NASA/TP-2016-218604

Normalization and Implementation of Three Gravitational Acceleration Models

Revision A, Final

Randy A. Eckman[1]
Aaron J. Brown[2]
Daniel R. Adamo[3]

Reviewed by Robert G. Gottlieb[4]

March 4, 2014

[1] *Aerospace engineer, NASA Johnson Space Center, Mail Code DM33*
[2] *Aerospace engineer, NASA Johnson Space Center, Mail Code DM46*
[3] *Independent astrodynamics consultant*
[4] *Technical fellow, Odyssey Space Research*

National Aeronautics and
Space Administration

Johnson Space Center
Houston, Texas 77058

June 2016

NASA/TP-2016-218604

Normalization and Implementation of Three Gravitational Acceleration Models

Revision A, Final

Randy A. Eckman[1]
Aaron J. Brown[2]
Daniel R. Adamo[3]

Reviewed by Robert G. Gottlieb[4]

March 4, 2014

[1] *Aerospace engineer, NASA Johnson Space Center, Mail Code DM33*
[2] *Aerospace engineer, NASA Johnson Space Center, Mail Code DM46*
[3] *Independent astrodynamics consultant*
[4] *Technical fellow, Odyssey Space Research*

National Aeronautics and
Space Administration

Johnson Space Center
Houston, Texas 77058

June 2016

Contents

List of Tables

THIS PAGE IS INTENTIONALLY LEFT BLANK.

List of Figures

THIS PAGE IS INTENTIONALLY LEFT BLANK.

Introduction

Unlike the uniform density spherical shell approximations of Newton, the consequence of spaceflight in the real universe is that gravitational fields are sensitive to the asphericity of their generating central bodies. The gravitational potential of an aspherical central body is typically resolved using spherical harmonic approximations. However, attempting to directly calculate the spherical harmonic approximations results in at least two singularities that must be removed to generalize the method and solve for any possible orbit, including polar orbits. Samuel Pines [1], Bill Lear [2], and Robert Gottlieb [3] developed three unique algorithms to eliminate these singularities.

This paper documents the methodical normalization of two[1] of the three known formulations for singularity-free gravitational acceleration (namely, the Lear [2] and Gottlieb [3] algorithms) and formulates a general method for defining normalization parameters used to generate normalized Legendre polynomials and Associated Legendre Functions (ALFs) for any algorithm. A treatment of the conventional formulation of the gravitational potential and acceleration is also provided, in addition to a brief overview of the philosophical differences between the three known singularity-free algorithms.

[1]The Pines algorithm (Section 3.1) has been previously normalized and thoroughly investigated by Lundberg and Schutz [4] and subsequently implemented by DeMars [5]. See Section 3.1.2.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

Chapter 1

Gravitational Potential and Acceleration

1.1 History

In 1782, Adrien-Marie Legendre authored a report entitled *Recherches sur l'attraction des sphéroïdes homogènes*. This report examined the problem of the attraction on a particle due to homogeneous spheroids in two dimensions. As part of Legendre's approach to the problem, he introduced a family of functions that eventually became known as Associated Legendre Functions (ALFs).

Twenty years after Legendre's groundbreaking work, Pierre-Simon de Laplace published Volume 3 of his magnum opus *Mécanique céleste* in 1802. In it, Laplace provides a complete treatment on the attraction of a spheroid on a particle by expanding on the work of Legendre (and, notably, providing no credit to Legendre whatsoever). By extending Legendre's approach to three dimensions, Laplace first developed and introduced the concept of spherical harmonics, which provides the foundation for modern-day gravity modeling algorithms as well as applications in many other fields of research.

1.2 Derivation

1.2.1 Conservative Forces and Potential

Gravity is a conservative field force, which has both qualitative and quantitative implications. Work done by conservative forces is path-independent. Additionally, mechanical energy E, which is the sum of kinetic energy T and potential energy V, is conserved (constant) in conservative fields, as seen in Eq. 1.1.

$$T_1 + V_1 = T_2 + V_2 = E \tag{1.1}$$

To relate these two seemingly unrelated concepts, a quantity is defined called simply the potential U. Potential is always the opposite of potential energy[1] as shown in Eq. 1.2.

$$U = -V \tag{1.2}$$

Scalar functions of position (and time for time-varying fields) that describe the potential are called potential functions. By substituting Eq. 1.2 into Eq. 1.1, we arrive at the relation shown in Eq. 1.3.

$$T_1 - U_1 = T_2 - U_2 \tag{1.3}$$

[1]It should be noted that convention is to use U and V for potential and potential energy, but not necessarily respectively. The convention used in this document represents the rough consensus of sources cited in this document, but care should be taken when reading literature to determine and distinguish which variable is used to symbolize the quantities of potential and potential energy.

Finally, by rearranging Eq. 1.3, we arrive at the unique conclusion for conservative field forces that the change in T is exactly that of a change in U per Eq. 1.4

$$\Delta T = \Delta U \tag{1.4}$$

Work is defined as the change in the kinetic energy due to a force $\vec{F}$ acting over a path with position vector $\vec{r}$, as shown in Eq. 1.5.

$$W = \Delta T = \int \vec{F} \cdot d\vec{r} \tag{1.5}$$

For a conservative force, since $\Delta T = \Delta U$, the work done by the force is also equal to the change in the potential. Substituting ΔU into Eq. 1.5 and rewriting in differential form, Eq. 1.6 arises.

$$\partial U = \vec{F} \cdot \partial \vec{r} \tag{1.6}$$

Rearranging, Eq. 1.7 thus shows that any conservative force is defined by the gradient of its potential.

$$\vec{F} = \frac{\partial U}{\partial \vec{r}} = \nabla U \tag{1.7}$$

The divergence of a vector field is a measure of the presence of sources or sinks. In the case of gravity, each particle with mass acts as a sink, giving $\nabla \cdot \vec{F} \neq 0$ at each point where matter is located. Elsewhere in space in the absence of matter at that point, $\nabla \cdot \vec{F} = 0$. Locations outside of the central body are typically the only ones considered in astrodynamic applications, and the assumption can be made that the solution to $\nabla \cdot \vec{F} = 0$ is the only one of interest in this document.

1.2.2 Properties of Potential Functions

Substituting Eq. 1.7 in the divergence of a conservative field (in empty space for gravity) results in Eq. 1.8.

$$\nabla \cdot \vec{F} = \nabla \cdot \nabla U = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = \nabla^2 U = 0 \tag{1.8}$$

Eq. 1.8 is called Laplace's equation. The symbol ∇^2 is commonly called the Laplacian operator. Solutions to Laplace's equations are harmonic functions, thus potential functions for conservative fields are harmonic functions[2]. Since harmonic functions are also analytic functions, potential functions can also be expressed as a convergent power series.

Harmonic functions, also called "harmonics," exhibit the property of superposition. Therefore, a solution to Laplace's equation can be decomposed using a set of harmonics that are convenient for the solution.

From Newton's Second Law of Motion, we know that $\vec{F}_{\text{net}} = m\vec{a}$, where m is an arbitrary mass. Combining this definition with Eq. 1.7, Eq. 1.9 arises.

$$m\vec{a} = \nabla U \tag{1.9}$$

Solving for the acceleration $\vec{a}$ and using the property of superposition for gradients, we define a new potential function u, which is the gravitational potential per unit mass as shown in Eq. 1.10.

$$\vec{a} = \nabla \left(\frac{U}{m} \right) = \nabla u \tag{1.10}$$

Using u (rather than U) for the remainder of the derivation allows us to compute the gravitational acceleration directly and independently of a satellite mass.

[2]The study of harmonic functions is called *potential theory* since many of their properties were historically discovered through the study of gravitational potential functions.

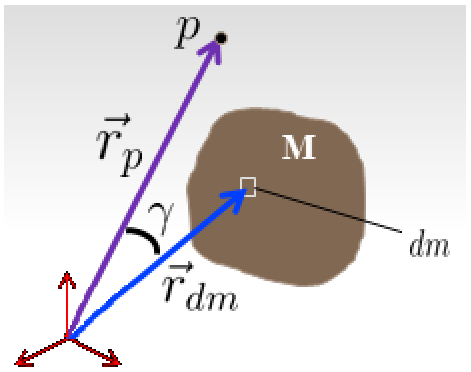

Figure 1.1: Definition of vectors, angle, and differential mass.

1.2.3 Defining the Gravitational Potential

According to Newton, the gravitational potential u at a given point p due to the point mass M is defined in Eq. 1.11, where G is the Newtonian gravitational constant and $\mu = GM$ is the gravitational parameter of the central body.

$$u = \frac{GM}{|\vec{r}_p - \vec{r}_M|} = \frac{\mu}{r} \tag{1.11}$$

The gravitational potential at point p due to a non-uniform central body can be found by integrating Eq. 1.11 over differential masses dm as shown in Eq. 1.12.

$$u = \int \frac{G\ dm}{|\vec{r}_p - \vec{r}_{dm}|} \tag{1.12}$$

Figure 1.1 illustrates the geometry of the vectors relative to an arbitrary coordinate frame outside the central body. The subtraction of the two position vectors in Eq. 1.12 can be replaced with the law of cosines using the angle γ between the two position vectors and ratio $R = r_{dm}/r_p$ according to Eq. 1.13

$$|\vec{r}_p - \vec{r}_{dm}| = \sqrt{r_p^2 + r_{dm}^2 - 2r_p r_{dm}\cos\gamma} = r_p\sqrt{1 + R^2 - 2R\cos\gamma} \tag{1.13}$$

Substituting Eq. 1.13 into Eq. 1.12 gives Eq. 1.14.

$$u = \frac{G}{r_p}\int \frac{1}{\sqrt{1 + R^2 - 2R\cos\gamma}}\, dm \tag{1.14}$$

1.2.4 Power Series Expansion

The definition of a power series is shown in Eq. 1.15.

$$f(x) = \sum_{n=0}^{\infty} a_n x^n \tag{1.15}$$

The ratio $1/\sqrt{1 + R^2 - 2R\cos\gamma}$ from Eq. 1.14 is a common form that can be expanded using a power series. Because this is a simplified expression of a potential function, the power series will be convergent. The resulting power series from expanding this ratio, derived in Appendix A, is shown in Eq. 1.16.

$$\frac{1}{\sqrt{1 + R^2 - 2R\cos\gamma}} = \sum_{n=0}^{\infty} P_n(\cos\gamma)R^n \tag{1.16}$$

The power series coefficients $P_n(\cos\gamma)$ are called the Legendre polynomials with argument $\cos\gamma$. Substituting back into Eq. 1.14, the potential is now effectively in polar coordinates of r_p and γ in Eq. 1.17 (R is a function of r_p).

$$u = \frac{G}{r_p} \int \sum_{n=0}^{\infty} P_n(\cos\gamma) R^n \; dm \tag{1.17}$$

1.2.5 Spherical Harmonic Expansion

To find acceleration in three dimensions, a potential needs to be defined in three dimensions. By defining γ as a function of two spherical angles, it can be decomposed using spherical triangles and used to redefine the potential in three dimensions. Geocentric latitude ϕ and longitude θ are typically used in this case. By substituting this decomposition into Eq. 1.17, gathering like terms, and using properties of Legendre polynomials to expand, Eq. 1.18 arises.

$$u = \frac{\mu}{r_p} \left\{ 1 - \sum_{n=1}^{\infty} \left(\frac{a_{eq}}{r_p} \right)^n \left[J_n P_n(\sin\phi) - \sum_{m=1}^{n} (C_{n,m} \cos m\theta + S_{n,m} \sin m\theta) P_{n,m}(\sin\phi) \right] \right\} \tag{1.18}$$

An alternate formulation [6, Pg. 52] shown in Eq. 1.19 can also be used to increase computational efficiency by using a common form for all three of the fundamental harmonics.

$$u = \frac{\mu}{r_p} \left\{ 1 + \sum_{n=1}^{\infty} \sum_{m=0}^{n} \left(\frac{a_{eq}}{r_p} \right)^n (C_{n,m} \cos m\theta + S_{n,m} \sin m\theta) P_{n,m}(\sin\phi) \right\} \tag{1.19}$$

The three fundamental harmonics are named after the coefficients used in the respective terms in Eq. 1.19. The coefficients and their names are described in Section 1.2.5.

Constants, Coefficients, and Nomenclature

Certain variables have specific names commonly used in the literature to describe the equations. The *degree* of a term in a sum use the subscript n while the *order* of a term uses the subscript m.

J_n, $C_{n,m}$, and $S_{n,m}$ are known in the literature as the spherical harmonic mass coefficients of the central body, which we will refer to simply as the *mass coefficients*. Certain subsets of the mass coefficients are given special names.

- $J_n = -C_{n,0}$ = zonal coefficients
- $C_{n,n}, S_{n,n}$ = sectorial coefficients ($n = m$)
- $C_{n,m}, S_{n,m}$ = tesseral coefficients ($n \neq m$)

The J_n nomenclature is mostly of historical use in satellite geodesy; most modern notation consists solely of references to $C_{n,m}$ and $S_{n,m}$. In parallel with the nomenclature for the coefficient each term contains, the terms in Eq. 1.18 are referred to as zonal, sectorial, and tesseral terms.

Tables of mass coefficients are usually provided with the specific gravitational parameter μ and scaling (or reference) radius a_{eq} for which the coefficients are calibrated. These model-specific values *must* be used with the corresponding tables of coefficients to obtain correct results.

The phrase “gravity model” typically refers exclusively to the tables of coefficients and their constants. The potential functions are the same for *all* central bodies; the model (coefficients and constants) for a desired central body is simply plugged into the potential function to approximate the gravitational potential.

If the origin of the coordinate system used to derive the model is coincident with the center of mass of the central body, all of the coefficients with a degree of one (1) become exactly zero. It will thus be assumed for the rest of the paper that the $C_{1,0}$, $S_{1,1}$, and $C_{1,1}$ coefficients are all zero and thus sums will begin with degree two (2).

Legendre Polynomials and Associated Legendre Functions

The various functions denoted P in this paper are the Legendre polynomials and the ALFs. Legendre polynomials of the first kind are denoted P_n. With the exception of the equations in Section 1.2.4, the Legendre polynomials for the remainder of the paper always have an argument of $\sin\phi$. ALFs of the first kind are similarly denoted $P_{n,m}$ also with argument $\sin\phi$. Legendre polynomials and ALFs have subscripts n and m, which are the degree and order of the polynomial, respectively. ALFs with $m > n$ are defined as zero. The Legendre polynomials are equivalent to the ALF of the same degree and with $m = 0$.

For the remainder of this document, the argument of $\sin\phi$ will be assumed and omitted for brevity for all Legendre polynomials and ALFs.

Square Gravity Models

The $S_{n,m}$, $C_{n,m}$, and $P_{n,m}$ values can be used to form lower triangular matrices, where the terms above the diagonal are all zero, n is the row, and m is the column.

Models where maximum degree and order are equal are referred to as *square models.* Let n_d be the desired maximum degree and order of the gravity model. Then

$$u = \frac{\mu}{r_p}\left\{1 + \sum_{n=2}^{n_d}\sum_{m=0}^{n}\left(\frac{a_{eq}}{r_p}\right)^n P_{n,m}\left(S_{n,m}\sin m\theta + C_{n,m}\cos m\theta\right)\right\} \tag{1.20}$$

With the zonal terms separated out, Eq. 1.20 becomes

$$\begin{aligned} u = \frac{\mu}{r_p}\Bigg\{1 &+ \sum_{n=2}^{n_d}\left(\frac{a_{eq}}{r_p}\right)^n P_n C_{n,0} \\ &+ \sum_{n=2}^{n_d}\sum_{m=1}^{n}\left(\frac{a_{eq}}{r_p}\right)^n P_{n,m}\left(S_{n,m}\sin m\theta + C_{n,m}\cos m\theta\right)\Bigg\} \end{aligned} \tag{1.21}$$

Non-Square Gravity Models

Whereas Eqs. 1.20 and 1.21 assume square models, it is also possible to approximate the gravitational potential using *non-square models.* This can be accomplished by changing the bounds of the sums in Eqs. 1.20 and 1.21 to n_d and m_d, the desired degree and desired order, respectively. The outcome of a non-square model can then be written as

$$u = \frac{\mu}{r_p}\left\{1 + \sum_{n=2}^{n_d}\sum_{\substack{m=0\\ m\le m_d}}^{n}\left(\frac{a_{eq}}{r_p}\right)^n P_{n,m}\left(S_{n,m}\sin m\theta + C_{n,m}\cos m\theta\right)\right\} \tag{1.22}$$

resembling Eq. 1.20, or separating out zonal terms,

$$\begin{aligned} u = \frac{\mu}{r_p}\Bigg\{1 &+ \sum_{n=2}^{n_d}\left(\frac{a_{eq}}{r_p}\right)^n P_n C_{n,0} \\ &+ \sum_{n=2}^{n_d}\sum_{\substack{m=1\\ m\le m_d}}^{n}\left(\frac{a_{eq}}{r_p}\right)^n P_{n,m}\left(S_{n,m}\sin m\theta + C_{n,m}\cos m\theta\right)\Bigg\} \end{aligned} \tag{1.23}$$

resembling Eq. 1.21.

Because most algorithms for generating the Legendre polynomials and ALFs are optimized to generate values for square models, it is best to pass only the desired degree of the non-square gravity model to these function-generating subroutines. In calculating a non-square potential, the excess ALFs beyond the desired order are then simply unused.

The fact that Eqs. 1.20 and 1.21 (and thus Eqs. 1.22 and 1.23) are orthogonal expansions of u means that any lower-order expansion is merely a truncated higher-order expansion. No refit of the coefficients is necessary [2].

1.3 Coordinates

Gravitational potential u is typically resolved using equatorial central-body-centered, central-body-fixed Cartesian coordinates $\begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T$, where

- x_b is the coordinate positive along a line from the center of mass to the intersection of the equator and prime meridian of the central body
- z_b is the coordinate positive along the north polar axis of the central body
- y_b is the coordinate positive along the vector that completes the right-handed triad (y_b is positive in the eastern hemisphere)

Spherical coordinates $\begin{bmatrix} r & \theta & \phi \end{bmatrix}^T$ can then be defined in terms of the central-body-fixed coordinates

- $r = \sqrt{x_b^2 + y_b^2 + z_b^2}$ = magnitude of satellite position vector (previously r_p)
- $\theta = \text{arctan2}(y_b, x_b)$ = (positive east) longitude
- $\phi = \arcsin(\frac{z_b}{r})$ = (positive north) latitude or declination

If $x_b = y_b = 0$, then θ may be set to any value. For convenience, it is typically set to $\theta = 0$ in this case.

The relationship between central-body-fixed and spherical coordinates is shown in Figure 1.2. The spherical coordinate angles themselves are rarely used directly since they typically appear as arguments in trigonometric functions. See Section 1.3.2 for more information.

