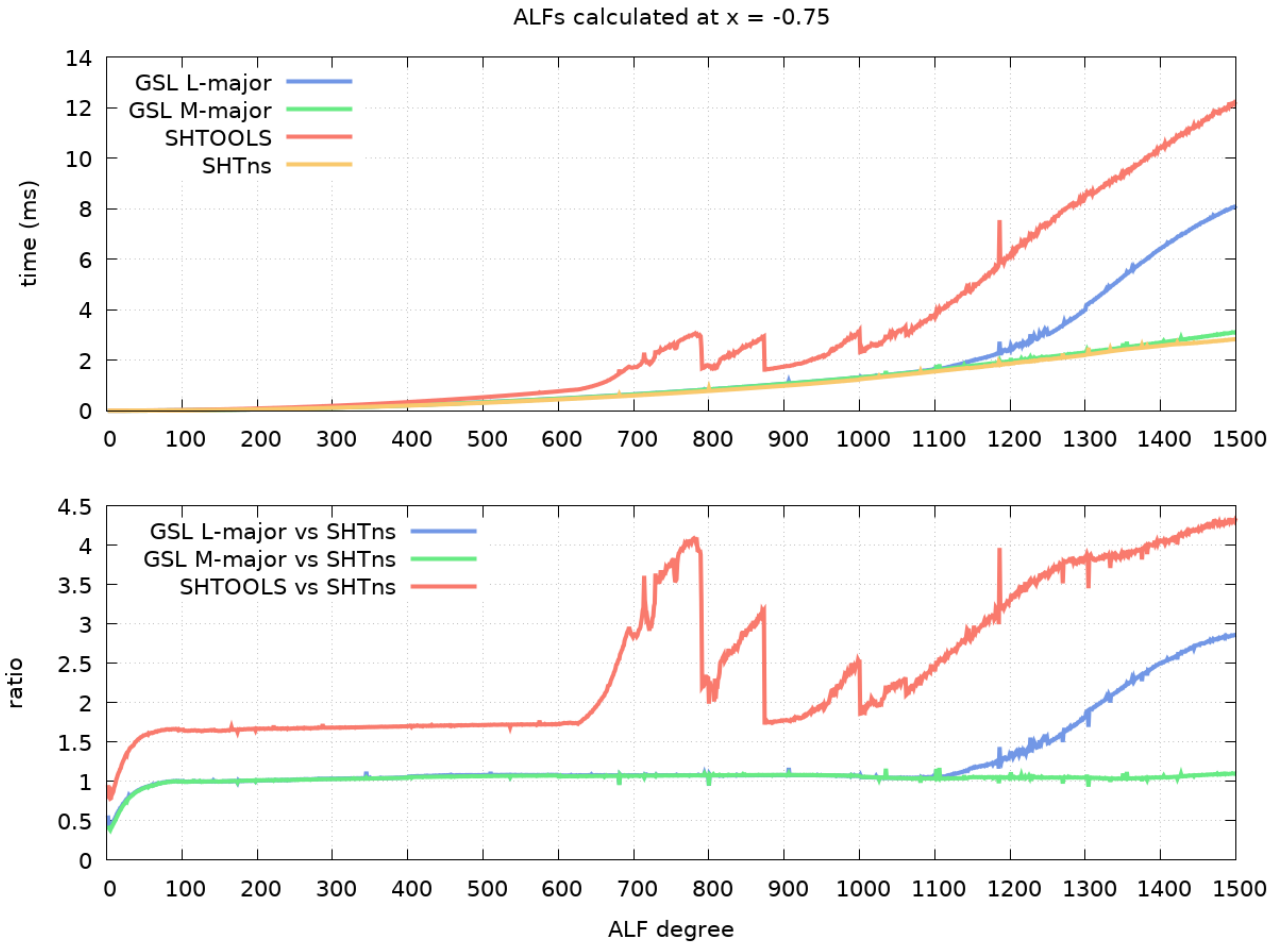

Figure 3: Top: Average time of ALF calculation for different libraries as a function of ALF degree, for the input $x = -0.75$. Bottom: Ratio of library times to the SHTns library.

Winch, D. E., Ivers, D. J., Turner, J. P. R., and Stening, R. J., "Geomagnetism and Schmidt quasi-normalization," *Geophys. J. Int.*, 160, p. 487-504, 2005.