1.3.1 The Gravitational Acceleration

The gradient of the gravitational potential u is the force of gravity per unit mass, which is the *acceleration*. The gravitational acceleration at the central-body-fixed Cartesian coordinates $\begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T$ is thus

$$a = \begin{bmatrix} a_{x_b} & a_{y_b} & a_{z_b} \end{bmatrix}^T = \nabla_b u = \begin{bmatrix} \frac{\partial u}{\partial x_b} & \frac{\partial u}{\partial y_b} & \frac{\partial u}{\partial z_b} \end{bmatrix}^T \tag{1.24}$$

where ∇_b is the gradient with respect to the central-body-fixed coordinates.

Taking the gradient with respect to a central-body-fixed coordinate system means that we must include the Coriolis, centripetal, tangential, and relative acceleration terms [6, pg. 54-55]. To eliminate the complication of including these terms, an alternative coordinate system centered in the central body (frame rotationally fixed with respect to satellite) is used to compute the gradient.

Consider a displacement $\Delta\theta$ of the position vector in spherical coordinates in the direction of increasing θ (away from the x_b axis). While displacement of θ is actually the arc of a circle, a limit is approached as $\Delta\theta \to 0$ when taking the gradient in spherical coordinates and thus the displacement arc becomes arbitrarily close to its subtending chord. The instantaneous positive displacement of θ for a vector in spherical coordinates is thus perpendicular to the radial vector in the direction of increasing θ. This logic can be similarly applied for ϕ. The directions of increasing θ and ϕ form the basis of the new coordinate system [7, pg. 143].

The new x axis is parallel to the position vector and is defined as the r_o axis. The new y axis is located on the central-body-fixed x_by_b plane at a right-angle to the r_o axis in the direction of increasing θ and is defined as the θ_o axis. The new z axis completes the right-handed triad and is defined as the ϕ_o axis. The

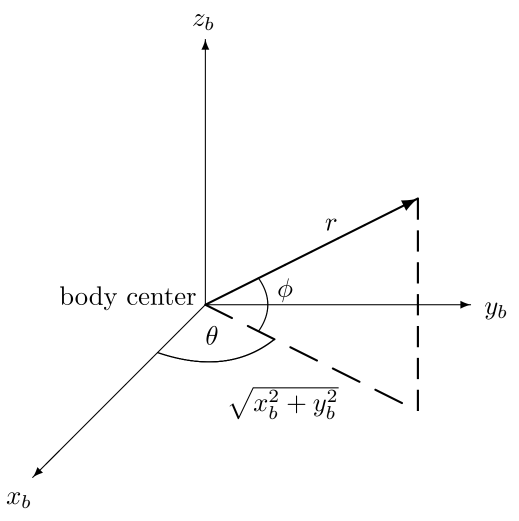

Figure 1.2: Body-fixed coordinates and spherical coordinates illustrated.

ϕ_o axis forms the angle ϕ with the z_b axis in the direction of increasing ϕ (away from the z_b axis). The coordinates along these new axes are referred to as the *orthogonal spherical coordinates* $\begin{bmatrix} r_o & \theta_o & \phi_o \end{bmatrix}^T$, not to be confused with the conventional spherical coordinates $\begin{bmatrix} r & \theta & \phi \end{bmatrix}^T$. The relationship between central-body-fixed coordinates and orthogonal spherical coordinates is shown in Figure 1.3.

The transformation matrix from the orthogonal spherical coordinates to the central-body-fixed coordinates is [2]

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \begin{bmatrix} \cos\phi\cos\theta & -\sin\theta & -\sin\phi\cos\theta \\ \cos\phi\sin\theta & \cos\theta & -\sin\phi\sin\theta \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} r_o \\ \theta_o \\ \phi_o \end{bmatrix} \tag{1.25}$$

All of the trigonometric functions in the transformation can be precomputed using the trigonometric relationships outlined in Section 1.3.2.

The gravitational acceleration in the orthogonal spherical coordinate system is [6, pg. 54]

$$a = \begin{bmatrix} a_{r_o} & a_{\theta_o} & a_{\phi_o} \end{bmatrix}^T = \nabla u = \begin{bmatrix} \frac{\partial u}{\partial r} & \frac{1}{r\cos\phi}\frac{\partial u}{\partial \theta} & \frac{1}{r}\frac{\partial u}{\partial \phi} \end{bmatrix}^T \tag{1.26}$$

The result of Eq. 1.26 can then be transformed back to central-body-fixed coordinates via Eq. 1.25.

1.3.2 Trigonometric Relationships

Instead of solving for the spherical coordinate angles ϕ and θ, it is more efficient to pre-compute the values of trigonometric functions of these angles given by the definitions of sine and cosine.

$$\sin\theta = \frac{y_b}{\sqrt{x_b^2+y_b^2}} \tag{1.27}$$

$$\cos\theta = \frac{x_b}{\sqrt{x_b^2+y_b^2}} \tag{1.28}$$

$$\sin\phi = \frac{z_b}{r} \tag{1.29}$$

$$\cos\phi = \frac{\sqrt{x_b^2+y_b^2}}{r} \tag{1.30}$$

A logical check should be included prior to computing these values to identify the cases $x_b = y_b = 0$ or $r = 0$. If $x_b = y_b = 0$, then the equations $\sin\theta = 0$ and $\cos\theta = 1$ are typically substituted.

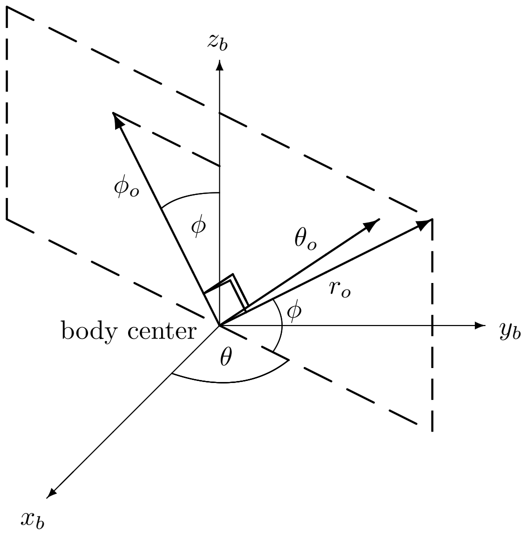

Figure 1.3: Orthogonal spherical coordinates $\begin{bmatrix} r_o & \theta_o & \phi_o \end{bmatrix}^T$ and central-body-fixed coordinates $\begin{bmatrix} x_b & y_b & z_b \end{bmatrix}^T$ shown with the $r_o\phi_o$ plane.

Chapter 2

Normalization

2.1 The Normalization Factor

Using a normalization factor allows mass coefficients to be electronically represented as tractable values well within the valid range for IEEE-754 double-precision floating point variables, even when degree n and order m are relatively large. The mass coefficients of a central body are "normalized" when they are divided by this normalization factor $N_{n,m}$, typically defined as [8, pg. 544]

$$N_{n,m} = \sqrt{\frac{(n-m)!(2n+1)(2-\delta_{0,m})}{(n+m)!}}, \; N_n = N_{n,0} \tag{2.1}$$

where $\delta_{0,m}$ is the Kronecker delta function that returns one if $m = 0$ and zero otherwise. Table 2.1 lists the normalization factors through degree and order of four.

Historically, normalized mass coefficients were "unnormalized" by the end user of the model by multiplying the mass coefficients with their corresponding normalization factor. This was done because conventional gravitational potential algorithms required unnormalized coefficients in their formulations. A fundamental problem encountered by attempting to unnormalize mass coefficients with very high degrees and orders is that the normalization factors become prone to overflow or underflow in modern computers. This is due to the factorial in the denominator of Eq. 2.1, $(n+m)!$. The user is restricted to $n_d + m_d < 171$ using IEEE 754 double-precision floating point variables to calculate normalization factors because computing such a factorial expression outside of Eq. 2.1 alone would overflow beyond this boundary (see Section 2.2).

It was later established that the additional work of unnormalizing mass coefficients becomes unnecessary by "normalizing" ALFs in gravitational potential formulations. Legendre polynomials and ALFs are said to be "normalized" when multiplied by $N_{n,m}$. This normalization scheme is ideal because the product of an ALF and its corresponding coefficient is equal to the product of the normalized ALF and its corresponding

Table 2.1: Normalization Factors Through 4×4

	$m \rightarrow$				
$n \downarrow$	0	1	2	3	4
0	1	0	0	0	0
1	$\sqrt{3}$	$\sqrt{3}$	0	0	0
2	$\sqrt{5}$	$\sqrt{\frac{5}{3}}$	$\frac{1}{2}\sqrt{\frac{5}{3}}$	0	0
3	$\sqrt{7}$	$\sqrt{\frac{7}{6}}$	$\frac{1}{2}\sqrt{\frac{7}{15}}$	$\frac{1}{6}\sqrt{\frac{7}{10}}$	0
4	3	$3\sqrt{\frac{1}{10}}$	$\frac{1}{2}\sqrt{\frac{1}{5}}$	$\frac{1}{2}\sqrt{\frac{1}{70}}$	$\frac{1}{8}\sqrt{\frac{1}{35}}$

normalized coefficient. In this paper, an overbar will be used to indicate normalized quantities; i.e. $\bar{C}_{n,m}$ and $\bar{P}_{n,m}$. By the aforementioned normalization conventions, Eq. 2.2 shows that the product of the ALF and corresponding coefficient holds true for normalized quantities.

$$\bar{P}_{n,m}\bar{C}_{n,m} = (P_{n,m}N_{n,m})\left(\frac{C_{n,m}}{N_{n,m}}\right) = P_{n,m}C_{n,m} \tag{2.2}$$

This eliminates the restrictions imposed by requiring unnormalized mass coefficients for gravitational potential formulations. The relationship from Eq. 2.2 can be seen with $S_{n,m}$ and $\bar{S}_{n,m}$ as well.

2.2 Recursive Mass Coefficient Normalization

Gravity models evaluating spherical harmonic associated Legendre functions at high degree n and order m require normalized Legendre coefficients to accurately compute terms ranging over hundreds of orders of magnitude. The standard normalization scheme is to multiply the unnormalized coefficients by the reciprocal of the normalization factor $N_{n,m}$, defined in Eq. 2.1. If $1/N_{n,m}$ is calculated directly, computation difficulties arise when $n + m > 170$ because 171! triggers an overflow condition in IEEE-754 double-precision (64-bit) real numbers. The overflow limit is ±1.797693134862316e+308[1].

Suppose $1/N_{n,m}$ values are directly calculated and grouped as elements in a lower-triangular matrix, as is customary, by incrementing m from 0 until $m = n$ before n is incremented to begin another matrix row. In performing this task, the first $(n + m)!$ overflow will be encountered for $1/N_{86,85}$. Table 2.2 provides $1/N_{n,m}$ values neighboring the $1/N_{86,85}$ element. In Table 2.2, $1/N_{n,m}$ values that *can* be computed are nowhere near an overflow condition because they are quotients with large denominators. This suggests a recursive computation will succeed in generating accurate $1/N_{n,m}$ values far beyond element $1/N_{86,85}$. The recursion will only fail when the $1/N_{n,m}$ quotient overflows.

Table 2.2: Reciprocal of Normalization Factors Neighboring $1/N_{86,85}$

	Order m		
Degree n	84	85	86
85	1.117258027e+151	1.456726244e+152	N/A
86	1.024089587e+152	Overflow	Overflow
87	Overflow	Overflow	Overflow

To document the recursion, suppose the value of $1/N_{n',m'}$ is given as x. Assuming $n' - m' > 0$ for "upward" or "rightward" recursion[2], Table 2.3 supplies recursive formulae for adjacent elements in terms of x and its associated degree n' and order m'.

Table 2.3: Recursions for Generating Recursions of Normalization Factors

	Order		
Degree	$m' - 1$	m'	$m' + 1$
$n' - 1$		$x\sqrt{\frac{(n'-m')(2n'+1)}{(n'+m')(2n'-1)}}$	
n'	$x\sqrt{\frac{2-\delta_{0,m'}}{(n'+m')(n'-m'+1)}}$	$x = 1/N_{n',m'}$	$x\sqrt{\frac{(n'+m'+1)(n'-m')}{2-\delta_{0,m'}}}$
$n' + 1$		$x\sqrt{\frac{(n'+1+m')(2n'+1)}{(n'+1-m')(2n'+3)}}$	

[1]This number was obtained from the `realmax('double')` command in MATLAB.

[2]These forbidden recursions would otherwise step outside lower-triangular matrix limits if x corresponded to a diagonal element with $n' - m' = 0$.

Each formula has been verified using the three finite $1/N_{n,m}$ elements appearing in Table 2.2. An example of invoking the "downward" recursion is shown in Eq. 2.3.

$$N_{86,84} = N_{85,84}\sqrt{\frac{170 \cdot 171}{2 \cdot 173}} \approx 9.16609737241 N_{85,84} \tag{2.3}$$

Most gravity models are formally published with normalized coefficients. Those wishing to use normalized coefficients at $n+m > 170$ in an unnormalized model will find the Table 2.3 recursions useful in their work. At $n + m \leq 170$, the Table 2.3 recursions offer computational efficiencies over the reciprocal of Eq. 2.1 evaluations, but these will be of little consequence if coefficients are to be unnormalized in a single pass with the results stored for all subsequent use.

2.3 Normalization Ratios

Consider a recursion formula for Legendre polynomials [9, pg. 114, sec. 3]

$$P_n = \frac{1}{n}[(2n-1)\sin\phi\, P_{n-1} - (n-1)P_{n-2}] \tag{2.4}$$

The corresponding normalized Legendre polynomial $\bar{P}_n$ is found by multiplying Eq. 2.4 by the normalization factor N_n.

$$\bar{P}_n = N_n P_n = \frac{1}{n}[(2n-1)\sin\phi\, N_n P_{n-1} - (n-1)N_n P_{n-2}] \tag{2.5}$$

Because Eq. 2.5 returns normalized polynomials and must recur over its own inputs, the equation needs to be written as a function of normalized polynomials by replacing the conventional polynomials with their normalized equivalents. By substituting

$$P_n = \frac{\bar{P}_n}{N_n} \tag{2.6}$$

Eq. 2.5 becomes

$$\bar{P}_n = \frac{1}{n}\left[(2n-1)\sin\phi\, \frac{N_n}{N_{n-1}}\bar{P}_{n-1} - (n-1)\frac{N_n}{N_{n-2}}\bar{P}_{n-2}\right] \tag{2.7}$$

Eq. 2.7 now contains *ratios* of normalization factors, namely $\frac{N_n}{N_{n-1}}$ and $\frac{N_n}{N_{n-2}}$.

2.4 The Recursion Normalization Parameter

The ratios of the normalization factors, which are referred to as the *normalization parameters* and denoted by λ, can be pre-computed since many ratios will be needed more than once while normalizing the algorithms. The ratios of Eq. 2.7 are written as λ_{n-1} and λ_{n-2}, where the subscripts of the parameters refer to the subscript of the corresponding polynomial that they normalize in Eq. 2.7.

When $n = 2$, $\lambda_{n-1} = \lambda_1$. However, when $n = 3$, $\lambda_{n-2} \neq \lambda_1$. For this reason, the parameters are written as functions of the current values of n and m, such as $\lambda_{n-1}(n)$ and $\lambda_{n-2,m}(n,m)$, and ignore the actual value of the subscripts of the parameters. The subscripts are merely notation used to identify a specific parameter for normalizing the ALF with the same subscripts.

This notation is used in Eq. 2.7 to obtain

$$\bar{P}_n = \frac{1}{n}[(2n-1)\sin\phi\, \lambda_{n-1}(n)\bar{P}_{n-1} - (n-1)\lambda_{n-2}(n)\bar{P}_{n-2}] \tag{2.8}$$

An equation for the parameter λ_{n-1} can be found by substituting the definition of the normalization factors (Eq. 2.1) in the ratio and simplifying.

$$\lambda_{n-1}(n) = \frac{N_n}{N_{n-1}} = \frac{\sqrt{2n+1}}{\sqrt{2(n-1)+1}} = \sqrt{\frac{2n+1}{2n-1}} \tag{2.9}$$

The Lear algorithm (Section 3.2) requires five normalization parameters to recursively compute normalized Legendre polynomials and ALFs. Each of these parameters can similarly be derived by simplifying the definition of the normalization factor for each of the corresponding ratios.

Chapter 3

The Three Singularity-Free Algorithms

By inspection, we can identify two potential singularities in Eq. 1.26. They can occur when $\cos \phi = 0$ (such as in a polar orbit) or when taking the partial derivative of ALFs with $m = 1$. Three unique algorithms have been developed to eliminate these singularities by Samuel Pines [1], Bill Lear [2], and Robert Gottlieb [3].

3.1 Pines Algorithm

3.1.1 Basis of the Pines Approach

Pines approached the problem by first transforming a position vector to central-body-fixed coordinates. By reallocating factors in each term of the potential, he defined a series of special functions in terms of the unit position vector components, which can then be solved recursively without singularities. A set of polynomials he referred to as the *derived ALFs* were created by modifying the conventional definition of ALFs. The derived ALFs have similar recursive behaviors to their conventional counterparts but have no discontinuities in their partial derivatives.

3.1.2 Pines Algorithm Implementations

The Pines acceleration algorithm has previously been normalized by Lundberg and Schutz [4]. Within the normalized algorithm, Pines' derived ALFs can be recursively generated in a number of ways. To evaluate the various approaches, Lundberg and Schutz developed seven different recursions algorithms and then performed numerical analyses of the stability of normalized and unnormalized versions of each recursion algorithm. Lundberg and Schutz concluded that a simple row-wise or column-wise recursion provides the most stability of all the algorithms, normalized or unnormalized. In this analysis, the normalized and unnormalized versions of the column-wise recursion by Lundberg and Schutz [4, Recursion I] are utilized for implementations of Pines. The implementations used here are based on a normalized implementation provided by DeMars [5].

3.2 Lear Algorithm

3.2.1 Basis of the Lear Approach

Lear transformed the position vector to the orthogonal spherical coordinate system. This results in several $\sec \phi$ factors that emerge in the equations for the acceleration. Lear found that stable recursions could be developed by assimilating the $\sec \phi$ factors in the ALF recursion equations. Lear utilized the conventional (unmodified) ALFs and thus used traditional recursion equations for ALFs, across which he simply distributed the $\sec\phi$ factors. Terms in the potential that include an ALF but no $\sec\phi$ could easily eliminate the secant (which has been combined in the value of the ALF) by multiplying the term by $\cos\phi$, a value with no discontinuity.

3.2.2 Normalized Lear Algorithm

Each of the recursion equations of the Lear algorithm is now normalized using the normalization parameters of Section 2.4. The parameters required in the Lear algorithm are defined in Table 3.1.

Table 3.1: Normalization Parameters λ for a Normalized Lear Algorithm

$$\lambda_{n-1}(n) = \frac{N_n}{N_{n-1}} = \sqrt{\frac{2n+1}{2n-1}} \tag{3.1}$$

$$\lambda_{n-2}(n) = \frac{N_n}{N_{n-2}} = \sqrt{\frac{2n+1}{2n-3}} \tag{3.2}$$

$$\lambda_{n-1,n-1}(n) = \frac{N_{n,n}}{N_{n-1,n-1}} = \frac{1}{2n-1}\sqrt{\frac{2n+1}{2n}} \tag{3.3}$$

$$\lambda_{n-1,m}(n,m) = \frac{N_{n,m}}{N_{n-1,m}} = \sqrt{\frac{(n-m)(2n+1)}{(n+m)(2n-1)}} \tag{3.4}$$

$$\lambda_{n-2,m}(n,m) = \frac{N_{n,m}}{N_{n-2,m}} = \sqrt{\frac{(n-m)(n-m-1)(2n+1)}{(n+m)(n+m-1)(2n-3)}} \tag{3.5}$$

Each of the original equations can be found in Ref. [2]. Only the normalized equations are presented here and should replace their analogues in the original algorithm. The added normalization parameters are denoted with an underbrace or overbrace to bring attention to the changes from the equations in the original document.

In each of the following recursions, let n_d be the desired degree and order of the gravity model.

Recursion for Zonal Legendre Polynomials $\bar{P}_n$

This recursion utilizes Parameters 3.1 and 3.2. For $n = 2$ through n_d

$$\bar{P}_n = \frac{1}{n}[(2n-1)\sin\phi\, \underbrace{\lambda_{n-1}(n)}_{3.1}\, \bar{P}_{n-1} - (n-1)\, \underbrace{\lambda_{n-2}(n)}_{3.2}\, \bar{P}_{n-2}] \tag{3.6}$$

where $\bar{P}_0 = 1$ and $\bar{P}_1 = N_1 P_1 = \sqrt{3}\,\sin\phi$.

Recursion for Zonal Legendre Polynomial Derivatives $\bar{P}'_n$

This recursion utilizes Parameter 3.1. For $n = 2$ through n_d

$$\bar{P}'_n = \underbrace{\lambda_{n-1}(n)}_{3.1}[\sin\phi\, \bar{P}'_{n-1} + n\bar{P}_{n-1}] \tag{3.7}$$

where $\bar{P}'_1 = \sqrt{3}$.

Recursion for Sectorial ALFs $(\sec\phi\,\bar{P}_{n,n})$

This recursion utilizes Parameter 3.3. For $n = 2$ through n_d

$$(\sec\phi\,\bar{P}_{n,n}) = (2n-1)\cos\phi\,\underbrace{\lambda_{n-1,n-1}(n)}_{3.3}(\sec\phi\,\bar{P}_{n-1,n-1}) \tag{3.8}$$

where $(\sec\phi\,\bar{P}_{1,1}) = \sqrt{3}$.

Recursion for Tesseral ALFs $(\sec\phi\,\bar{P}_{n,m})$

This recursion utilizes Parameters 3.4 and 3.5. For $n = 2$ through n_d and (inner loop) $m = 1$ through $n-1$

$$\begin{aligned}(\sec\phi\,\bar{P}_{n,m}) = {} & [(2n-1)\sin\phi\,\overbrace{\lambda_{n-1,m}(n,m)}^{3.4}(\sec\phi\,\bar{P}_{n-1,m}) \\ & -(n+m-1)\,\underbrace{\lambda_{n-2,m}(n,m)}_{3.5}(\sec\phi\,\bar{P}_{n-2,m})]\frac{1}{n-m}\end{aligned} \tag{3.9}$$

where $(\sec\phi\,\bar{P}_{n-1,n}) = 0$ for $n = 1$ through n_d.

Recursion for Sectorial ALF Derivatives $(\cos\phi\,\bar{P}'_{n,n})$

This recursion has no normalization parameters because the input and output are the same degree and order. For $n = 1$ through n_d

$$(\cos\phi\,\bar{P}'_{n,n}) = -n\sin\phi\,(\sec\phi\,\bar{P}_{n,n}) \tag{3.10}$$

Recursion for Tesseral ALF Derivatives $(\cos\phi\,\bar{P}'_{n,m})$

This recursion utilizes Parameter 3.4. For $n = 2$ through n_d and (inner loop) $m = 1$ through $n-1$

$$\begin{aligned}(\cos\phi\,\bar{P}'_{n,m}) = {} & -n\sin\phi\,(\sec\phi\,\bar{P}_{n,m}) \\ & +(n+m)\,\underbrace{\lambda_{n-1,m}(n,m)}_{3.4}(\sec\phi\,\bar{P}_{n-1,m})\end{aligned} \tag{3.11}$$

3.2.3 Example of Normalized Lear Recursions

This section analytically demonstrates using normalized recursion relationships to generate Legendre polynomials and ALFs for a gravity model with a degree and order of four. Each of the final answers are written first simplified and then with its normalization factored out to demonstrate equivalence with its unnormalized value. These can be found in the examples outlined in Ref. [2].

Zonal Legendre Polynomials

$$\begin{aligned}\bar{P}_2 &= \frac{1}{2}\left[3\sin\phi\,\lambda_{n-1}(2)\bar{P}_1 - \lambda_{n-2}(2)\bar{P}_0\right] \\ &= \frac{1}{2}\left[3\sin\phi\sqrt{\frac{5}{3}}\sqrt{3}\sin\phi - \sqrt{5}\right] \\ &= \frac{3}{2}\sqrt{5}\sin^2\phi - \frac{1}{2}\sqrt{5} = \sqrt{5}\left(\frac{3}{2}\sin^2\phi - \frac{1}{2}\right) \\ \bar{P}_3 &= \frac{1}{3}\left[5\sin\phi\,\lambda_{n-1}(3)\bar{P}_2 - 2\lambda_{n-2}(3)\bar{P}_1\right]\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{3}\left[5\sin\phi\sqrt{\frac{7}{5}}\left(\frac{3}{2}\sqrt{5}\sin^2\phi-\frac{1}{2}\sqrt{5}\right)-2\sqrt{\frac{7}{3}}\sqrt{3}\sin\phi\right]\\
&= \frac{1}{3}\left[\frac{5\cdot 3}{2}\sqrt{7}\sin^3\phi-\frac{5}{2}\sqrt{7}\sin\phi-2\sqrt{7}\sin\phi\right]\\
&= \frac{5}{2}\sqrt{7}\sin^3\phi-\frac{3}{2}\sqrt{7}\sin\phi=\sqrt{7}\left(\frac{5}{2}\sin^3\phi-\frac{3}{2}\sin\phi\right)
\end{aligned}$$

$$\begin{aligned}
\bar{P}_4 &= \frac{1}{4}\left[7\sin\phi\,\lambda_{n-1}(4)\bar{P}_3-3\lambda_{n-2}(4)\bar{P}_2\right]\\
&= \frac{1}{4}\left[7\sin\phi\sqrt{\frac{9}{7}}\left(\frac{5}{2}\sqrt{7}\sin^3\phi-\frac{3}{2}\sqrt{7}\sin\phi\right)\right.\\
&\qquad\left.-3\sqrt{\frac{9}{5}}\left(\frac{3}{2}\sqrt{5}\sin^2\phi-\frac{1}{2}\sqrt{5}\right)\right]\\
&= \frac{1}{4}\left[\frac{7\cdot 5\cdot 3}{2}\sin^4\phi-\frac{7\cdot 3\cdot 3}{2}\sin^2\phi-\frac{3\cdot 3\cdot 3}{2}\sin^2\phi+\frac{3\cdot 3}{2}\right]\\
&= \frac{105}{8}\sin^4\phi-\frac{90}{8}\sin^2\phi+\frac{9}{8}=3\left(\frac{35}{8}\sin^4\phi-\frac{30}{8}\sin^2\phi+\frac{3}{8}\right)
\end{aligned}$$

Zonal Legendre Polynomial Derivatives

$$\begin{aligned}
\bar{P}'_2 &= \lambda_{n-1}(2)\left[\sin\phi\bar{P}'_1+2\bar{P}_1\right]\\
&= \sqrt{\frac{5}{3}}\left[\sin\phi\sqrt{3}+2\sqrt{3}\sin\phi\right]\\
&= 3\sqrt{5}\sin\phi=\sqrt{5}\left(3\sin\phi\right)
\end{aligned}$$

$$\begin{aligned}
\bar{P}'_3 &= \lambda_{n-1}(3)\left[\sin\phi\bar{P}'_2+3\bar{P}_2\right]\\
&= \sqrt{\frac{7}{5}}\left[\sin\phi\left(3\sqrt{5}\sin\phi\right)+3\left(\frac{3}{2}\sqrt{5}\sin^2\phi-\frac{1}{2}\sqrt{5}\right)\right]\\
&= 3\sqrt{7}\sin^2\phi+\frac{3\cdot 3}{2}\sqrt{7}\sin^2\phi-\frac{3}{2}\sqrt{7}\\
&= \frac{15}{2}\sqrt{7}\sin^2\phi-\frac{3}{2}\sqrt{7}=\sqrt{7}\left(\frac{15}{2}\sin^2\phi-\frac{3}{2}\right)
\end{aligned}$$

$$\begin{aligned}
\bar{P}'_4 &= \lambda_{n-1}(4)\left[\sin\phi\bar{P}'_3+4\bar{P}_3\right]\\
&= \sqrt{\frac{9}{7}}\left[\sin\phi\left(\frac{15}{2}\sqrt{7}\sin^2\phi-\frac{3}{2}\sqrt{7}\right)+4\left(\frac{5}{2}\sqrt{7}\sin^3\phi-\frac{3}{2}\sqrt{7}\sin\phi\right)\right]\\
&= \frac{3\cdot 15}{2}\sin^3\phi-\frac{3\cdot 3}{2}\sin\phi+\frac{3\cdot 4\cdot 5}{2}\sin^3\phi-\frac{4\cdot 3\cdot 3}{2}\sin\phi\\
&= \frac{105}{2}\sin^3\phi-\frac{45}{2}\sin\phi=3\left(\frac{35}{2}\sin^3\phi-\frac{15}{2}\sin\phi\right)
\end{aligned}$$

Sectorial (Diagonal) ALFs

$$(\sec\phi\,\bar{P}_{2,2}) = 3\cos\phi\,\lambda_{n-1,n-1}(2)(\sec\phi\,\bar{P}_{1,1})$$

$$\begin{aligned}
&= 3\cos\phi\,\frac{1}{3}\sqrt{\frac{5}{4}}\sqrt{3}\\
&= \frac{1}{2}\sqrt{15}\cos\phi = \frac{1}{2}\sqrt{\frac{5}{3}}\,(3\cos\phi)
\end{aligned}$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{3,3}) &= 5\cos\phi\,\lambda_{n-1,n-1}(3)(\sec\phi\,\bar{P}_{2,2})\\
&= 5\cos\phi\,\frac{1}{5}\sqrt{\frac{7}{6}}\left(\frac{1}{2}\sqrt{15}\cos\phi\right)\\
&= \frac{1}{2}\sqrt{\frac{35}{2}}\cos^2\phi = \frac{1}{6}\sqrt{\frac{7}{10}}\left(15\cos^2\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{4,4}) &= 7\cos\phi\,\lambda_{n-1,n-1}(4)(\sec\phi\,\bar{P}_{3,3})\\
&= 7\cos\phi\,\frac{1}{7}\sqrt{\frac{9}{8}}\left(\frac{1}{2}\sqrt{\frac{35}{2}}\cos^2\phi\right)\\
&= \frac{3}{8}\sqrt{35}\cos^3\phi = \frac{1}{8}\sqrt{\frac{1}{35}}\left(105\cos^3\phi\right)
\end{aligned}$$

Tesseral ALFs: Row 2 ($n = 2$)

$$\begin{aligned}
(\sec\phi\,\bar{P}_{2,1}) &= 3\sin\phi\,\lambda_{n-1,m}(2,1)(\sec\phi\,\bar{P}_{1,1}) - 2\lambda_{n-2,m}(2,1)(\sec\phi\,\bar{P}_{0,1})\\
&= 3\sin\phi\sqrt{\frac{5}{3\cdot 3}}\sqrt{3} - 2\sqrt{0}\cdot 0\\
&= \sqrt{15}\sin\phi = \sqrt{\frac{5}{3}}\,(3\sin\phi)
\end{aligned}$$

Tesseral ALFs: Row 3 ($n = 3$)

$$\begin{aligned}
(\sec\phi\,\bar{P}_{3,1}) &= \left[5\sin\phi\,\lambda_{n-1,m}(3,1)(\sec\phi\,\bar{P}_{2,1}) - 3\lambda_{n-2,m}(3,1)(\sec\phi\,\bar{P}_{1,1})\right]\frac{1}{2}\\
&= \frac{1}{2}\left[5\sin\phi\sqrt{\frac{2\cdot 7}{4\cdot 5}}\left(\sqrt{15}\sin\phi\right) - 3\sqrt{\frac{2\cdot 7}{4\cdot 3\cdot 3}}\sqrt{3}\right]\\
&= \frac{5}{2}\sqrt{\frac{7\cdot 3}{2}}\sin^2\phi - \frac{1}{2}\sqrt{\frac{7\cdot 3}{2}}\\
&= \frac{5}{2}\sqrt{\frac{21}{2}}\sin^2\phi - \frac{1}{2}\sqrt{\frac{21}{2}} = \sqrt{\frac{7}{6}}\left(\frac{15}{2}\sin^2\phi - \frac{3}{2}\right)
\end{aligned}$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{3,2}) &= 5\sin\phi\,\lambda_{n-1,m}(3,2)(\sec\phi\,\bar{P}_{2,2}) - 4\lambda_{n-2,m}(3,2)(\sec\phi\,\bar{P}_{1,2})\\
&= 5\sin\phi\sqrt{\frac{7}{5\cdot 5}}\left(\frac{1}{2}\sqrt{15}\cos\phi\right) - 4\sqrt{0}\cdot 0\\
&= \frac{5}{2}\sqrt{\frac{7\cdot 5\cdot 3}{5\cdot 5}}\sin\phi\cos\phi\\
&= \frac{1}{2}\sqrt{105}\sin\phi\cos\phi = \frac{1}{2}\sqrt{\frac{7}{15}}\,(15\sin\phi\cos\phi)
\end{aligned}$$

Tesseral ALFs: Row 4 ($n = 4$)

The example in the Lear document that corresponds to the following example contains a typographical error. The final result for the ALF should be

$$(\sec\phi\, P_{4,1}) = \frac{35}{2}\sin^3\phi - \frac{15}{2}\sin\phi$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{4,1}) &= \left[7\sin\phi\,\lambda_{n-1,m}(4,1)(\sec\phi\,\bar{P}_{3,1}) - 4\lambda_{n-2,m}(4,1)(\sec\phi\,\bar{P}_{2,1})\right]\frac{1}{3} \\
&= \frac{7}{3}\sin\phi\sqrt{\frac{3\cdot 9}{5\cdot 7}}\left(\frac{5}{2}\sqrt{\frac{21}{2}}\sin^2\phi - \frac{1}{2}\sqrt{\frac{21}{2}}\right) \\
&\quad - \frac{4}{3}\sqrt{\frac{3\cdot 2\cdot 9}{5\cdot 4\cdot 5}}\left(\sqrt{15}\sin\phi\right) \\
&= \frac{7\cdot 5}{3\cdot 2}\sqrt{\frac{3\cdot 9\cdot 7\cdot 3}{5\cdot 7\cdot 2}}\sin^3\phi - \frac{7}{3\cdot 2}\sqrt{\frac{3\cdot 9\cdot 7\cdot 3}{5\cdot 7\cdot 2}}\sin\phi \\
&\quad - \frac{4}{3}\sqrt{\frac{3\cdot 2\cdot 9\cdot 3\cdot 5}{5\cdot 4\cdot 5}}\sin\phi \\
&= \frac{21}{2}\sqrt{\frac{5}{2}}\sin^3\phi - \frac{9}{2}\sqrt{\frac{5}{2}}\sin\phi \\
&\quad = 3\sqrt{\frac{1}{10}}\left(\frac{35}{2}\sin^3\phi - \frac{15}{2}\sin\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{4,2}) &= \left[7\sin\phi\,\lambda_{n-1,m}(4,2)(\sec\phi\,\bar{P}_{2,2}) - 5\lambda_{n-2,m}(4,2)(\sec\phi\,\bar{P}_{2,2})\right]\frac{1}{2} \\
&= \frac{7}{2}\sin\phi\sqrt{\frac{2\cdot 9}{6\cdot 7}}\left(\frac{1}{2}\sqrt{105}\sin\phi\cos\phi\right) \\
&\quad - \frac{5}{2}\sqrt{\frac{2\cdot 9}{6\cdot 5\cdot 5}}\left(\frac{1}{2}\sqrt{15}\cos\phi\right) \\
&= \frac{7}{4}\sqrt{\frac{2\cdot 3\cdot 3\cdot 3\cdot 5\cdot 7}{2\cdot 3\cdot 7}}\sin^2\phi\cos\phi - \frac{5}{4}\sqrt{\frac{2\cdot 3\cdot 5\cdot 3\cdot 3}{2\cdot 3\cdot 5\cdot 5}}\cos\phi \\
&= \frac{21}{4}\sqrt{5}\sin^2\phi\cos\phi - \frac{3}{4}\sqrt{5}\cos\phi \\
&\quad = \frac{1}{2}\sqrt{\frac{1}{5}}\left(\frac{105}{2}\sin^2\phi\cos\phi - \frac{15}{2}\cos\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\sec\phi\,\bar{P}_{4,3}) &= 7\sin\phi\,\lambda_{n-1,m}(4,3)(\sec\phi\,\bar{P}_{1,1}) - 6\lambda_{n-2,m}(4,3)(\sec\phi\,\bar{P}_{2,3}) \\
&= 7\sin\phi\sqrt{\frac{9}{7\cdot 7}}\left(\frac{1}{2}\sqrt{\frac{35}{2}}\cos^2\phi\right) - 6\sqrt{0}\cdot 0 \\
&= \frac{7}{2}\sqrt{\frac{3\cdot 3\cdot 5\cdot 7}{2\cdot 7\cdot 7}}\sin\phi\cos^2\phi \\
&= \frac{3}{2}\sqrt{\frac{35}{2}}\sin\phi\cos^2\phi = \frac{1}{2}\sqrt{\frac{1}{70}}\left(105\sin\phi\cos^2\phi\right)
\end{aligned}$$