A Source Code for Benchmarks

The following code was used to generate Figure 3. It was compiled with the command:

```
g++ -O2 -Wall -W -o bench bench.cc -lm -lgsl -lshtns -lfftw3 -lSHTOOLS -llapack -lcblas -lf77blas -lgfortran
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>

#include <gsl/gsl_math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_sf_legendre.h>
#include <gsl/gsl_vector.h>
```

```
11
12 #include <shtns.h>
13
14 #include "shtools.h"
15
16 #define TIME_DIFF(a,b) (((b).tv_sec + (b).tv_usec * 1.0e-6) - ((a).tv_sec +
       (a).tv_usec * 1.0e-6))
17
18 /* GSL */
19 double
20 proc_P_gsl(const size_t flags, const size_t lmax, const size_t n, const
       double x, double * Plm)
21 {
22   struct timeval tv0, tv1;
23   double dt;
24   size_t i;
25
26   gettimeofday(&tv0, NULL);
27
28   gsl_sf_legendre_precompute(GSL_SF_LEGENDRE_SPHARM, lmax, flags, Plm);
29
30   for (i = 0; i < n; ++i)
31     gsl_sf_legendre_arrayx(GSL_SF_LEGENDRE_SPHARM, lmax, x, Plm);
32
33   gettimeofday(&tv1, NULL);
34
35   dt = TIME_DIFF(tv0, tv1);
36
37   return dt;
38 }
39
40 /* SHTOOLS, l indexing */
41 double
42 proc_P_shtools(const size_t lmax, const size_t n, const double x, double *
       Plm)
43 {
44   struct timeval tv0, tv1;
45   size_t i;
46   int lmaxi = (int) lmax;
47   int csphase = 1;
48   int cnorm = 1;
49   int status;
50   double dt;
51
52   gettimeofday(&tv0, NULL);
53
54   for (i = 0; i < n; ++i)
55     shtools::PlmON(&Plm[0], lmaxi, x, &csphase, &cnorm, &status);
56
57   gettimeofday(&tv1, NULL);
58
59   dt = TIME_DIFF(tv0, tv1);
60
61   /* deallocate memory */
62   shtools::PlmON(&Plm[0], -1, x, &csphase, &cnorm, &status);
```

```
63
64    return dt;
65  }
66
67  double
68  proc_P_shtns(shtns_cfg shtns, const size_t lmax, const size_t n, const
        double x, double * Plm)
69  {
70    struct timeval tv0, tv1;
71    size_t i;
72    double dt;
73
74    gettimeofday(&tv0, NULL);
75
76    for (i = 0; i < n; ++i)
77      {
78        size_t idx = 0;
79        int m;
80
81        for (m = 0; m <= (int) lmax; ++m)
82          {
83            legendre_sphPlm_array(shtns, lmax, m, x, &Plm[idx]);
84            idx += lmax + 1 - m;
85          }
86      }
87
88    gettimeofday(&tv1, NULL);
89
90    dt = TIME_DIFF(tv0, tv1);
91
92    return dt;
93  }
94
95  int
96  main(int argc, char * argv[])
97  {
98    const size_t lmax = 1500;
99    const double x = -0.75;
100   size_t eval_lmin = 2;
101   size_t eval_lmax = 400;
102   size_t n = 2000;
103   const size_t plm_size = gsl_sf_legendre_array_n(lmax);
104   double * Plm = (double *) malloc(plm_size * sizeof(double));
105   shtns_cfg shtns = shtns_create((int) lmax, (int) lmax, 1, (shtns_norm) (
        sht_orthonormal | SHT_NO_CS_PHASE));
106   size_t l;
107
108   if (argc > 1)
109     eval_lmin = (size_t) atoi(argv[1]);
110   if (argc > 2)
111     eval_lmax = (size_t) atoi(argv[2]);
112   if (argc > 3)
113     n = (size_t) atoi(argv[3]);
114
115   l = 1;
```

```
  printf("Field %zu: ALF degree\n", l++);
  printf("Field %zu: GSL (l indexing) (seconds)\n", l++);
  printf("Field %zu: GSL (m indexing) (seconds)\n", l++);
  printf("Field %zu: SHTOOLS (seconds)\n", l++);
  printf("Field %zu: SHTns (seconds)\n", l++);

  for (l = eval_lmin; l <= eval_lmax; ++l)
    {
      double dt_gsll = proc_P_gsl(GSL_SF_LEGENDRE_FLG_INDEXL, l, n, x, Plm);
      double dt_gslm = proc_P_gsl(0, l, n, x, Plm);
      double dt_shtools = proc_P_shtools(l, n, x, Plm);
      double dt_shtns = proc_P_shtns(shtns, l, n, x, Plm);

      printf("%zu %.12e %.12e %.12e %.12e\n",
             l,
             dt_gsll / n,
             dt_gslm / n,
             dt_shtools / n,
             dt_shtns / n);
      fflush(stdout);
    }

  free(Plm);
  shtns_destroy(shtns);

  return 0;
}
```