Sectorial (Diagonal) ALF Derivatives

$$(\cos\phi\,\bar{P}'_{1,1}) = -\sin\phi\,(\sec\phi\,\bar{P}_{1,1})$$

$$\begin{aligned}
&= -\sqrt{3}\sin\phi = \sqrt{3}\,(-\sin\phi)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{2,2}) &= -2\sin\phi\,(\sec\phi\,\bar{P}_{2,2}) \\
&= -2\sin\phi\,\left(\frac{1}{2}\sqrt{15}\cos\phi\right) \\
&= -\sqrt{15}\sin\phi\cos\phi = \frac{1}{2}\sqrt{\frac{5}{3}}\,(-6\sin\phi\cos\phi)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{3,3}) &= -3\sin\phi\,(\sec\phi\,\bar{P}_{3,3}) \\
&= -3\sin\phi\,\left(\frac{1}{2}\sqrt{\frac{35}{2}}\cos^2\phi\right) \\
&= -\frac{3}{2}\sqrt{\frac{35}{2}}\sin\phi\cos^2\phi = \frac{1}{6}\sqrt{\frac{7}{10}}\left(-45\sin\phi\cos^2\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{4,4}) &= -4\sin\phi\,(\sec\phi\,\bar{P}_{4,4}) \\
&= -4\sin\phi\,\left(\frac{3}{8}\sqrt{35}\cos^3\phi\right) \\
&= -\frac{3}{2}\sqrt{35}\sin\phi\cos^3\phi = \frac{1}{8}\sqrt{\frac{1}{35}}\left(-420\sin\phi\cos^3\phi\right)
\end{aligned}$$

Tesseral ALF Derivatives: Row 2 ($n = 2$)

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{2,1}) &= -2\sin\phi\,(\sec\phi\,\bar{P}_{2,1}) + 3\lambda_{n-1,m}(2,1)(\sec\phi\,\bar{P}_{1,1}) \\
&= -2\sin\phi\,\left(\sqrt{15}\sin\phi\right) + 3\sqrt{\frac{5}{3\cdot 3}}\sqrt{3} \\
&= -2\sqrt{15}\sin^2\phi + \sqrt{15} = \sqrt{\frac{5}{3}}\left(-6\sin^2\phi + 3\right)
\end{aligned}$$

Tesseral ALF Derivatives: Row 3 ($n = 3$)

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{3,1}) &= -3\sin\phi\,(\sec\phi\,\bar{P}_{3,1}) + 4\lambda_{n-1,m}(3,1)(\sec\phi\,\bar{P}_{2,1}) \\
&= -3\sin\phi\,\left(\frac{5}{2}\sqrt{\frac{21}{2}}\sin^2\phi - \frac{1}{2}\sqrt{\frac{21}{2}}\right) + 4\sqrt{\frac{2\cdot 7}{4\cdot 5}}\left(\sqrt{15}\sin\phi\right) \\
&= -\frac{15}{2}\sqrt{\frac{21}{2}}\sin^3\phi + \frac{3}{2}\sqrt{\frac{21}{2}}\sin\phi + \sqrt{2\cdot 7\cdot 4\cdot 3}\sin\phi \\
&= -\frac{15}{2}\sqrt{\frac{21}{2}}\sin^3\phi + \frac{11}{2}\sqrt{\frac{21}{2}}\sin\phi \\
&= \sqrt{\frac{7}{6}}\left(-\frac{45}{2}\sin^3\phi + \frac{33}{2}\sin\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{3,2}) &= -3\sin\phi\,(\sec\phi\,\bar{P}_{3,2}) + 5\lambda_{n-1,m}(3,2)(\sec\phi\,\bar{P}_{2,2}) \\
&= -3\sin\phi\,\left(\frac{1}{2}\sqrt{105}\sin\phi\cos\phi\right) + 5\sqrt{\frac{7}{5\cdot 5}}\left(\frac{1}{2}\sqrt{15}\cos\phi\right) \\
&= -\frac{3}{2}\sqrt{105}\sin^2\phi\cos\phi + \frac{1}{2}\sqrt{105}\cos\phi
\end{aligned}$$

$$= \frac{1}{2}\sqrt{\frac{7}{15}}\left(-45\sin^2\phi\cos\phi + 15\cos\phi\right)$$

Tesseral ALF Derivatives: Row 4 ($n = 4$)

The example in the Lear document that corresponds to the following example contains a typographical error. The final result for the ALF should be

$$(\cos\phi\, {P'}_{4,1}) = -70\sin^4\phi + \frac{135}{2}\sin^2\phi - \frac{15}{2}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{4,1}) &= -4\sin\phi\,(\sec\phi\,\bar{P}_{4,1}) + 5\lambda_{n-1,m}(4,1)(\sec\phi\,\bar{P}_{3,1}) \\
&= -4\sin\phi\left(\frac{21}{2}\sqrt{\frac{5}{2}}\sin^3\phi - \frac{9}{2}\sqrt{\frac{5}{2}}\sin\phi\right) \\
&\quad + 5\sqrt{\frac{3\cdot 9}{5\cdot 7}}\left(\frac{5}{2}\sqrt{\frac{21}{2}}\sin^2\phi - \frac{1}{2}\sqrt{\frac{21}{2}}\right) \\
&= -21\sqrt{10}\sin^4\phi + 9\sqrt{10}\sin^2\phi + \frac{5\cdot 5}{2}\sqrt{\frac{3\cdot 9\cdot 7\cdot 3}{2\cdot 5\cdot 7}}\sin^2\phi \\
&\quad - \frac{5}{2}\sqrt{\frac{3\cdot 9\cdot 7\cdot 3}{2\cdot 5\cdot 7}} \\
&= -21\sqrt{10}\sin^4\phi + \frac{81}{2}\sqrt{\frac{5}{2}}\sin^2\phi - \frac{9}{2}\sqrt{\frac{5}{2}} \\
&= 3\sqrt{\frac{1}{10}}\left(-70\sin^4\phi + \frac{135}{2}\sin^2\phi - \frac{15}{2}\right)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{4,2}) &= -4\sin\phi\,(\sec\phi\,\bar{P}_{4,2}) + 6\lambda_{n-1,m}(4,2)(\sec\phi\,\bar{P}_{3,2}) \\
&= -4\sin\phi\left(\frac{21}{4}\sqrt{5}\sin^2\phi\cos\phi - \frac{3}{4}\sqrt{5}\cos\phi\right) \\
&\quad + 6\sqrt{\frac{2\cdot 9}{6\cdot 7}}\left(\frac{1}{2}\sqrt{105}\sin\phi\cos\phi\right) \\
&= -21\sqrt{5}\sin^3\phi\cos\phi + 3\sqrt{5}\sin\phi\cos\phi \\
&\quad + 3\sqrt{\frac{9\cdot 3\cdot 5\cdot 7}{3\cdot 7}}\sin\phi\cos\phi \\
&= -21\sqrt{5}\sin^3\phi\cos\phi + 12\sqrt{5}\sin\phi\cos\phi \\
&= \frac{1}{2}\sqrt{\frac{1}{5}}\left(-210\sin^3\phi\cos\phi + 120\sin\phi\cos\phi\right)
\end{aligned}$$

$$\begin{aligned}
(\cos\phi\,\bar{P}'_{4,3}) &= -4\sin\phi\,(\sec\phi\,\bar{P}_{4,3}) + 7\lambda_{n-1,m}(4,3)(\sec\phi\,\bar{P}_{3,3}) \\
&= -4\sin\phi\left(\frac{3}{2}\sqrt{\frac{35}{2}}\sin\phi\cos^2\phi\right) + 7\sqrt{\frac{9}{7\cdot 7}}\left(\frac{1}{2}\sqrt{\frac{35}{2}}\cos^2\phi\right) \\
&= -3\sqrt{70}\sin^2\phi\cos^2\phi + \frac{3}{2}\sqrt{\frac{35}{2}}\cos^2\phi \\
&= \frac{1}{2}\sqrt{\frac{1}{70}}\left(-420\sin^2\phi\cos^2\phi + 105\cos^2\phi\right)
\end{aligned}$$

3.3 Gottlieb Algorithm

3.3.1 Basis of the Gottlieb Approach

Mueller [10] developed an efficient algorithm for solving the gravitational potential function which, like Pines, defined special functions by reallocating the factors in each term of the potential function. Gottlieb [3] then defined the gradient of Mueller's potential function in terms of the partial derivatives with respect to these special functions. Recursions for the partial derivatives of the special functions were then developed to resolve the gravitational acceleration.

Gottlieb's approach was originally documented in Ref. [11], but the use of an unstable ALF recursion limits its applicability to low degrees and orders. An updated algorithm was subsequently published in Ref. [3] including a more stable ALF generator. It should be noted, however, that the Ada code in Ref. [3] only features the stable ALF recursion in the normalized implementation. It is hereby recommended that any unnormalized implementation of the Gottlieb algorithm be based on the update in Ref. [3] with the careful inclusion of the stable ALF generator featured in the normalized implementation. Such a revised version of the unnormalized code was used in the current study and can be found in Section C.8.

3.3.2 Normalized Gottlieb Algorithm

Each of the recursion equations of the Gottlieb algorithm have been normalized in Ref. [3] using analogs to the normalization parameters of this document. The equations in the original normalized derivation have been reconfigured in this document to thus illustrate their use of the conventional normalization parameters in Section 2.4. Each of the original equations can be found in Ref. [3]. The added normalization parameters are denoted with an underbrace or overbrace to bring attention to the changes from the unnormalized equations in the original document.

Normalizing the Gottlieb recursions requires a few new normalization parameters in addition to the parameters derived for a normalized Lear implementation. The new normalization parameters are outlined in Table 3.2.

Table 3.2: Normalization Parameters λ Needed for a Normalized Gottlieb Algorithm

$$\lambda_{n,m+1}(n,m) = \frac{N_{n,m}}{N_{n,m+1}} = \sqrt{\frac{(n+m+1)(n-m)(2-\delta_{0,m})}{2}} \tag{3.12}$$

$$\lambda_{n,m+1}(n,0) = \frac{N_{n,0}}{N_{n,1}} = \sqrt{\frac{(n+1)n}{2}} \tag{3.13}$$

$$\tag{3.14}$$

In this section, $\sin\phi$ has been replaced by ϵ to keep a consistent notation with the original document.

Recursion for Sectorial ALFs $\bar{P}_n^n$

This recursion utilizes Parameter 3.3. For $n = 2$ through n_d

$$\bar{P}_n^n = \underbrace{\lambda_{n-1,n-1}(n,m)}_{3.3} \bar{P}_{n-1}^{n-1}(2n-1) \tag{3.15}$$

where $\bar{P}_1^1 = \sqrt{3}$.

Recursion for Tesseral ALFs $\bar{P}_n^{n-1}$

This recursion utilizes Parameter 3.12. For $n = 2$ through n_d

$$\bar{P}_n^{n-1} = \underbrace{\lambda_{n,m+1}(n, n-1)}_{3.12} \epsilon \bar{P}_n^n \tag{3.16}$$

Recursion for Zonal Legendre Polynomials $\bar{P}_n^0 = \bar{P}_n$

This recursion utilizes Parameters 3.1 and 3.2. For $n = 2$ through n_d

$$\bar{P}_n^0 = \bar{P}_n = \frac{1}{n}[(2n-1)\epsilon \underbrace{\lambda_{n-1}(n)}_{3.1} \bar{P}_{n-1} - (n-1) \underbrace{\lambda_{n-2}(n)}_{3.2} \bar{P}_{n-2}] \tag{3.17}$$

where $\bar{P}_0 = 1$ and $\bar{P}_1 = N_1 P_1 = \sqrt{3}\,\epsilon$.

Recursion for Tesseral ALFs $\bar{P}_n^m$

This recursion utilizes Parameters 3.4 and 3.5. For $n = 2$ through n_d and (inner loop) $m = 1$ through $n_d - 2$

$$\bar{P}_n^m = \frac{1}{n-m}[(2n-1)\epsilon \underbrace{\lambda_{n-1,m}(n,m)}_{3.4} \bar{P}_{n-1}^m - (n+m-1) \underbrace{\lambda_{n-2,m}(n,m)}_{3.5} \bar{P}_{n-2}^m] \tag{3.18}$$

Recursion for Intermediate Sum H_n

This recursion utilizes Parameters 3.13 and 3.12. For $n = 2$ through n_d

$$\begin{aligned} H_n &= C_{n,0} \overbrace{\lambda_{n,m+1}(n,0)}^{3.13} \bar{P}_n^1 \\ &\quad + \sum_{m=1}^{n} \underbrace{\lambda_{n,m+1}(n,m)}_{3.12} \frac{\bar{P}_n^{m+1}}{r^m} (C_{n,m} C_m + S_{n,m} S_m) \end{aligned} \tag{3.19}$$

THIS PAGE IS INTENTIONALLY LEFT BLANK.

Chapter 4

Verifications and Conclusions

Each of the algorithms addressed in this paper were at first coded directly from the unnormalized equations (or adapted from provided unnormalized code if included) in the original primary sources and then tested against normalized implementations of each developed by adding normalization parameters to the unnormalized codes (with the notable exception of the Pines algorithm, for which the already-coded DeMars normalized implementation was used in conjunction with the unnormalized implementation coded from scratch).

Surprisingly, the early results showed a fairly large discrepancy between the results of the normalized and unnormalized implementations for two of the three algorithms. Originally, only the Lear algorithm had consistent results between normalized and unnormalized implementations. Through numerical analysis and extremely laborious debugging, it became evident that the generator for the ALFs was the source of error. Further experimentation verified that the stability of the algorithms depended largely on the stability of the ALF generator used. Finally, the results of all testing appeared to indicate that normalization amplifies any inherent noise and error in each of the algorithms, a conclusion that further drove the development of additional conclusions and recommendations.

After appropriate updates were made to each of the codes, all six implementations agreed with each other in all tests with negligible error. Figure 4.1 shows that, using the proper implementations found in Appendix C, the differences in the algorithms are limited to numerical noise.

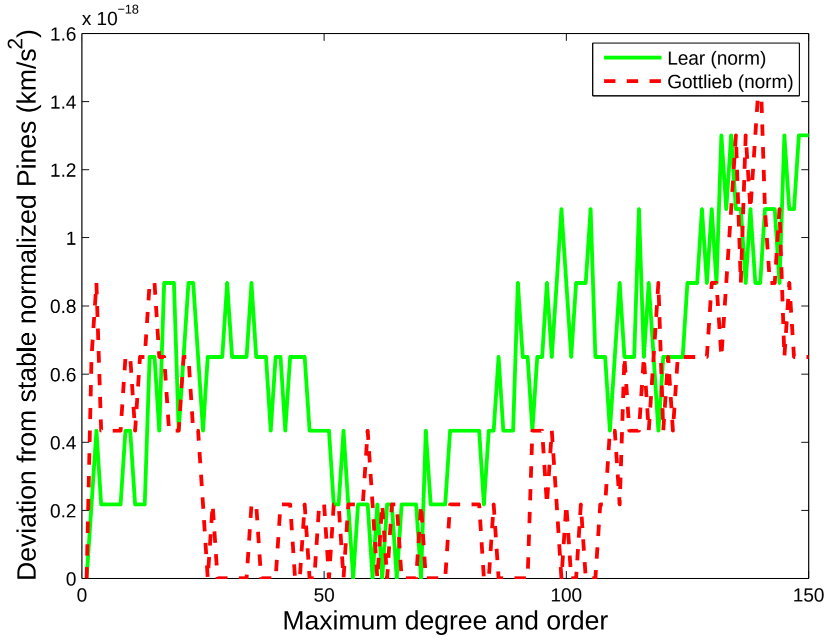

Figure 4.1: Acceleration error magnitude for normalized models at equator $(\phi = 0^\circ)$.

4.1 Implementation Notes

4.1.1 Pines Unnormalized

Since the normalized Pines implementation was provided to the authors already coded by DeMars but its results appeared more stable than the unnormalized implementation coded from scratch directly using the Pines equations, it was suspected that the two implementations were in some way different. As an experiment, the unnormalized Pines code was replaced by a copy of the normalized version of the code which had been "unnormalized," revealing that the ALF generator was in fact different between the two Pines implementations utilized in the first test.

4.1.2 Gottlieb Unnormalized

In a troubleshooting approach similar to that of the Pines issues early in testing, a line-by-line analysis of the provided Ada codes in Ref. [3] revealed that the unnormalized implementation continued to utilize a recursion equation described earlier in that document to be unstable. The code as-written for the normalized implementation, however, had been updated to include the same stable recursion as the Lear algorithm and the DeMars implementation of Pines [4, Recursion I]. Doing so resolved all of the errors seen in early preliminary testing. A corrected version of the code can be found in Section C.8.

4.2 Preliminary Testing

To test both the normalized and unnormalized implementations of each algorithm (six total implementations) for agreement, acceleration vectors were computed at a set of positions around a given central body. The Moon was chosen for the central body, and normalized LP150Q mass coefficients were utilized for the test (see Appendix C). The mass coefficients were unnormalized using the recursions in Section 2.2 to pass to the unnormalized algorithms. Tests utilized central-body-fixed position vectors with latitudes ranging from

$-90°$ to $+90°$ in $30°$ increments and at each longitude from $-150°$ to $+180°$ in $30°$ increments. Each position vector was given a 200-kilometer altitude above the lunar reference radius of 1738 kilometers.

At each of these positions, the acceleration was computed with all six algorithms with a variety of gravity model sizes. As a first check to ensure the algorithms functioned properly, a 0×0 model was tested to obtain the central body acceleration. Square models 2×2 through 50×50 were then tested, followed by the non-square models 50×0 through 50×49. Finally, the "extreme" cases of 125×125 and 150×150 were tested to ensure the normalized models in fact converged at relatively high degrees and orders. Section C.1 provides the MATLAB script used to drive the preliminary tests.

The DeMars [5] implementation of normalized Pines was considered the baseline model because it was based on the extensive stability studies of Lundberg and Schutz. Error, defined for each tested iteration as the magnitude of the delta vector between the DeMars-calculated acceleration vector and the calculated accleration vectors from each of the other algorithms, was considered acceptable if the order of magnitude was 10^{-18} or smaller. This is the order of magnitude of 10 times truncation error for the acceleration magnitudes tested, which was obtained by passing various acceleration vector magnitudes from the DeMars subroutine as arguments to the MATLAB `eps` function.

The output of the first MATLAB script to test the algorithms is listed in Appendix B. Once the implementations were all verified to contain stable ALF generators, the "preliminary" testing showed no major discrepancies in accuracy between the implementations. An additional trend study was conducted to further examine effects due to increasing degree and order.

4.3 Increasing Degree and Order Trend Study

The tests outlined in Section 4.2 were modified to only calculate accelerations with all the square models from 2×2 through 150×150. This would allow a trend to emerge when plotting the error versus the degree and order of the model if any existed. Section C.2 provides the MATLAB script used to drive the trend study tests. Meaningful plots could not be created because the differences between each of the nominal algorithms is on the order of numerical noise.

4.4 Trend Study with Unstable Associated Legendre Function Generators

To study trends in unstable ALF generators, Gottlieb and Pines implementations with known unstable ALF generators were intentionally run through the trending script.

Note that these results are *not* true results for the implementations described in Appendix C but implementations that were intentionally run in an unstable configuration.

Figure 4.2 shows the growth of deviation in an unnormalized Gottlieb from the normalized Pines at large degree and order using an unstable ALF generator[1]. At the equator, the deviation from normalized Pines

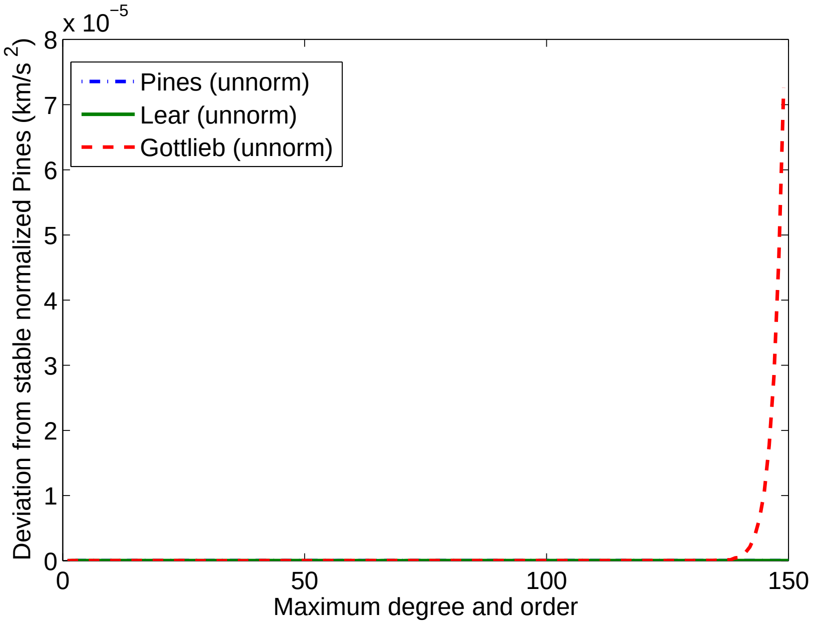

Figure 4.2: Acceleration error magnitude for unnormalized models (intentionally unstable Gottlieb) at equator ($\phi = 0°$)**.**

in both the normalized and unnormalized unstable Gottlieb grows slowly with increasing degree and order and suddenly diverges to positive infinity when the degree and order approaches 150. The rate of growth of error in Gottlieb is drastically larger as seen by comparing the scales of the y-axis of Figures 4.2 and 4.3. The beginning of the divergence of Gottlieb from the other two models (which remain closely in line with each other) in the unstable implementations can be clearly seen in Figure 4.4.

[1]This is the ALF generator utilized in the original Gottlieb reference [11] which was later found to be unstable. It has no analogue in Lundberg and Schutz [4].

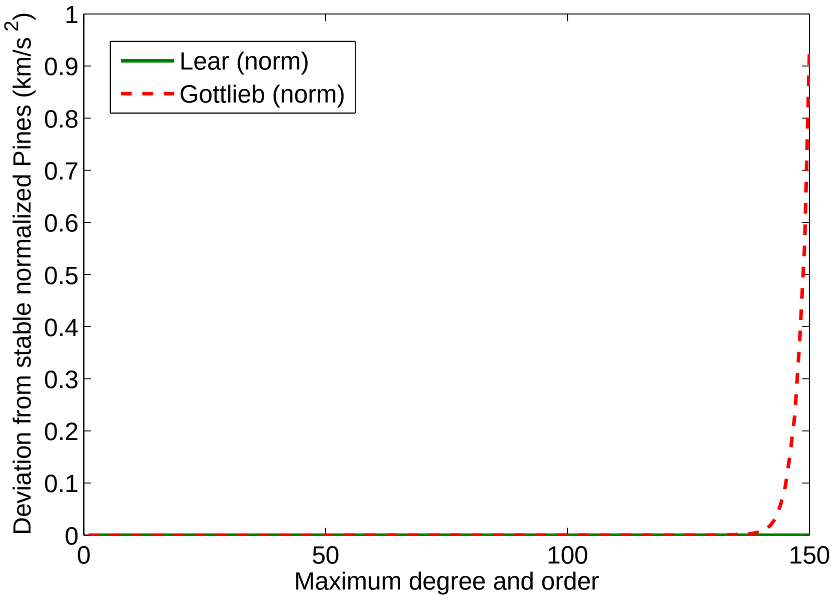

Figure 4.3: Acceleration error magnitude for normalized models (intentionally unstable Gottlieb) at equator ($\phi = 0°$).

This same test was run using a known unstable ALF generator[2] in the Pines algorithm. The error behavior of this known unstable algorithm at the poles mirrors the behavior of the Gottlieb error at the equator. This is shown by comparing unnormalized deviations from normalized Pines in Figure 4.5. A normalized implementation of the unstable Pines shows that the error is again much larger than the deviation of the unnormalized unstable Pines, as seen in Figure 4.6. This large disparity between normalized and unnormalized error also resembles the deviations of the two unstable Gottlieb implementations tested.

[2]Unnormalized Recursion IV from Lundberg and Schutz [4].

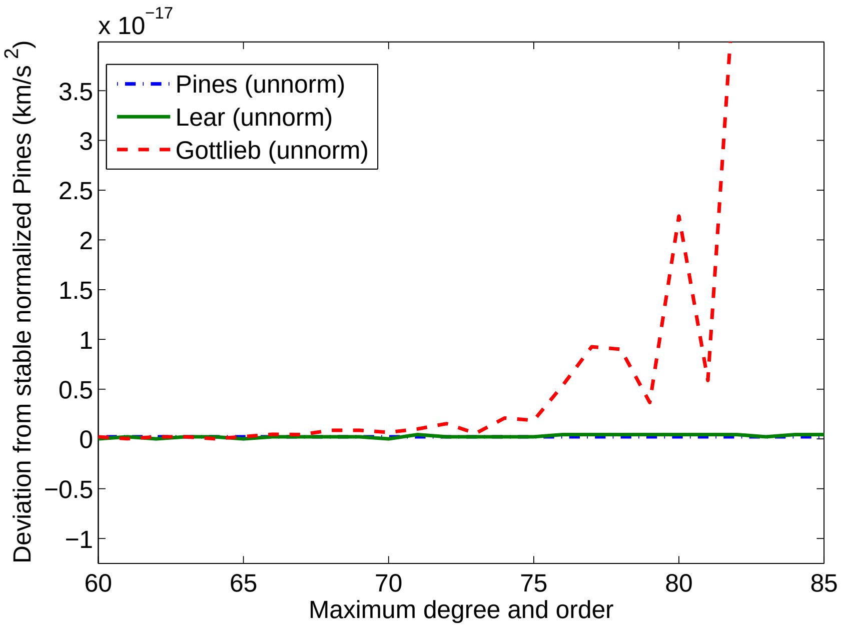

Figure 4.4: Acceleration error magnitude for unnormalized models (intentionally unstable Gottlieb) at equator ($\phi = 0°$) with degree and order 60–85.

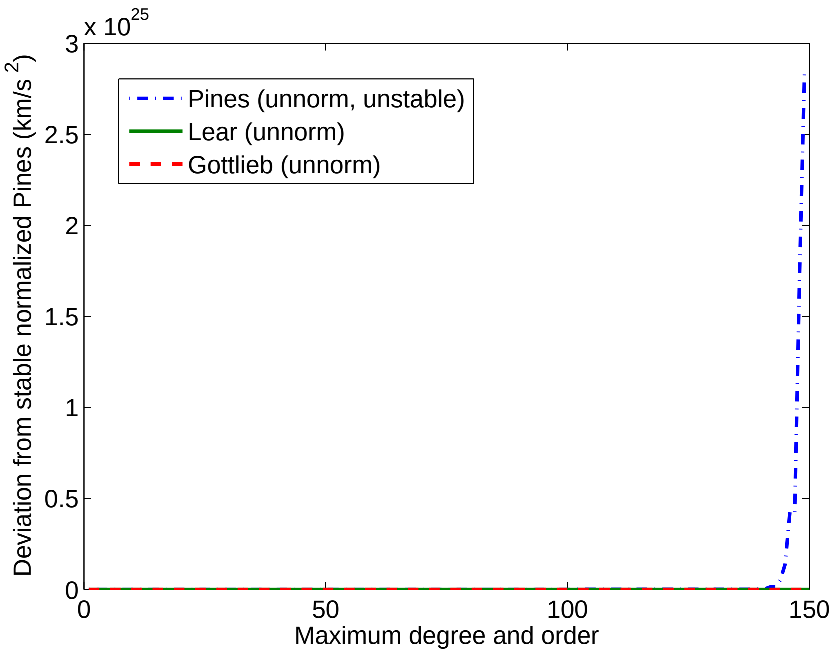

Figure 4.5: Acceleration error magnitude for unnormalized models (intentionally unstable Pines) at south pole ($\phi = -90°$).

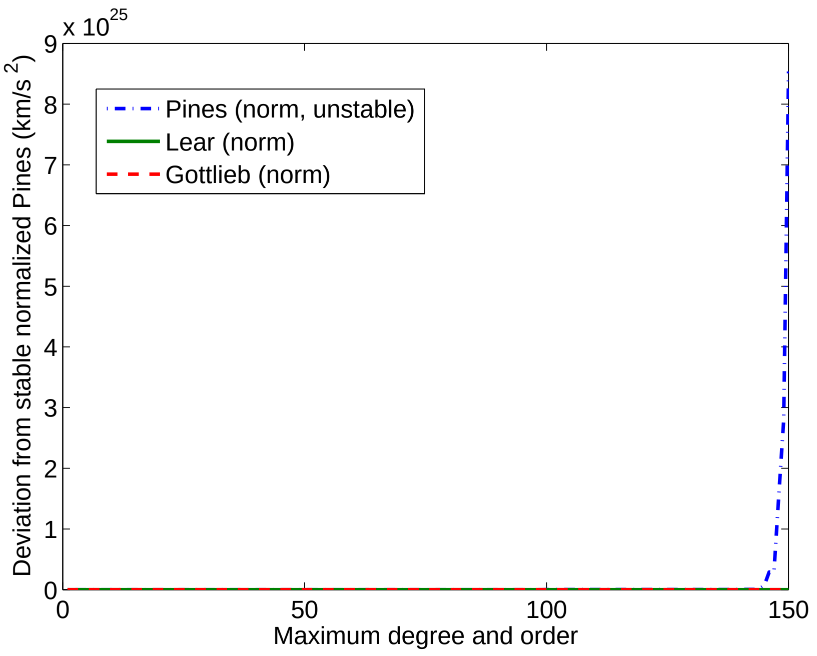

Figure 4.6: Acceleration error magnitude for normalized models (intentionally unstable Pines) at south pole ($\phi = -90°$).

4.5 Conclusions

Four primary conclusions can be drawn from the data presented in this paper.

1. **Pines (as implemented by DeMars), Lear, and Gottlieb (as implemented in this paper) algorithms are stable because they use a stable ALF recursion.** It is worth noting that virtually the same recursion equation [4, Recursion I] is used for generating ALFs in the stable implementations of all three algorithms. The very similar behavior between the three algorithms can thus be explained by the similarity in their ALF generators.

2. **Gottlieb [11] and Pines algorithms, as originally published, are unstable due to unstable ALF recursions.** The apparently unstable behavior of the original Gottlieb algorithm, also erroneously reprinted in the updated unnormalized implementation code [3], is presumed to be the result of an unstable ALF generator used in the algorithm. This conclusion is motivated by the similar signature of the error data with a known unstable ALF generator, the Pines algorithm with unnormalized recursion IV from Lundberg and Schutz [4] with which it was originally published. The error is position-vector-specific, much like the error that develops in the inherently unstable Pines implementation, albeit in a different position. The fact that both error signatures are latitude-dependent implies the ALF generator is the source of the instability, since the argument of ALFs used for spherical harmonic expansion is always $\sin\phi$ and is the only use of latitude in the equations.

3. **Normalization of recursions amplifies numerical instability.** Normalizing unstable ALF generators further decreases their stability and amplifies their error dramatically. This conclusion is supported by observing the same amplified error in both a known unstable ALF generator in Pines and the presumed-unstable ALF generator of the original Gottlieb in their "normalized" implementations relative to their unnormalized equivalents.

4. **Unnormalized algorithms provide perfectly valid results at high degree and order as long as coefficients can be reliably unnormalized.** Normalized and unnormalized implementations of all algorithms agree very well with each other, even at high degree and order. This leads to the conclusion that it is safe to use unnormalized algorithms as long as proper unnormalization of the coefficients is performed, such as using the recursions in Section 2.2, and ensuring the coefficients themselves are not too small to be electronically represented. If a relatively small gravity model is always desired, such as for computational speed efficiencies, it is perfectly acceptable to continue the practice of implementing unnormalized algorithms for calculating gravitational acceleration.

4.6 Recommendations

The authors present the following two recommendations:

1. **The Gottlieb [11] and Pines algorithms should not be implemented directly as originally published.** Unless the more stable ALF generation scheme of normalized Ref. [3] is implemented as is presented in this paper in Section C.8 or the improvements to Pines by DeMars, these algorithms should not be used in their originally published forms if incorporating models much larger than approximately 40×40 due to the potential for instability. **The normalized implementation later updated by Gottlieb [3], however, is perfectly valid as printed.**

   Stable recursions have been examined by Lundberg and Schutz [4], so their recursions should be adapted for new algorithms or implementations.

2. **Normalized implementations are better suited to software packages than unnormalized algorithms.** Normalized algorithms are not only better suited for acceleration calculation with larger gravity models because of the smaller range of orders of magnitude involved in computation, their consistency with unnormalized algorithms makes them desirable for all implementations in software

packages seeking to maintain versatility and robustness when computing gravitational acceleration. Normalized algorithms, when properly implemented, should always return a valid acceleration given $r > 0$, a set of normalized coefficients, and a valid degree and order. Fast recursions for generating the normalization parameters can also reduce or eliminate any differences in computational time from unnormalized implementations.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

Appendix A

Legendre Polynomials as a Maclaurin Series Expansion

A function of the form $f(\alpha) = (1-\alpha)^{-1/2}$ can be expanded through a Maclaurin series.

$$f(\alpha) = \sum_{n=0}^{\infty} \frac{\alpha^n}{n!} \left[\frac{d^n f(\beta)}{d\beta^n} \right]_{\beta=0} \tag{A.1}$$

By inspection, it can be shown that Eq. A.2 is the form of the derivative of f.

$$\frac{d^n f(\alpha)}{d\alpha^n} = \frac{(2n)!}{2^{2n} n!} (1-\alpha)^{-(2n+1)/2} \tag{A.2}$$

For the Maclaurin condition $\beta = 0$, the derivative reduces to the ratio of factorials. Substituting back into Eq. A.1,

$$f(\alpha) = \sum_{n=0}^{\infty} \frac{(2n)!}{(2^n n!)^2} \alpha^n \tag{A.3}$$

For the traditional expansion of $1/\sqrt{1 + R^2 - 2R\cos\gamma}$,

$$\alpha = 2R\cos\gamma - R^2 \tag{A.4}$$

Given this binomial expression for α, the α^n factor of Eq. A.3 can be further expanded via the binomial theorem.

$$(2R\cos\gamma - R^2)^n = \sum_{k=0}^{n} \binom{n}{k} (2R\cos\gamma)^{n-k} (-R^2)^k \tag{A.5}$$

Gathering like factors and substituting into Eq. A.3,

$$f(\alpha) = \sum_{n=0}^{\infty} \sum_{k=0}^{n} \frac{(-1)^k (2n)!}{2^{n+k} (n!)^2} \binom{n}{k} R^{n+k} \cos^{n-k}\gamma \tag{A.6}$$

Because it is ideal to gather terms with common exponents of R, the addends of Eq. A.6 are written as a function of $q = n + k$.

$$g(q,k) = \frac{(-1)^k (2q-2k)!}{2^q ((q-k)!)^2} \binom{q-k}{k} R^q \cos^{q-2k}\gamma \tag{A.7}$$

Expanding the binomial coefficient contains a factorial $(q-2k)!$, which imposes the constraint $q \geq 2k$ or $q/2 \geq k$ on the function since factorials are undefined for negative numbers. Since $q = n$ when $k = 0$ and

$k \geq 0$, q is applicable over the same range as n. Substituting the function $g(q, k)$ into Eq. A.6, expanding the binomial coefficient, and updating the summation bounds to match the new variables and constraints,

$$f(\alpha) = \sum_{q=0}^{\infty} \sum_{k=0}^{\lfloor q/2 \rfloor} \frac{(-1)^k (2q-2k)!}{2^q (q-k)! k! (q-2k)!} R^q \cos^{q-2k} \gamma \tag{A.8}$$

where $\lfloor \ \rfloor$ is the integer floor function.

Eq. A.9 groups factors that can be further simplified in later steps.

$$f(\alpha) = \sum_{q=0}^{\infty} R^q \left[\frac{1}{2^q q!} \sum_{k=0}^{\lfloor q/2 \rfloor} \left(\frac{q!}{(q-k)! k!} \right) \left(\frac{(2q-2k)!}{(q-2k)!} \cos^{q-2k} \gamma \right) (-1)^k \right] \tag{A.9}$$

The factorial ratio in the first set of parentheses is the familiar binomial coefficient. The second set of parentheses is the form for a recursive derivative with respect to $\cos \gamma$. Eq. A.10 substitutes these simplifications.

$$f(\alpha) = \sum_{q=0}^{\infty} R^q \left[\frac{1}{2^q q!} \sum_{k=0}^{q} \left(\begin{array}{c} q \\ k \end{array} \right) \left(\frac{d^q}{d(\cos \gamma)^q} \cos^{2q-2k} \gamma \right) (-1)^k \right] \tag{A.10}$$

Backing out the derivative removed the $(q-2k)!$ factorial, thus removing the constraint on the summation over k. Since differentiation is a linear operation, the differential operator can be pulled outside the summation over k.

$$f(\alpha) = \sum_{q=0}^{\infty} R^q \left[\frac{1}{2^q q!} \frac{d^q}{d(\cos \gamma)^q} \sum_{k=0}^{q} \left\{ \left(\begin{array}{c} q \\ k \end{array} \right) (\cos^2 \gamma)^{q-k} (-1)^k \right\} \right] \tag{A.11}$$

The quantity in curly braces is an expansion via the binomial theorem. Simplifying back into binomial form, the quantity in square brackets in Eq. A.12 is Rodrigues' formula for the Legendre polynomials.

$$f(\alpha) = \sum_{q=0}^{\infty} R^q \left[\frac{1}{2^q q!} \frac{d^q}{d(\cos \gamma)^q} (\cos^2 \gamma - 1)^q \right] \tag{A.12}$$

Finally, we arrive at the desired power series expansion in Eq. A.13.

$$\frac{1}{\sqrt{1 + R^2 - 2R \cos \gamma}} = f(\alpha) = \sum_{q=0}^{\infty} P_q(\cos \gamma) R^q \tag{A.13}$$

Appendix B

Preliminary Results

This section contains the final output of the "preliminary" MATLAB test script. Note: If neither "normalized" nor "unnormalized" are specified in the error nomenclature, error is defined as the difference between the results of the normalized and unnormalized versions of the respective algorithm. Otherwise, error is defined as the magnitude of the delta vector between each algorithm's acceleration vector and the Normalized Pines acceleration vector. See Section 4.2 for additional information.

All the results were within the acceptable tolerance of $O(10^-18)$ as defined in Section 4.2.

```
Maximum central body unnormalized Pines error: 2.65574e-019
at lat/lon: -30/-60
degree x order: 0x0

Maximum central body unnormalized Lear error: 4.8487e-019
at lat/lon: 0/-120
degree x order: 0x0

Maximum central body unnormalized Gottlieb error: 6.50521e-019
at lat/lon: -90/-150
degree x order: 0x0

Maximum central body normalized Lear error: 0
at lat/lon: -90/-150
degree x order: 0x0

Maximum central body normalized Gottlieb error: 0
at lat/lon: -90/-150
degree x order: 0x0

Maximum square unnormalized Lear error: 1.99033e-018
at lat/lon: 0/30
degree x order: 48x48

Maximum square unnormalized Gottlieb error: 2.31522e-018
at lat/lon: 30/120
degree x order: 45x45

Maximum non-square unnormalized Lear error: 2.28713e-018
at lat/lon: 30/-90
degree x order: 50x9

Maximum non-square unnormalized Gottlieb error: 3.03577e-018
at lat/lon: 0/180
degree x order: 50x3

Maximum square Pines error: 2.65574e-019
at lat/lon: -30/-150
degree x order: 1x1
```

```
Maximum square Lear error: 2.48422e-019
at lat/lon: 60/60
degree x order: 46x46

Maximum square Gottlieb error: 2.42435e-019
at lat/lon: 60/-90
degree x order: 9x9

Maximum non-square Pines error: 4.96844e-019
at lat/lon: -60/-60
degree x order: 50x13

Maximum non-square Lear error: 2.65574e-019
at lat/lon: -30/150
degree x order: 50x25

Maximum non-square Gottlieb error: 2.48422e-019
at lat/lon: -60/-60
degree x order: 50x35

Maximum square normalized Lear error 125: 9.00606e-019
at lat/lon: 30/-90
degree x order: 125x125

Maximum square normalized Gottlieb error 125: 2.82725e-018
at lat/lon: 0/-120
degree x order: 125x125

Maximum square normalized Lear error 150: 2.7959e-018
at lat/lon: 30/150
degree x order: 150x150

Maximum square normalized Gottlieb error 150: 2.89504e-018
at lat/lon: -30/150
degree x order: 150x150
```

Appendix C

MATLAB Code

This appendix contains the verification testing codes (*test_sh.m, test2.m*) as well as the coded implementations of the three algorithms themselves, each one unnormalized (*pines.m, lear.m, gottlieb.m*) and normalized (*pinesnorm.m, learnorm.m, gottliebnorm.m*). The verification tests were run with the LP150Q Spherical Harmonic lunar mass coefficients by Alex S. Konopliv from the Spherical Harmonics Gravity ASCII Data Record, contained in the file *jgl150q1.sha*. The file contains a header with the applicable gravitational parameter μ and reference radius a_{eq}. This file was obtained from

`http://pds-geosciences.wustl.edu/lunar01/lp-l-rss-5-gravity-v1/lp_1001/sha/jgl150q1.sha`

linked from the NASA Planetary Data System Geosciences Node webpage on June 7, 2010. A function for parsing the data file (*load_sha.m*) is also included.

Each of the algorithms started with the original code provided with each of the algorithms, except for the Pines implementations that were provided by DeMars [5]. Lines that were modified by the author, either to extend functionality or to translate properly from FORTRAN to MATLAB, are tagged with the comment "`%RAE`". Lines modified in order to implement normalization are tagged with the comment "`%norm`".

C.1 test_sh.m

```
clear
close
clc
format long g

[mu, r_eq, degree, order, c, s] = load_sha('jgl150q1.sha');

rmag = r_eq + 200;
rnp = eye(3);

cunnorm(1,1) = 1;
for n=1:degree
    ni = n+1;

    normfac = sqrt(2*n+1);

    for m=0:n
        mi = m+1;

        cunnorm(ni,mi) = c(ni,mi) * normfac;
        sunnorm(ni,mi) = s(ni,mi) * normfac;
        normfac = normfac/sqrt((n+m+1)*(n-m)/(1+(m==0)));
    end
end
```

```
for nlat = 1:7
    for nlon = 1:12
        lat = (nlat-4)*pi/6;
        lon = (nlon-6)*pi/6;
        r(1,1) = rmag*cos(lat)*cos(lon);
        r(2,1) = rmag*cos(lat)*sin(lon);
        r(3,1) = rmag*sin(lat);

        clc
        disp(nlat)
        disp(nlon)
        disp(r)

        %central body
        apines(nlat,nlon,1,:) = pines(mu, r_eq, r, cunnorm, sunnorm, 0, 0, rnp);
        alear(nlat,nlon,1,:) = lear(mu, r_eq, r, cunnorm, sunnorm, 0, 0, rnp);
        agott(nlat,nlon,1,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, 0, 0, ...
            rnp);
        apinesn(nlat,nlon,1,:) = pinesnorm(mu, r_eq, r, c, s, 0, 0, rnp);
        alearn(nlat,nlon,1,:) = learnorm(mu, r_eq, r, c, s, 0, 0, rnp);
        agottn(nlat,nlon,1,:) = gottliebnorm(mu, r_eq, r, c, s, 0, 0, rnp);

        %square models
        for nmax = 2:50
            apines(nlat,nlon,nmax,:) = pines(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            alear(nlat,nlon,nmax,:) = lear(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            agott(nlat,nlon,nmax,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            apinesn(nlat,nlon,nmax,:) = pinesnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);
            alearn(nlat,nlon,nmax,:) = learnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);
            agottn(nlat,nlon,nmax,:) = gottliebnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);
        end

        %nonsquare models with n=50
        for mmax = 0:49
            mi = mmax + 51;
            apines(nlat,nlon,mi,:) = pines(mu, r_eq, r, cunnorm, sunnorm, 50, ...
                mmax, rnp);
            alear(nlat,nlon,mi,:) = lear(mu, r_eq, r, cunnorm, sunnorm, 50, ...
                mmax, rnp);
            agott(nlat,nlon,mi,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, ...
                50, mmax, rnp);
            apinesn(nlat,nlon,mi,:) = pinesnorm(mu, r_eq, r, c, s, 50, mmax, ...
                rnp);
            alearn(nlat,nlon,mi,:) = learnorm(mu, r_eq, r, c, s, 50, mmax, rnp);
            agottn(nlat,nlon,mi,:) = gottliebnorm(mu, r_eq, r, c, s, 50, ...
                mmax, rnp);
        end

        %normalized at 125x125
        apines(nlat,nlon,101,:) = pines(mu, r_eq, r, cunnorm, sunnorm, 125, ...
            125, rnp);
        alear(nlat,nlon,101,:) = lear(mu, r_eq, r, cunnorm, sunnorm, 125, ...
            125, rnp);
        agott(nlat,nlon,101,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, 125, ...
            125, rnp);
        apinesn(nlat,nlon,101,:) = pinesnorm(mu, r_eq, r, c, s, 125, 125, rnp);
        alearn(nlat,nlon,101,:) = learnorm(mu, r_eq, r, c, s, 125, 125, rnp);
        agottn(nlat,nlon,101,:) = gottliebnorm(mu, r_eq, r, c, s, 125, 125, rnp);
```

```
%normalized at 150x150
apines(nlat,nlon,102,:) = pines(mu, r_eq, r, cunnorm, sunnorm, 150, ...
    150, rnp);
alear(nlat,nlon,102,:) = lear(mu, r_eq, r, cunnorm, sunnorm, 150, ...
    150, rnp);
agott(nlat,nlon,102,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, 150, ...
    150, rnp);
apinesn(nlat,nlon,102,:) = pinesnorm(mu, r_eq, r, c, s, 150, 150, rnp);
alearn(nlat,nlon,102,:) = learnorm(mu, r_eq, r, c, s, 150, 150, rnp);
agottn(nlat,nlon,102,:) = gottliebnorm(mu, r_eq, r, c, s, 150, 150, rnp);

% deviation of unnormalized from standard
for i = 1:102
    uleardiff(nlat,nlon,i) = norm(squeeze(...
        alear(nlat,nlon,i,:)-apinesn(nlat,nlon,i,:)));
    ugottdiff(nlat,nlon,i) = norm(squeeze(...
        agott(nlat,nlon,i,:)-apinesn(nlat,nlon,i,:)));
end

% deviation of normalized from unnormalized
for i = 1:102
    pinesdiff(nlat,nlon,i) = norm(squeeze(...
        apines(nlat,nlon,i,:)-apinesn(nlat,nlon,i,:)));
    leardiff(nlat,nlon,i) = norm(squeeze(...
        alear(nlat,nlon,i,:)-alearn(nlat,nlon,i,:)));
    gottdiff(nlat,nlon,i) = norm(squeeze(...
        agott(nlat,nlon,i,:)-agottn(nlat,nlon,i,:)));
end

% deviation of normalized from standard
nleardiff(nlat,nlon,1) = norm(squeeze(...
    alearn(nlat,nlon,101,:)-apinesn(nlat,nlon,101,:)));
ngottdiff(nlat,nlon,1) = norm(squeeze(...
    agottn(nlat,nlon,101,:)-apinesn(nlat,nlon,101,:)));

nleardiff(nlat,nlon,2) = norm(squeeze(...
    alearn(nlat,nlon,102,:)-apinesn(nlat,nlon,102,:)));
ngottdiff(nlat,nlon,2) = norm(squeeze(...
    agottn(nlat,nlon,102,:)-apinesn(nlat,nlon,102,:)));

[maxucpinesdiff(nlat,nlon) maxucpinesdiffn(nlat,nlon)] = ...
    max(squeeze(pinesdiff(nlat,nlon,1)));
[maxucleardiff(nlat,nlon) maxucleardiffn(nlat,nlon)] = ...
    max(squeeze(uleardiff(nlat,nlon,1)));
[maxucgottdiff(nlat,nlon) maxucgottdiffn(nlat,nlon)] = ...
    max(squeeze(ugottdiff(nlat,nlon,1)));

[maxcleardiff(nlat,nlon) maxcleardiffn(nlat,nlon)] = ...
    max(squeeze(leardiff(nlat,nlon,1)));
[maxcgottdiff(nlat,nlon) maxcgottdiffn(nlat,nlon)] = ...
    max(squeeze(gottdiff(nlat,nlon,1)));

[maxuleardiff(nlat,nlon,1) maxuleardiffn(nlat,nlon,1)] = ...
    max(squeeze(uleardiff(nlat,nlon,2:50)));
[maxugottdiff(nlat,nlon,1) maxugottdiffn(nlat,nlon,1)] = ...
    max(squeeze(ugottdiff(nlat,nlon,2:50)));

[maxuleardiff(nlat,nlon,2) maxuleardiffn(nlat,nlon,2)] = ...
    max(squeeze(uleardiff(nlat,nlon,51:100)));
[maxugottdiff(nlat,nlon,2) maxugottdiffn(nlat,nlon,2)]= ...
    max(squeeze(ugottdiff(nlat,nlon,51:100)));

[maxpinesdiff(nlat,nlon,1) maxpinesdiffn(nlat,nlon,1)] = ...
    max(squeeze(pinesdiff(nlat,nlon,2:50)));
[maxleardiff(nlat,nlon,1) maxleardiffn(nlat,nlon,1)] = ...
```

```
            max(squeeze(leardiff(nlat,nlon,2:50)));
        [maxgottdiff(nlat,nlon,1) maxgottdiffn(nlat,nlon,1)] = ...
            max(squeeze(gottdiff(nlat,nlon,2:50)));

        [maxpinesdiff(nlat,nlon,2) maxpinesdiffn(nlat,nlon,2)] = ...
            max(squeeze(pinesdiff(nlat,nlon,51:100)));
        [maxleardiff(nlat,nlon,2) maxleardiffn(nlat,nlon,2)] = ...
            max(squeeze(leardiff(nlat,nlon,51:100)));
        [maxgottdiff(nlat,nlon,2) maxgottdiffn(nlat,nlon,2)] = ...
            max(squeeze(gottdiff(nlat,nlon,51:100)));

        [maxnleardiff(nlat,nlon,1) maxnleardiffn(nlat,nlon,1)] = ...
            max(squeeze(nleardiff(nlat,nlon,1)));
        [maxngottdiff(nlat,nlon,1) maxngottdiffn(nlat,nlon,1)] = ...
            max(squeeze(ngottdiff(nlat,nlon,1)));

        [maxnleardiff(nlat,nlon,2) maxnleardiffn(nlat,nlon,2)] = ...
            max(squeeze(nleardiff(nlat,nlon,2)));
        [maxngottdiff(nlat,nlon,2) maxngottdiffn(nlat,nlon,2)] = ...
            max(squeeze(ngottdiff(nlat,nlon,2)));
    end
end

maxuleardiffn(:,:,2) = maxuleardiffn(:,:,2) + 50;
maxugottdiffn(:,:,2) = maxugottdiffn(:,:,2) + 50;
maxpinesdiffn(:,:,2) = maxpinesdiffn(:,:,2) + 50;
maxleardiffn(:,:,2) = maxleardiffn(:,:,2) + 50;
maxgottdiffn(:,:,2) = maxgottdiffn(:,:,2) + 50;

%max deviation
[x maxucpinesdiffi] = max(maxucpinesdiff(1:84));
[x maxucleardiffi] = max(maxucleardiff(1:84));
[x maxucgottdiffi] = max(maxucgottdiff(1:84));

[x maxcleardiffi] = max(maxcleardiff(1:84));
[x maxcgottdiffi] = max(maxcgottdiff(1:84));

[x maxuleardiff1] = max(maxuleardiff(1:84));
[x maxugottdiff1] = max(maxugottdiff(1:84));

[x maxuleardiff2] = max(maxuleardiff(85:168));
[x maxugottdiff2] = max(maxugottdiff(85:168));

[x maxpinesdiff1] = max(maxpinesdiff(1:84));
[x maxleardiff1] = max(maxleardiff(1:84));
[x maxgottdiff1] = max(maxgottdiff(1:84));

[x maxpinesdiff2] = max(maxpinesdiff(85:168));
[x maxleardiff2] = max(maxleardiff(85:168));
[x maxgottdiff2] = max(maxgottdiff(85:168));

[x maxnleardiff1] = max(nleardiff(1:84));
[x maxngottdiff1] = max(ngottdiff(1:84));

[x maxnleardiff2] = max(nleardiff(85:168));
[x maxngottdiff2] = max(ngottdiff(85:168));

maxuleardiff2 = maxuleardiff2 + 84;
maxugottdiff2 = maxugottdiff2 + 84;
maxpinesdiff2 = maxpinesdiff2 + 84;
maxleardiff2 = maxleardiff2 + 84;
maxgottdiff2 = maxgottdiff2 + 84;
maxnleardiff2 = maxnleardiff2 + 84;
maxngottdiff2 = maxngottdiff2 + 84;
```

```
nlat = 1 + mod(maxucpinesdiffi-1,7);
nlon = 1 + (maxucpinesdiffi - nlat)/7;
fprintf('Maximum central body unnormalized Pines error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxucpinesdiff(maxucpinesdiffi), (nlat-4)*30, (nlon-6)*30, 0, 0)

nlat = 1 + mod(maxucleardiffi-1,7);
nlon = 1 + (maxucleardiffi - nlat)/7;
fprintf('Maximum central body unnormalized Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxucleardiff(maxucleardiffi), (nlat-4)*30, (nlon-6)*30, 0, 0)

nlat = 1 + mod(maxucgottdiffi-1,7);
nlon = 1 + (maxucgottdiffi - nlat)/7;
fprintf('Maximum central body unnormalized Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxucgottdiff(maxucgottdiffi), (nlat-4)*30, (nlon-6)*30, 0, 0)

nlat = 1 + mod(maxcleardiffi-1,7);
nlon = 1 + (maxcleardiffi - nlat)/7;
fprintf('Maximum central body normalized Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxcleardiff(maxcleardiffi), (nlat-4)*30, (nlon-6)*30, 0, 0)

nlat = 1 + mod(maxcgottdiffi-1,7);
nlon = 1 + (maxcgottdiffi - nlat)/7;
fprintf('Maximum central body normalized Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxcgottdiff(maxcgottdiffi), (nlat-4)*30, (nlon-6)*30, 0, 0)

nlat = 1 + mod(maxuleardiff1-1,7);
nlon = 1 + (maxuleardiff1 - nlat)/7;
fprintf('Maximum square unnormalized Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxuleardiff(maxuleardiff1), (nlat-4)*30, (nlon-6)*30, ...
    maxuleardiffn(nlat,nlon,1), maxuleardiffn(nlat,nlon,1))

nlat = 1 + mod(maxugottdiff1-1,7);
nlon = 1 + (maxugottdiff1 - nlat)/7;
fprintf('Maximum square unnormalized Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxugottdiff(maxugottdiff1), (nlat-4)*30, (nlon-6)*30, ...
    maxugottdiffn(nlat,nlon,1), maxugottdiffn(nlat,nlon,1))

nlat = 1 + mod(maxuleardiff2-85,7);
nlon = 1 + (maxuleardiff2 - 84 - nlat)/7;
fprintf('Maximum non-square unnormalized Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxuleardiff(maxuleardiff2), (nlat-4)*30, (nlon-6)*30, 50, ...
    maxuleardiffn(nlat,nlon,2)-51)

nlat = 1 + mod(maxugottdiff2-85,7);
nlon = 1 + (maxugottdiff2 - 84 - nlat)/7;
fprintf('Maximum non-square unnormalized Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxugottdiff(maxugottdiff2), (nlat-4)*30, (nlon-6)*30, 50, ...
    maxugottdiffn(nlat,nlon,2)-51)

nlat = 1 + mod(maxpinesdiff1-1,7);
nlon = 1 + (maxpinesdiff1 - nlat)/7;
fprintf('Maximum square Pines error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxpinesdiff(maxpinesdiff1), (nlat-4)*30, (nlon-6)*30, ...
    maxpinesdiffn(nlat,nlon,1), maxpinesdiffn(nlat,nlon,1))
```

```
nlat = 1 + mod(maxleardiff1-1,7);
nlon = 1 + (maxleardiff1 - nlat)/7;
fprintf('Maximum square Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxleardiff(maxleardiff1), (nlat-4)*30, (nlon-6)*30, ...
    maxleardiffn(nlat,nlon,1), maxleardiffn(nlat,nlon,1))

nlat = 1 + mod(maxgottdiff1-1,7);
nlon = 1 + (maxgottdiff1 - nlat)/7;
fprintf('Maximum square Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxgottdiff(maxgottdiff1), (nlat-4)*30, (nlon-6)*30, ...
    maxgottdiffn(nlat,nlon,1), maxgottdiffn(nlat,nlon,1))

nlat = 1 + mod(maxpinesdiff2-85,7);
nlon = 1 + (maxpinesdiff2 - 84 - nlat)/7;
fprintf('Maximum non-square Pines error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxpinesdiff(maxpinesdiff2), (nlat-4)*30, (nlon-6)*30, 50, ...
    maxpinesdiffn(nlat,nlon,2)-51)

nlat = 1 + mod(maxleardiff2-85,7);
nlon = 1 + (maxleardiff2 - 84 - nlat)/7;
fprintf('Maximum non-square Lear error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxleardiff(maxleardiff2), (nlat-4)*30, (nlon-6)*30, 50, ...
    maxleardiffn(nlat,nlon,2)-51)

nlat = 1 + mod(maxgottdiff2-85,7);
nlon = 1 + (maxgottdiff2 - 84 - nlat)/7;
fprintf('Maximum non-square Gottlieb error: ')
fprintf('%g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxgottdiff(maxgottdiff2), (nlat-4)*30, (nlon-6)*30, 50, ...
    maxgottdiffn(nlat,nlon,2)-51)

nlat = 1 + mod(maxnleardiff1-1,7);
nlon = 1 + (maxnleardiff1 - nlat)/7;
fprintf('Maximum square normalized Lear error 125:')
fprintf(' %g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxnleardiff(maxleardiff1), (nlat-4)*30, (nlon-6)*30, 125, 125)

nlat = 1 + mod(maxngottdiff1-1,7);
nlon = 1 + (maxngottdiff1 - nlat)/7;
fprintf('Maximum square normalized Gottlieb error 125:')
fprintf(' %g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxngottdiff(maxngottdiff1), (nlat-4)*30, (nlon-6)*30, 125, 125)

nlat = 1 + mod(maxnleardiff2-85,7);
nlon = 1 + (maxnleardiff2 - 84 - nlat)/7;
fprintf('Maximum square normalized Lear error 150:')
fprintf(' %g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxnleardiff(maxnleardiff2), (nlat-4)*30, (nlon-6)*30, 150, 150)

nlat = 1 + mod(maxngottdiff2-85,7);
nlon = 1 + (maxngottdiff2 - 84 - nlat)/7;
fprintf('Maximum square normalized Gottlieb error 150:')
fprintf(' %g\nat lat/lon: %d/%d\ndegree x order: %dx%d\n\n', ...
    maxngottdiff(maxngottdiff2), (nlat-4)*30, (nlon-6)*30, 150, 150)
```

C.2 test2.m

```
clear
close
clc
```

```
format long g

[mu, r_eq, degree, order, c, s] = load_sha('jgl150q1.sha');

rmag = r_eq + 200;
rnp = eye(3);

cunnorm(1,1) = 1;
for n=1:degree
    ni = n+1;

    normfac = sqrt(2*n+1);

    for m=0:n
        mi = m+1;

        cunnorm(ni,mi) = c(ni,mi) * normfac;
        sunnorm(ni,mi) = s(ni,mi) * normfac;
        normfac = normfac*sqrt((1+(m==0))/((n+m+1)*(n-m)));
    end
end

for nlat = 1:7
    for nlon = 1:12
        lat = (nlat-4)*pi/6;
        lon = (nlon-6)*pi/6;
        r(1,1) = rmag*cos(lat)*cos(lon);
        r(2,1) = rmag*cos(lat)*sin(lon);
        r(3,1) = rmag*sin(lat);

        clc
        disp(nlat)
        disp(nlon)
        disp(r)

        %square models
        for nmax = 2:150
            apines(nlat,nlon,nmax,:) = pines(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            alear(nlat,nlon,nmax,:) = lear(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            agott(nlat,nlon,nmax,:) = gottlieb(mu, r_eq, r, cunnorm, sunnorm, ...
                nmax, nmax, rnp);
            apinesn(nlat,nlon,nmax,:) = pinesnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);
            alearn(nlat,nlon,nmax,:) = learnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);
            agottn(nlat,nlon,nmax,:) = gottliebnorm(mu, r_eq, r, c, s, nmax, ...
                nmax, rnp);

            uleardiff(nlat,nlon,nmax) = norm(squeeze(...
                alear(nlat,nlon,nmax,:)-apinesn(nlat,nlon,nmax,:)));
            ugottdiff(nlat,nlon,nmax) = norm(squeeze(...
                agott(nlat,nlon,nmax,:)-apinesn(nlat,nlon,nmax,:)));
            nleardiff(nlat,nlon,nmax) = norm(squeeze(...
                alearn(nlat,nlon,nmax,:)-apinesn(nlat,nlon,nmax,:)));
            ngottdiff(nlat,nlon,nmax) = norm(squeeze(...
                agottn(nlat,nlon,nmax,:)-apinesn(nlat,nlon,nmax,:)));

            pinesdiff(nlat,nlon,nmax) = norm(squeeze(...
                apines(nlat,nlon,nmax,:)-apinesn(nlat,nlon,nmax,:)));
            leardiff(nlat,nlon,nmax) = norm(squeeze(...
                alear(nlat,nlon,nmax,:)-alearn(nlat,nlon,nmax,:)));
            gottdiff(nlat,nlon,nmax) = norm(squeeze(...
                agott(nlat,nlon,nmax,:)-agottn(nlat,nlon,nmax,:)));
        end
    end
end
```

C.3 load_sha.m

```
function [mu, r_eq, degree, order, c, s] = load_sha(filename)
% open file for reading (MATLAB default mode)
f = fopen(filename);

% read header line
r_eq = fscanf(f, '%g,', 1);
mu = fscanf(f, '%g,', 1);
degree = fscanf(f, '%*g, %d,', 1);
order = fscanf(f, '%d,', 1);
normalized = fscanf(f, '%d,', 1);

% c at degree = 0 and order = 0 is defined as 1
c(1,1) = 1;

if (normalized > 1)
    fprintf('This file uses a nonstandard normalization.\n\n')
    degree = -1;
    order = -1;
    return
end

% iterate from 'degree' 1 to degree
for n = 1:degree
    ni = n+1; % the value of n to be used as an array index

    % if normalized flag is not present, calculate normalization
    % factor at order = 0 using recursive method defined by Adamo
    if (normalized==0)
        % Because the files load in coefficients across rows and
        % then down, we seed each row by using factor for order 0
        normfac = 1/sqrt(2*n+1);
    end

    %iterate from 'order' 0 to n
    for m = 0:n
        mi = m+1; % the value of m to be used as an array index

        % read one line
        c(ni,mi) = fscanf(f, '%*g, %*g %*d, %*d, %g,', 1);
        s(ni,mi) = fscanf(f, '%g,', 1);
        % if normalized flag is not present, apply normalization
        if (normalized==0)
            c(ni,mi) = c(ni,mi) * normfac;
            s(ni,mi) = s(ni,mi) * normfac;
            % calculate factor for next order
            normfac = normfac*sqrt((n+m+1)*(n-m)/(1+(m==0)));
        end
    end
end

fclose(f);
return
```

C.4 pines.m

```
function accel = pines(MU, REQ, x, CNM, SNM, NMAX, MMAX, rnp)
```

```
R_F = rnp*x; %RAE
RMAG = norm(R_F);
S    = R_F(1)/RMAG;
T    = R_F(2)/RMAG;
U    = R_F(3)/RMAG;

ANM = zeros(NMAX+3,NMAX+3); %RAE
ANM(1,1) = 1;
for M = 0:NMAX+2 %RAE
    M_A = M + 1;

    if(M ~= 0) % DIAGONAL RECURSION
        ANM(M_A,M_A) = (2*M-1)*ANM(M_A-1,M_A-1);
    end

    if(M ~= NMAX+2) % FIRST OFF-DIAGONAL RECURSION %RAE
        ANM(M_A+1,M_A) = (2*M+1)*U*ANM(M_A,M_A);
    end

    if(M < NMAX+1) % COLUMN RECURSION %RAE
        for N = M+2:NMAX+2 %RAE
            N_A          = N + 1;

            ANM(N_A,M_A) = ((2*N-1)*U*ANM(N_A-1,M_A)-(N+M-1)*ANM(N_A-2,M_A))/ ...
                (N-M);
        end
    end
end

RM = zeros(MMAX+2,1); %RAE
IM = zeros(MMAX+2,1); %RAE
RM(1) = 0.0D0;
IM(1) = 0.0D0;
RM(2) = 1.0D0; %RAE
IM(2) = 0.0D0; %RAE
for M = 1:MMAX %RAE
    M_RI     = M + 2; %RAE
    RM(M_RI) = S*RM(M_RI-1) - T*IM(M_RI-1);
    IM(M_RI) = S*IM(M_RI-1) + T*RM(M_RI-1);
end

RHO  = (MU)/(REQ*RMAG);
RHOP = (REQ)/(RMAG);
G1   = 0.0D0;
G2   = 0.0D0;
G3   = 0.0D0;
G4   = 0.0D0;
for N = 0:NMAX
    N_A    = N + 1;

    G1TEMP = 0.0D0;
    G2TEMP = 0.0D0;
    G3TEMP = 0.0D0;
    G4TEMP = 0.0D0;

    if (N>MMAX) %RAE
        nmodel=MMAX; %RAE
    else %RAE
        nmodel=N; %RAE
    end %RAE
    for M = 0:nmodel %RAE
        M_A    = M + 1;
        M_RI   = M + 2; %RAE

        DNM = CNM(N_A,M_A)*RM(M_RI)   + SNM(N_A,M_A)*IM(M_RI);
        ENM = CNM(N_A,M_A)*RM(M_RI-1) + SNM(N_A,M_A)*IM(M_RI-1);
        FNM = SNM(N_A,M_A)*RM(M_RI-1) - CNM(N_A,M_A)*IM(M_RI-1);

        G1TEMP = G1TEMP + ANM(N_A,M_A)*(M)*ENM;
        G2TEMP = G2TEMP + ANM(N_A,M_A)*(M)*FNM;
        G3TEMP = G3TEMP + ANM(N_A,M_A+1)*DNM;
        G4TEMP = G4TEMP + ((N+M+1)*ANM(N_A,M_A) + U*ANM(N_A,M_A+1))*DNM;
    end
    RHO = RHOP*RHO;
    G1  = G1 + RHO*G1TEMP;
    G2  = G2 + RHO*G2TEMP;
    G3  = G3 + RHO*G3TEMP;
    G4  = G4 + RHO*G4TEMP;
end

G_F = [G1 - G4*S;G2 - G4*T;G3 - G4*U];

accel=rnp'*G_F; %RAE
return
```

C.5 pinesnorm.m

```
function accel = pinesnorm(MU, REQ, x, CNM, SNM, NMAX, MMAX, rnp)
R_F = rnp*x; %RAE
RMAG = norm(R_F);
S    = R_F(1)/RMAG;
T    = R_F(2)/RMAG;
U    = R_F(3)/RMAG;

ANM = zeros(NMAX+3,NMAX+3); %RAE
ANM(1,1) = sqrt(2.0D0); %norm
for M = 0:NMAX+2 %RAE
    M_A = M + 1;

    if(M ~= 0) % DIAGONAL RECURSION
        ANM(M_A,M_A) = sqrt(1+(1/(2*M)))*ANM(M_A-1,M_A-1); %norm
    end

    if(M ~= NMAX+2) % FIRST OFF-DIAGONAL RECURSION %RAE
        ANM(M_A+1,M_A) = sqrt(2*M+3)*U*ANM(M_A,M_A); %norm
    end

    if(M < NMAX+1) % COLUMN RECURSION %RAE
        for N = M+2:NMAX+2 %RAE
            N_A          = N + 1;

            ALPHA_NUM    = (2*N+1)*(2*N-1);
            ALPHA_DEN    = (N-M)*(N+M);
            ALPHA        = sqrt(ALPHA_NUM/ALPHA_DEN);
            BETA_NUM     = (2*N+1)*(N-M-1)*(N+M-1);
            BETA_DEN     = (2*N-3)*(N+M)*(N-M);
            BETA         = sqrt(BETA_NUM/BETA_DEN);
            ANM(N_A,M_A) = ALPHA*U*ANM(N_A-1,M_A) - BETA*ANM(N_A-2,M_A); %norm
        end
    end
end
for N = 0:NMAX+2 %RAE
    N_A      = N + 1;

    ANM(N_A,1) = ANM(N_A,1)*sqrt(0.5D0); %norm
end

RM = zeros(MMAX+2,1); %RAE
IM = zeros(MMAX+2,1); %RAE
```

```
RM(1) = 0.0D0;
IM(1) = 0.0D0;
RM(2) = 1.0D0; %RAE
IM(2) = 0.0D0; %RAE
for M = 1:MMAX %RAE
    M_RI     = M + 2; %RAE
    RM(M_RI) = S*RM(M_RI-1) - T*IM(M_RI-1);
    IM(M_RI) = S*IM(M_RI-1) + T*RM(M_RI-1);
end

RHO  = (MU)/(REQ*RMAG);
RHOP = (REQ)/(RMAG);
G1   = 0.0D0;
G2   = 0.0D0;
G3   = 0.0D0;
G4   = 0.0D0;
for N = 0:NMAX
    N_A    = N + 1;

    G1TEMP = 0.0D0;
    G2TEMP = 0.0D0;
    G3TEMP = 0.0D0;
    G4TEMP = 0.0D0;

    SM     = 0.5D0;
    if (N>MMAX) %RAE
        nmodel=MMAX; %RAE
    else %RAE
        nmodel=N; %RAE
    end %RAE
    for M = 0:nmodel %RAE
        M_A    = M + 1;
        M_RI   = M + 2; %RAE

        DNM = CNM(N_A,M_A)*RM(M_RI)   + SNM(N_A,M_A)*IM(M_RI);
        ENM = CNM(N_A,M_A)*RM(M_RI-1) + SNM(N_A,M_A)*IM(M_RI-1);
        FNM = SNM(N_A,M_A)*RM(M_RI-1) - CNM(N_A,M_A)*IM(M_RI-1);

        ALPHA  = sqrt(SM*(N-M)*(N+M+1)); %norm

        G1TEMP = G1TEMP + ANM(N_A,M_A)*(M)*ENM;
        G2TEMP = G2TEMP + ANM(N_A,M_A)*(M)*FNM;
        G3TEMP = G3TEMP + ALPHA*ANM(N_A,M_A+1)*DNM; %norm
        G4TEMP = G4TEMP + ((N+M+1)*ANM(N_A,M_A)+ALPHA*U*ANM(N_A,M_A+1))*DNM;%norm
        if(M == 0); SM = 1.0D0; end; %norm
    end
    RHO = RHOP*RHO;
    G1  = G1 + RHO*G1TEMP;
    G2  = G2 + RHO*G2TEMP;
    G3  = G3 + RHO*G3TEMP;
    G4  = G4 + RHO*G4TEMP;
end

G_F = [G1 - G4*S;G2 - G4*T;G3 - G4*U];

accel=rnp'*G_F; %RAE
return
```

C.6 lear.m

```
function ag = lear(mu, rbar, r, c, s, nmax, mmax, rnp)
for n=2:nmax %RAE
    pnm(n-1,n)=0;
end
```

```
rgr=rnp*r; %RAE
e1=rgr(1)^2+rgr(2)^2;
r2=e1+rgr(3)^2;
absr=sqrt(r2);
r1=sqrt(e1);
sphi=rgr(3)/absr;
cphi=r1/absr;
sm(1)=0;
cm(1)=1;
if (r1~=0)
    sm(1)=rgr(2)/r1;
    cm(1)=rgr(1)/r1;
end
rb(1)=rbar/absr;
rb(2)=rb(1)^2;
sm(2)=2*cm(1)*sm(1);
cm(2)=2*cm(1)^2-1;
pn(1)=sphi;
pn(2)=(3*sphi^2-1)/2;
ppn(1)=1;
ppn(2)=3*sphi;
pnm(1,1)=1;
pnm(2,2)=3*cphi;
pnm(2,1)=ppn(2);
ppnm(1,1)=-sphi;
ppnm(2,2)=-6*sphi*cphi;
ppnm(2,1)=3-6*sphi^2;
if (nmax>=3) %RAE
    for n=3:nmax %RAE
        nm1=n-1;
        nm2=n-2;
        rb(n)=rb(nm1)*rb(1);
        sm(n)=2*cm(1)*sm(nm1)-sm(nm2);
        cm(n)=2*cm(1)*cm(nm1)-cm(nm2);
        e1=2*n-1;
        pn(n)=(e1*sphi*pn(nm1)-nm1*pn(nm2))/n;
        ppn(n)=sphi*ppn(nm1)+n*pn(nm1);
        pnm(n,n)=e1*cphi*pnm(nm1,nm1);
        ppnm(n,n)=-n*sphi*pnm(n,n);
    end
    for n=3:nmax %RAE
        nm1=n-1;
        e1=(2*n-1)*sphi;
        e2=-n*sphi;
        for m=1:nm1
            e3=pnm(nm1,m);
            e4=n+m;
            e5=(e1*e3-(e4-1)*pnm(n-2,m))/(n-m);
            pnm(n,m)=e5;
            ppnm(n,m)=e2*e5+e4*e3;
        end
    end
end
asph(1)=-1;
asph(3)=0;
for n=2:nmax %RAE
    ni=n+1; %RAE
    e1=c(ni,1)*rb(n); %RAE
    asph(1)=asph(1)-(n+1)*e1*pn(n);
    asph(3)=asph(3)+e1*ppn(n);
end
asph(3)=cphi*asph(3);
t1=0;
t3=0;
asph(2)=0;
```

```
for n=2:nmax %RAE
    ni=n+1; %RAE
    e1=0;
    e2=0;
    e3=0;
    if (n>mmax) %RAE
        nmodel=mmax; %RAE
    else %RAE
        nmodel=n; %RAE
    end %RAE
    for m=1:nmodel %RAE
        mi=m+1; %RAE
        tsnm=s(ni,mi); %RAE
        tcnm=c(ni,mi); %RAE
        tsm=sm(m);
        tcm=cm(m);
        tpnm=pnm(n,m);
        e4=tsnm*tsm+tcnm*tcm;
        e1=e1+e4*tpnm;
        e2=e2+m*(tsnm*tcm-tcnm*tsm)*tpnm;
        e3=e3+e4*ppnm(n,m);
    end
    t1=t1+(n+1)*rb(n)*e1;
    asph(2)=asph(2)+rb(n)*e2;
    t3=t3+rb(n)*e3;
end
e4=mu/r2;
asph(1)=e4*(asph(1)-cphi*t1);
asph(2)=e4*asph(2);
asph(3)=e4*(asph(3)+t3);
e5=asph(1)*cphi-asph(3)*sphi;
agr(1,1)=e5*cm(1)-asph(2)*sm(1); %RAE
agr(2,1)=e5*sm(1)+asph(2)*cm(1); %RAE
agr(3,1)=asph(1)*sphi+asph(3)*cphi; %RAE
ag=rnp'*agr; %RAE
return
```

C.7 learnorm.m

```
function ag = learnorm(mu, rbar, r, c, s, nmax, mmax, rnp)
for n = 2:nmax %RAE
    norm1(n) = sqrt((2*n+1)/(2*n-1)); %RAE
    norm2(n) = sqrt((2*n+1)/(2*n-3)); %RAE
    norm11(n) = sqrt((2*n+1)/(2*n))/(2*n-1); %RAE
    for m = 1:n %RAE
        norm1m(n,m) = sqrt((n-m)*(2*n+1)/((n+m)*(2*n-1))); %RAE
        norm2m(n,m) = sqrt((n-m)*(n-m-1)*(2*n+1)/((n+m)*(n+m-1)*(2*n-3))); %RAE
    end %RAE
end %RAE

for n=2:nmax %RAE
    pnm(n-1,n)=0;
end
rgr=rnp*r; %RAE
e1=rgr(1)^2+rgr(2)^2;
r2=e1+rgr(3)^2;
absr=sqrt(r2);
r1=sqrt(e1);
sphi=rgr(3)/absr;
cphi=r1/absr;
sm(1)=0;
cm(1)=1;
if (r1~=0)
    sm(1)=rgr(2)/r1;
    cm(1)=rgr(1)/r1;
end
rb(1)=rbar/absr;
rb(2)=rb(1)^2;
sm(2)=2*cm(1)*sm(1);
cm(2)=2*cm(1)^2-1;
root3=sqrt(3); %RAE
root5=sqrt(5); %RAE
pn(1)=root3*sphi; %norm
pn(2)=root5*(3*sphi^2-1)/2; %norm
ppn(1)=root3; %norm
ppn(2)=root5*3*sphi; %norm
pnm(1,1)=root3; %norm
pnm(2,2)=root5*root3*cphi/2; %norm
pnm(2,1)=root5*root3*sphi; %norm %RAE
ppnm(1,1)=-root3*sphi; %norm
ppnm(2,2)=-root3*root5*sphi*cphi; %norm
ppnm(2,1)=root5*root3*(1-2*sphi^2); %norm
if (nmax>=3) %RAE
    for n=3:nmax %RAE
        nm1=n-1;
        nm2=n-2;
        rb(n)=rb(nm1)*rb(1);
        sm(n)=2*cm(1)*sm(nm1)-sm(nm2);
        cm(n)=2*cm(1)*cm(nm1)-cm(nm2);
        e1=2*n-1;
        pn(n)=(e1*sphi*norm1(n)*pn(nm1)-nm1*norm2(n)*pn(nm2))/n; %norm
        ppn(n)=norm1(n)*(sphi*ppn(nm1)+n*pn(nm1)); %norm
        pnm(n,n)=e1*cphi*norm11(n)*pnm(nm1,nm1); %norm
        ppnm(n,n)=-n*sphi*pnm(n,n);
    end
    for n=3:nmax %RAE
        nm1=n-1;
        e1=(2*n-1)*sphi;
        e2=-n*sphi;
        for m=1:nm1
            e3=norm1m(n,m)*pnm(nm1,m); %norm
            e4=n+m;
            e5=(e1*e3-(e4-1)*norm2m(n,m)*pnm(n-2,m))/(n-m); %norm
            pnm(n,m)=e5;
            ppnm(n,m)=e2*e5+e4*e3;
        end
    end
end
asph(1)=-1;
asph(3)=0;
for n=2:nmax %RAE
    ni=n+1; %RAE
    e1=c(ni,1)*rb(n); %RAE
    asph(1)=asph(1)-(n+1)*e1*pn(n);
    asph(3)=asph(3)+e1*ppn(n);
end
asph(3)=cphi*asph(3);
t1=0;
t3=0;
asph(2)=0;
for n=2:nmax %RAE
    ni=n+1; %RAE
    e1=0;
    e2=0;
    e3=0;
    if (n>mmax) %RAE
        nmodel=mmax; %RAE
    else %RAE
        nmodel=n; %RAE
    end %RAE
    for m=1:nmodel %RAE
        mi=m+1; %RAE
        tsnm=s(ni,mi); %RAE
        tcnm=c(ni,mi); %RAE
        tsm=sm(m);
        tcm=cm(m);
        tpnm=pnm(n,m);
        e4=tsnm*tsm+tcnm*tcm;
        e1=e1+e4*tpnm;
        e2=e2+m*(tsnm*tcm-tcnm*tsm)*tpnm;
        e3=e3+e4*ppnm(n,m);
    end
    t1=t1+(n+1)*rb(n)*e1;
    asph(2)=asph(2)+rb(n)*e2;
    t3=t3+rb(n)*e3;
end
e4=mu/r2;
asph(1)=e4*(asph(1)-cphi*t1);
asph(2)=e4*asph(2);
asph(3)=e4*(asph(3)+t3);
e5=asph(1)*cphi-asph(3)*sphi;
agr(1,1)=e5*cm(1)-asph(2)*sm(1); %RAE
agr(2,1)=e5*sm(1)+asph(2)*cm(1); %RAE
agr(3,1)=asph(1)*sphi+asph(3)*cphi; %RAE
ag=rnp'*agr; %RAE
return
```

C.8 gottlieb.m

```
function accel = gottlieb(mu, re, xin, c, s, nax, max, rnp)
x=rnp*xin; %RAE
r = sqrt(x(1)^2+x(2)^2+x(3)^2);
ri=1/r;
xor=x(1)*ri;
yor=x(2)*ri;
zor=x(3)*ri;
ep=zor;
reor=re*ri;
reorn=reor;
muor2=mu*ri*ri;

p(1,1) = 1; %RAE
p(1,2) = 0; %RAE
p(1,3) = 0; %RAE
p(2,2) = 1; %RAE
p(2,3) = 0; %RAE
p(2,4) = 0; %RAE
for n = 2:nax %RAE
    ni = n+1; %RAE
    p(ni,ni) = p(n,n)*(2*n-1); %RAE
    p(ni,ni+1) = 0; %RAE
    p(ni,ni+2) = 0; %RAE
end

ctil(1)=1; %RAE
stil(1)=0; %RAE
ctil(2)=xor; %RAE
stil(2)=yor; %RAE
sumh=0;
sumgm=1;
sumj=0;
sumk=0;
```

```
p(2,1) = ep; %RAE
for n=2:nax
    ni=n+1; %RAE
    reorn=reorn*reor;
    n2m1=n+n-1;
    nm1=n-1;
    np1=n+1;

    p(ni,n) = ep*p(ni,ni); %RAE
    p(ni,1) = (n2m1*ep*p(n,1)-nm1*p(nm1,1))/n; %RAE
    p(ni,2) = (n2m1*ep*p(n,2)-n*p(nm1,2))/(nm1); %RAE

    sumhn=p(ni,2)*c(ni,1); %RAE
    sumgmn=p(ni,1)*c(ni,1)*np1; %RAE
    if (max>0)
        for m = 2:n-2
            mi = m+1; %RAE
            p(ni,mi) = (n2m1*ep*p(n,mi)-(nm1+m)*p(nm1,mi))/(n-m); %RAE
        end
        sumjn=0;
        sumkn=0;
        ctil(ni)=ctil(2)*ctil(ni-1)-stil(2)*stil(ni-1); %RAE
        stil(ni)=stil(2)*ctil(ni-1)+ctil(2)*stil(ni-1); %RAE
        if(n<max)
            lim=n;
        else
            lim=max;
        end
        for m=1:lim
            mi=m+1; %RAE
            mm1=mi-1; %RAE
            mp1=mi+1; %RAE

            mxpnm=m*p(ni,mi); %RAE
            bnmtil=c(ni,mi)*ctil(mi)+s(ni,mi)*stil(mi); %RAE
            sumhn=sumhn+p(ni,mp1)*bnmtil; %RAE
            sumgmn=sumgmn+(n+m+1)*p(ni,mi)*bnmtil; %RAE
            bnmtm1=c(ni,mi)*ctil(mm1)+s(ni,mi)*stil(mm1); %RAE
            anmtm1=c(ni,mi)*stil(mm1)-s(ni,mi)*ctil(mm1); %RAE
            sumjn=sumjn+mxpnm*bnmtm1;
            sumkn=sumkn-mxpnm*anmtm1;
        end
        sumj=sumj+reorn*sumjn;
        sumk=sumk+reorn*sumkn;
    end
    sumh = sumh+reorn*sumhn;
    sumgm = sumgm+reorn*sumgmn;
end
lambda=sumgm+ep*sumh;
g(1,1)=-muor2*(lambda*xor-sumj);
g(2,1)=-muor2*(lambda*yor-sumk);
g(3,1)=-muor2*(lambda*zor-sumh);
accel=rnp'*g; %RAE
return
```

C.9 gottliebnorm.m

```
function accel = gottliebnorm(mu, re, xin, c, s, nax, max, rnp)
for n = 2:nax+1 %RAE
    norm1(n) = sqrt((2*n+1)/(2*n-1)); %RAE
    norm2(n) = sqrt((2*n+1)/(2*n-3)); %RAE
    norm11(n) = sqrt((2*n+1)/(2*n))/(2*n-1); %RAE
    normn10(n) = sqrt((n+1)*n/2); %RAE
    for m = 1:n %RAE
```

```
        norm1m(n,m) = sqrt((n-m)*(2*n+1)/((n+m)*(2*n-1))); %RAE
        norm2m(n,m) = sqrt((n-m)*(n-m-1)*(2*n+1)/((n+m)*(n+m-1)*(2*n-3))); %RAE
        normn1(n,m) = sqrt((n+m+1)*(n-m)); %RAE
    end %RAE
end %RAE

x=rnp*xin; %RAE
r = sqrt(x(1)^2+x(2)^2+x(3)^2);
ri=1/r;
xor=x(1)*ri;
yor=x(2)*ri;
zor=x(3)*ri;
ep=zor;
reor=re*ri;
reorn=reor;
muor2=mu*ri*ri;

p(1,1) = 1; %RAE
p(1,2) = 0; %RAE
p(1,3) = 0; %RAE
p(2,2) = sqrt(3); %RAE %norm
p(2,3) = 0; %RAE
p(2,4) = 0; %RAE
for n = 2:nax %RAE
    ni = n+1; %RAE
    p(ni,ni) = norm11(n)*p(n,n)*(2*n-1); %RAE %norm
    p(ni,ni+1) = 0; %RAE
    p(ni,ni+2) = 0; %RAE
end

ctil(1)=1; %RAE
stil(1)=0; %RAE
ctil(2)=xor; %RAE
stil(2)=yor; %RAE
sumh=0;
sumgm=1;
sumj=0;
sumk=0;

p(2,1) = sqrt(3)*ep; %RAE %norm
for n=2:nax
    ni=n+1; %RAE
    reorn=reorn*reor;
    n2m1=n+n-1;
    nm1=n-1;
    np1=n+1;

    p(ni,n) = normn1(n,n-1)*ep*p(ni,ni); %RAE %norm
    p(ni,1) = (n2m1*ep*norm1(n)*p(n,1)-nm1*norm2(n)*p(nm1,1))/n; %RAE %norm
    p(ni,2) = (n2m1*ep*norm1m(n,1)*p(n,2)-n*norm2m(n,1)*p(nm1,2))/(nm1); %RAE %norm

    sumhn=normn10(n)*p(ni,2)*c(ni,1); %norm %RAE
    sumgmn=p(ni,1)*c(ni,1)*np1; %RAE
    if (max>0)
        for m = 2:n-2
            mi = m+1; %RAE
            p(ni,mi) = (n2m1*ep*norm1m(n,m)*p(n,mi)-...
                (nm1+m)*norm2m(n,m)*p(nm1,mi))/(n-m); %RAE %norm
        end
        sumjn=0;
        sumkn=0;
        ctil(ni)=ctil(2)*ctil(ni-1)-stil(2)*stil(ni-1); %RAE
        stil(ni)=stil(2)*ctil(ni-1)+ctil(2)*stil(ni-1); %RAE
        if(n<max)
            lim=n;
        else
            lim=max;
        end
        for m=1:lim
            mi=m+1; %RAE
            mm1=mi-1; %RAE
            mp1=mi+1; %RAE

            mxpnm=m*p(ni,mi); %RAE
            bnmtil=c(ni,mi)*ctil(mi)+s(ni,mi)*stil(mi); %RAE
            sumhn=sumhn+normn1(n,m)*p(ni,mp1)*bnmtil; %RAE %norm
            sumgmn=sumgmn+(n+m+1)*p(ni,mi)*bnmtil; %RAE
            bnmtm1=c(ni,mi)*ctil(mm1)+s(ni,mi)*stil(mm1); %RAE
            anmtm1=c(ni,mi)*stil(mm1)-s(ni,mi)*ctil(mm1); %RAE
            sumjn=sumjn+mxpnm*bnmtm1;
            sumkn=sumkn-mxpnm*anmtm1;
        end
        sumj=sumj+reorn*sumjn;
        sumk=sumk+reorn*sumkn;
    end
    sumh = sumh+reorn*sumhn;
    sumgm = sumgm+reorn*sumgmn;
end
lambda=sumgm+ep*sumh;
g(1,1)=-muor2*(lambda*xor-sumj);
g(2,1)=-muor2*(lambda*yor-sumk);
g(3,1)=-muor2*(lambda*zor-sumh);
accel=rnp'*g; %RAE
return
```

Nomenclature

$\bar{C}_{n,m}$	normalized C coefficient
$\bar{P}_{n,m}$	normalized ALF
$\bar{S}_{n,m}$	normalized S coefficient
$\delta_{0,m}$	Kronecker delta function: returns one (1) if $m = 0$ and zero (0) otherwise
γ	angle between position vectors of a differential mass dm and arbitrary point
λ	normalization parameter (subscript expressions refer to ALF of same subscript expression)
μ	central body gravitational parameter
∇	gradient
$\nabla\cdot$	divergence
∇^2	Laplacian
∇_b	gradient with respect to central-body-fixed coordinates
ϕ	geocentric latitude
ϕ_o	orthogonal spherical z coordinate
θ	longitude
θ_o	orthogonal spherical y coordinate
$\vec{a}$	acceleration
$\vec{F}$	force
$\vec{r}$	position vector
dm	differential mass of central body
E	mechanical energy
M	central body mass
m	order of term
m_d	desired maximum order of model
n	degree of term
n_d	desired maximum degree of model

P_n	Legendre polynomial (assumed argument of $\sin\phi$)
$P_n(\cos\gamma)$	Legendre polynomial with argument $\cos\gamma$ for power series
$P_{n,m}$	ALF (assumed argument of $\sin\phi$)
R	ratio of position vector lengths of a differential mass dm over an arbitrary point
r	magnitude of satellite position vector
r_o	orthogonal spherical x coordinate
T	kinetic energy
U	potential
u	potential per unit mass
V	potential energy
x_b	central-body-fixed x coordinate
y_b	central-body-fixed y coordinate
z_b	central-body-fixed z coordinate
$N_{n,m}$	normalization factor
ALF	Associated Legendre Function

Bibliography

[1] Pines, S., "Uniform Representation of the Gravitational Potential and its Derivatives," *AIAA Journal*, Vol. 11, No. 11, Nov. 1973, pp. 1508–1511.

[2] Lear, W. M., "The Gravitational Acceleration Equations," JSC Internal Note 86-FM-15 (JSC-22080), NASA, April 1986.

[3] Gottlieb, R. G., "Fast Gravity, Gravity Partials, Normalized Gravity, Gravity Gradient Torque and Magnetic Field: Derivation, Code and Data," NASA Contractor Report 188243, NASA, Feb. 1993.

[4] Lundberg, J. B. and Schutz, B. E., "Recursion Formulas of Legendre Functions for Use with Nonsingular Geopotential Models," *Journal of Guidance, Control, and Dynamics*, Vol. 11, Jan.-Feb. 1988, pp. 31–38.

[5] DeMars, K. J., Bishop, R. H., Crain, T. P., and Condon, G. L., "Engineering Analysis of Guidance and Navigation Performance in the Uncertain Lunar Environment to Support Human Exploration," *Thirty-First Annual AAS Guidance and Control Conference*, No. AAS 08-046, American Astronautical Society, Breckenridge, CO, Feb. 2008.

[6] Tapley, B. D., Schutz, B. E., and Born, G. H., *Statistical Orbit Determination*, Elsevier, 2004.

[7] Schey, H. M., *Div, Grad, Curl, and All That: An Informal Text on Vector Calculus*, W. W. Norton, 4th ed., 2005.

[8] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, Microcosm Press/Springer, 3rd ed., 2007.

[9] Jahnke, E. and Emde, F., *Tables of Functions with Formulae and Curves*, Dover, 4th ed., 1945.

[10] Mueller, A. C., "A Fast Recursive Algorithm for Calculating the Forces Due to the Geopotential (Program: GEOPOT)," JSC Internal Note 75-FM-42 (JSC-09731), NASA, June 1975.

[11] Gottlieb, R. G., "A Fast Recursive Singularity Free Algorithm for Calculating the First and Second Derivatives of the Geopotential," JSC Internal Note 89-FM-10 (JSC-23762), NASA, July 1990.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE June 2016	3. REPORT TYPE AND DATES COVERED Technical Publication

4. TITLE AND SUBTITLE Normalization and Implementation of Three Gravitational Acceleration Models	5. FUNDING NUMBERS
6. AUTHOR(S) Randy A. Eckman, Aaron J. Brown, Daniel R. Adamo	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lyndon B. Johnson Space Center Houston, Texas 77058	8. PERFORMING ORGANIZATION REPORT NUMBERS S-1222
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER TP-2016-218604

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited Available from the NASA Center for AeroSpace Information (CASI) 7115 Standard Hanover, MD 21076-1320 Category: 66	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Unlike the uniform density spherical shell approximations of Newton, the consequence of spaceflight in the real universe is that gravitational fields are sensitive to the asphericity of their generating central bodies. The gravitational potential of an aspherical central body is typically resolved using spherical harmonic approximations. However, attempting to directly calculate the spherical harmonic approximations results in at least two singularities that must be removed to generalize the method and solve for any possible orbit, including polar orbits. Samuel Pines, Bill Lear, and Robert Gottlieb developed three unique algorithms to eliminate these singularities.This paper documents the methodical normalization of two of the three known formulations for singularity-free gravitational acceleration (namely, the Lear and Gottlieb algorithms) and formulates a general method for defining normalization parameters used to generate normalized Legendre polynomials and Associated Legendre Functions (ALFs) for any algorithm. A treatment of the conventional formulation of the gravitational potential and acceleration is also provided, in addition to a brief overview of the philosophical differences between the three known singularity-free algorithms.

14. SUBJECT TERMS gravity, acceleration, gravitational acceleration, spherical harmonics, normalization, singularity, Lear, Gottlieb, Pines, Legendre	15. NUMBER OF PAGES 74	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited