NASA Contractor Report 188243

Fast Gravity, Gravity Partials, Normalized Gravity, Gravity Gradient Torque and Magnetic Field: Derivation, Code and Data

Robert G. Gottlieb
McDonnell Douglas Space Systems - Houston Division
Houston, Texas

Prepared for
Lyndon B. Johnson Space Center
under contract NAS9-17885

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

February 1993

CONTENTS

List of Figures

List of Tables

1.0 Summary

This report contains the development of a recursive, non-singular method for computing the first and second partials of the gravitation potential, with respect to position, using both unnormalized and normalized harmonic coefficients. When unnormalized coefficients are used, every attempt was been made to build a "fast" algorithm. When normalized coefficients were used, the algorithm developed uses a more stable, albeit more complex, recursive algorithm for the derived Legendre functions. Even so, the normalized algorithm is still quite efficient. The normalized algorithm should be quite stable and portable for model sizes exceeding 180x180 in degree and order. Efficiency in computation was gained by precomputing everything that was not a function of the state and by using singly dimensioned arrays wherever possible as well as arrays of pointers to arrays.

A complete derivation of the gravity gradient torque resulting from a full (nxm) gravity model is given since it uses the second partial of the potential developed earlier.

A complete derivation of the geomagnetic field vector was included since the computation of the magnetic field is so similar to that of the gravitational field.

Ada code for all of the algorithms is included.

Test cases compare the algorithms to each other and to previously published data.

2.0 Introduction

This report is basically a rewrite of Ref [2], with some useful additions. First of all, by examining the derived Legendre functions that are used to compute the gravitational acceleration it is noted that some of them are not functions of the state and hence may be computed only once. This fact is used to speed up the computation of gravity and its partials.

Secondly, a derivation using normalized gravity coefficients and a superior recursion formula for the derived Legendre functions is presented. As the size of gravity models increases, an algorithm using normalized coefficients becomes more attractive since the unnormalizing process requires the computation of terms on the order of 2n! Even for models of size 50x50 this would be a number so large ($\approx 10^{158}$) that some computers might not be able to compute it.

Algorithms are developed, using both normalized and unnormalized gravity coefficients, that compute the first and second derivatives of the potential function. This yields the gravitational acceleration, and the partial derivative of the gravitational acceleration with respect to the position vector. The partial derivative of the gravitational acceleration is needed in the computation of the state transition matrix for both estimation and optimization. In addition, the partial derivative matrix can be applied to the problem of computing general gravity gradient torque.

Next, a general gravity gradient torque derivation is presented that uses the second partial of the potential developed in the previous section.

Since the geomagnetic field is defined in terms of Legendre functions, a derivation of the geomagnetic field is included which is very similar in form to the gravity derivation.

And finally, Ada code as implemented in the Ada Simulation Development System (ASDS), [9], is given for the various algorithms in addition to test cases that verify the validity of the derivations.

3.0 The Gravitational Potential Function

The gravitational potential function is normally written

$$V = -\frac{\mu}{r} - \sum_{n=2}^{\infty}\sum_{m=0}^{n}\frac{\mu}{r}\left(\frac{\alpha_e}{r}\right)^n P_{n,m}(\varepsilon)\left(C_{n,m}\cos(m\lambda) + S_{n,m}\sin(m\lambda)\right) \tag{3-1}$$

where μ is the gravitational constant, α_e is the equatorial radius, r is the magnitude of the position vector, $X = (x_1, x_2, x_3)$, and

$$P_{n,m} = (1-\varepsilon^2)^{\frac{m}{2}}\left(\frac{\partial^m P_n}{\partial \varepsilon^m}\right) \tag{3-2}$$

are the associated Legendre functions and P_n are the Legendre polynomials. Also, we have the sine of the latitude

$$\varepsilon = x_3/r \tag{3-3}$$

and the longitude is computed from

$$\tan\lambda = \frac{x_2}{x_1} \tag{3-4}$$

For notational convention, we define a potential function U to be

$$U \equiv -V$$

and write eq(2-1) as

$$U = \frac{\mu}{r} + \sum_{n=2}^{\infty}\sum_{m=0}^{n}\frac{\mu}{r}\left(\frac{\alpha_e}{r}\right)^n P_{n,m}(\varepsilon)\left(C_{n,m}\cos(m\lambda) + S_{n,m}\sin(m\lambda)\right) \tag{3-5}$$

Given the equation for ε above, $P_{n,m}$ becomes

$$P_{n,m} = \left(\frac{r^2 - x_3^2}{r^2}\right)^{m/2}\frac{\partial^m P_n}{\partial \varepsilon^m} \equiv \frac{\rho^m}{r^m}P_n^m \tag{3-6}$$

where

$$\rho^2 \equiv x_1^2 + x_2^2$$
$$P_n^m \equiv \frac{\partial^m P_n}{\partial \varepsilon^m} \tag{3-7}$$

The P_n^m are known as derived Legendre functions.

Note that

$$\frac{\partial \rho}{\partial \underline{X}} = \left[\frac{x_1}{\rho} \quad \frac{x_2}{\rho} \quad 0 \right]$$
$$\frac{\partial \lambda}{\partial \underline{X}} = \left[\frac{-x_2}{\rho^2} \quad \frac{x_1}{\rho^2} \quad 0 \right] \tag{3-8}$$

Also define,

$$C_m \equiv \rho^m \cos m\lambda$$
$$S_m \equiv \rho^m \sin m\lambda \tag{3-9}$$
$$B_{n,m} \equiv C_{n,m} C_m + S_{n,m} S_m$$

The $C_{n,m}$ and the $S_{n,m}$ are the unnormalized cosine and sine gravity coefficients that result from the mass distribution of the planet. When these coefficients are published, they are published in normalized form. The relationship between the normalized and unnormalized form is given by

$$C_{n,m} = N(n,m)\,\bar{C}_{n,m}$$
$$S_{n,m} = N(n,m)\,\bar{S}_{n,m} \tag{3-10}$$

where

$$N(n,m) \equiv \left(\frac{(n-m)!\,(2n+1)\,(2-\delta_{om})}{(n+m)!} \right)^{1/2} \tag{3-11}$$

Where δ_{om} is 1.0 if m = 0, and is zero otherwise.

The derivation will proceed using unnormalized notation because the derivation is somewhat simpler. In a later section, a derivation using normalized coefficients will be developed. The normalized form does not require the computation of terms on the order of (n+m)! This may be desirable on some computers where very large or small numbers may cause a problem. For now, the potential can be written

$$U = \frac{\mu}{r} + \sum_{n=2}^{\infty} \sum_{m=0}^{n} \frac{\mu}{r} \left(\frac{\alpha_e}{r} \right)^n \frac{P_n^m}{r^m} B_{n,m} \tag{3-12}$$

This form is especially useful since P_n^m, C_m, and S_m can be calculated recursively and the singularity at the pole (ρ = 0) can be avoided.

The unnormalized derived Legendre functions may be calculated recursively a number of different ways. In [1] and [2] the P_n^m were computed from

$$P_n^m = P_{n-2}^m + (2n-1)\,P_{n-1}^{m-1}, (m \geq 1)$$
$$P_n^0 = P_n = ((2n-1)\,\varepsilon P_{n-1} - (n-1)\,P_{n-2})/n \tag{3-13}$$
$$P_0^0 = 1 \qquad P_0^1 = 0$$
$$P_1^0 = \varepsilon \qquad P_1^1 = 1$$

In [3], seven recursion algorithms were compared numerically for stability. Unfortunately, eq(3-13) was not among those studied. Of the seven algorithms studied, two were clearly superior. The simpler of these is

$$P_n^m = ((2n-1)\varepsilon P_{n-1}^m - (n+m-1)P_{n-2}^m)/(n-m),\ (m<n) \tag{3-14}$$

Note that when m=0, eq(3-14) reduces to eq(3-13) for m=0. Experiments similar to those carried out in [3] were conducted by the author using the normalized error between a single precision computation of the P_n^m and a double precision computation of the P_n^m using both eq(3-13) and eq(3-14). In every case eq(3-14) had lower error. The worst error in all cases occurred for $\varepsilon = 0.2$.

Although eq(3-14) is highly stable, it cannot generate the diagonal elements P_n^n. Realizing that $P_{n-2}^n = 0$, (all P_n^m beyond the diagonal are zero), eq(3-13) can be used to compute

$$P_n^n = (2n-1)P_{n-1}^{n-1} \tag{3-15}$$

Starting with either eq(3-13) or eq(3-14) it is rather easy to show that the inner diagonal terms, P_n^{n-1}, can be computed from

$$P_n^{n-1} = \varepsilon P_n^n \tag{3-16}$$

Also, note that

$$\begin{aligned} C_m &= \rho^m \cos m\lambda = \rho^{m-1+1}\cos(m-1+1)\lambda = C_1 C_{m-1} - S_1 S_{m-1} \\ S_m &= \rho^m \sin m\lambda = \rho^{m-1+1}\sin(m-1+1)\lambda = S_1 C_{m-1} + C_1 S_{m-1} \\ C_1 &= \rho\cos\lambda = \rho\frac{x_1}{\rho} = x_1 \\ S_1 &= \rho\sin\lambda = \rho\frac{x_2}{\rho} = x_2 \end{aligned} \tag{3-17}$$

It is helpful during coding to note that $\bar{C}_m \equiv C_m/r^m$ and $\bar{S}_m \equiv S_m/r^m$ are also recursive, since

$$\begin{aligned} \bar{C}_m &= \frac{C_m}{r^m} = \frac{C_m}{r^{m-1+1}} = \bar{C}_1\bar{C}_{m-1} - \bar{S}_1\bar{S}_{m-1} \\ \bar{S}_m &= \frac{S_m}{r^m} = \frac{S_m}{r^{m-1+1}} = \bar{S}_1\bar{C}_{m-1} + \bar{C}_1\bar{S}_{m-1} \\ \bar{C}_1 &= \frac{x_1}{r} \\ \bar{S}_1 &= \frac{x_2}{r} \end{aligned} \tag{3-18}$$

4.0 The First Partial

The gravitational acceleration vector, g, is calculated as the first partial derivative of U with respect to the planet-fixed position vector, $\underline{X}$. From eq(3-12), we have

$$g = \frac{\partial U}{\partial \underline{X}} = \frac{\partial U}{\partial r}\frac{\partial r}{\partial \underline{X}} + \frac{\partial U}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial \underline{X}} + \frac{\partial U}{\partial B_{n,m}}\frac{\partial B_{n,m}}{\partial \underline{X}} \tag{4-1}$$

where[1]

$$\frac{\partial U}{\partial r} = -\frac{\mu}{r^2} - \sum\sum \frac{\mu}{r^2}\left(\frac{\alpha_e}{r}\right)^n (n+m+1)\frac{P_n^m B_{n,m}}{r^m} \tag{4-2}$$

$$\frac{\partial U}{\partial \varepsilon} = \sum\sum \frac{\mu}{r}\left(\frac{\alpha_e}{r}\right)^n P_n^{m+1}\frac{B_{n,m}}{r^m} \tag{4-3}$$

$$\frac{\partial U}{\partial B_{n,m}} = \sum\sum \frac{\mu}{r}\left(\frac{\alpha_e}{r}\right)^n \frac{P_n^m}{r^m} \tag{4-4}$$

We note that

$$\frac{\partial B_{n,m}}{\partial \underline{X}} = C_{n,m}\left(\frac{\partial C_m}{\partial \rho}\frac{\partial \rho}{\partial \underline{X}} + \frac{\partial C_m}{\partial \lambda}\frac{\partial \lambda}{\partial \underline{X}}\right) + S_{n,m}\left(\frac{\partial S_m}{\partial \rho}\frac{\partial \rho}{\partial \underline{X}} + \frac{\partial S_m}{\partial \lambda}\frac{\partial \lambda}{\partial \underline{X}}\right) \tag{4-5}$$

or

$$\frac{\partial B_{n,m}}{\partial \underline{X}} = mC_{n,m}\left(\rho^{m-1}\cos m\lambda\frac{\partial \rho}{\partial \underline{X}} - \rho^m \sin m\lambda\frac{\partial \lambda}{\partial \underline{X}}\right) + mS_{n,m}\left(\rho^{m-1}\sin m\lambda\frac{\partial \rho}{\partial \underline{X}} + \rho^m \cos m\lambda\frac{\partial \lambda}{\partial \underline{X}}\right)$$

Now, using the definitions of $\frac{\partial \rho}{\partial \underline{X}}$ and $\frac{\partial \lambda}{\partial \underline{X}}$ given earlier,

$$\frac{\partial B_{n,m}}{\partial \underline{X}} = m\rho^{m-1}C_{n,m}\left(\frac{\cos m\lambda}{\rho}\begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} - \frac{\sin m\lambda}{\rho}\begin{bmatrix} -x_2 \\ x_1 \\ 0 \end{bmatrix}\right) + m\rho^{m-1}S_{n,m}\left(\frac{\sin m\lambda}{\rho}\begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} + \frac{\cos m\lambda}{\rho}\begin{bmatrix} -x_2 \\ x_1 \\ 0 \end{bmatrix}\right)$$

where column matrices are used in the interest of saving space. Since $\cos\lambda = \frac{x_1}{\rho}$ and $\sin\lambda = \frac{x_2}{\rho}$, and using the definition of C_m and S_m, it is rather easy to show that

$$\frac{\partial B_{n,m}}{\partial \underline{X}} = \begin{bmatrix} m(C_{n,m}C_{m-1} + S_{n,m}S_{m-1}) \\ -m(C_{n,m}S_{m-1} - S_{n,m}C_{m-1}) \\ 0 \end{bmatrix} \equiv m\begin{bmatrix} B_{n,m-1} \\ -A_{n,m-1} \\ 0 \end{bmatrix} \equiv m\underline{b} \tag{4-6}$$

Also,

1. Note: For notational simplicity, $\sum\sum \equiv \sum_{n=2}^{\infty}\sum_{m=0}^{n}$ throughout.

$$\frac{\partial r}{\partial \underline{X}} = \frac{\underline{X}}{r} \tag{4-7}$$

and

$$\frac{\partial \varepsilon}{\partial \underline{X}} = \frac{1}{r}\underline{\alpha} - \frac{x_3}{r^3}\underline{X} \tag{4-8}$$

where

$$\underline{\alpha}^T = (\,0\;0\;1\,) \tag{4-9}$$

Combining these partials and substituting into eq(4-1), we get

$$\begin{aligned}\frac{\partial U}{\partial \underline{X}} = &-\frac{\mu}{r^2}\left(1+\sum\sum\left(\frac{\alpha_e}{r}\right)^n (n+m+1)\frac{P_n^m B_{n,m}}{r^m}\right)\frac{\underline{X}}{r} \\ &+\frac{\mu}{r^2}\sum\sum\left(\frac{\alpha_e}{r}\right)^n P_n^{m+1}\frac{B_{n,m}}{r^m}\underline{\alpha} - \frac{\mu}{r^2}\sum\sum\left(\frac{\alpha_e}{r}\right)^n P_n^{m+1}\frac{B_{n,m}}{r^m}\frac{x_3}{r}\frac{\underline{X}}{r} \\ &+\frac{\mu}{r^2}\sum\sum\left(\frac{\alpha_e}{r}\right)^n \frac{P_n^m}{r^{m-1}}\begin{bmatrix} m(C_{n,m}C_{m-1}+S_{n,m}S_{m-1}) \\ -m(C_{n,m}S_{m-1}-S_{n,m}C_{m-1}) \\ 0 \end{bmatrix}\end{aligned} \tag{4-10}$$

Defining

$$\begin{aligned} J_n &\equiv \sum_{m=1}^{n} m\frac{P_n^m}{r^{m-1}}(C_{n,m}C_{m-1}+S_{n,m}S_{m-1}) \\ K_n &\equiv -\sum_{m=1}^{n} m\frac{P_n^m}{r^{m-1}}(C_{n,m}S_{m-1}-S_{n,m}C_{m-1}) \\ \Gamma_n &\equiv C_{n,0}(n+1)P_n^0 + \sum_{m=1}^{n}(1+n+m)\frac{P_n^m}{r^m}(C_{n,m}C_m+S_{n,m}S_m) \\ H_n &\equiv C_{n,0}P_n^1 + \sum_{m=1}^{n}\frac{P_n^{m+1}}{r^m}(C_{n,m}C_m+S_{n,m}S_m) \end{aligned} \tag{4-11}$$

$$
\begin{aligned}
H &\equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n H_n \\
\Gamma &\equiv 1 + \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \Gamma_n \\
J &\equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n J_n \\
K &\equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n K_n \\
\Lambda &\equiv \Gamma + \frac{x_3 H}{r} \\
\hat{X} &\equiv \frac{X}{r}
\end{aligned}
\tag{4-12}
$$

we can write

$$
g \equiv \frac{\partial U}{\partial X} = -\frac{\mu}{r^2}\left(\Lambda \hat{X} - \begin{bmatrix} J \\ K \\ H \end{bmatrix}\right) \tag{4-13}
$$

Note that the final result for the first partial derivative of the potential is a rather simple, compact, vector equation.

5.0 The Second Partial

Next, we calculate $\frac{\partial^2 U}{\partial X^2}$ starting with eq(4-1), as it leads directly to a compact symmetric notation.

We have from eq(4-1)

$$\frac{\partial U}{\partial X} = \frac{\partial U \partial r}{\partial r \partial X} + \frac{\partial U \partial \varepsilon}{\partial \varepsilon \partial X} + \frac{\partial U}{\partial B_{n,m} \partial X} \frac{\partial B_{n,m}}{\partial X}$$

Thus,

$$\begin{aligned}
\frac{\partial^2 U}{\partial X^2} &= \frac{\partial^2 U \partial r}{\partial r^2 \partial X \partial X} \frac{\partial r^T}{\partial X} + \frac{\partial^2 U \partial \varepsilon}{\partial \varepsilon^2 \partial X \partial X} \frac{\partial \varepsilon^T}{\partial X} + \frac{\partial^2 U}{\partial B_{n,m}^2 \partial X} \frac{\partial B_{n,m}}{\partial X} \frac{\partial B_{n,m}^T}{\partial X} \\
&+ \frac{\partial^2 U}{\partial \varepsilon \partial r} \left(\frac{\partial r}{\partial X \partial X} \frac{\partial \varepsilon^T}{\partial X} + \frac{\partial \varepsilon}{\partial X \partial X} \frac{\partial r^T}{\partial X} \right) + \frac{\partial^2 U}{\partial r \partial B_{n,m}} \left(\frac{\partial B_{n,m}}{\partial X} \frac{\partial r^T}{\partial X} + \frac{\partial r}{\partial X \partial X} \frac{\partial B_{n,m}^T}{\partial X} \right) \\
&+ \frac{\partial^2 U}{\partial \varepsilon \partial B_{n,m}} \left(\frac{\partial B_{n,m}}{\partial X} \frac{\partial \varepsilon^T}{\partial X} + \frac{\partial \varepsilon}{\partial X \partial X} \frac{\partial B_{n,m}^T}{\partial X} \right) + \frac{\partial U \partial^2 r}{\partial r \partial X^2} + \frac{\partial U \partial^2 \varepsilon}{\partial \varepsilon \partial X^2} + \frac{\partial U}{\partial B_{n,m} \partial X^2} \frac{\partial^2 B_{n,m}}{\partial X^2}
\end{aligned} \tag{5-1}$$

The second and cross partials appearing in eq(5-1) are

$$\begin{aligned}
\frac{\partial^2 U}{\partial r^2} &= \frac{2\mu}{r^3} + \sum\sum \mu \left(\frac{\alpha_e}{r}\right)^n \frac{(m+n+1)}{r^3} \frac{(m+n+2)}{r^m} P_n^m B_{n,m} \\
\frac{\partial^2 U}{\partial \varepsilon^2} &= \sum\sum \frac{\mu}{r} \left(\frac{\alpha_e}{r}\right)^n \frac{1}{r^m} P_n^{m+2} B_{n,m} \\
\frac{\partial^2 U}{\partial B_{n,m}} &= 0 \\
\frac{\partial^2 U}{\partial \varepsilon \partial r} &= -\sum\sum \frac{\mu}{r^2} \left(\frac{\alpha_e}{r}\right)^n \frac{(m+n+1)}{r^m} P_n^{m+1} B_{n,m} \\
\frac{\partial^2 U}{\partial \varepsilon \partial B_{n,m}} &= \sum\sum \frac{\mu}{r} \left(\frac{\alpha_e}{r}\right)^n \frac{P_n^{m+1}}{r^m} \\
\frac{\partial^2 U}{\partial r \partial B_{n,m}} &= -\sum\sum \frac{\mu}{r^2} \left(\frac{\alpha_e}{r}\right)^n \frac{(m+n+1)}{r^m} P_n^m \\
\frac{\partial r}{\partial X} \frac{\partial r^T}{\partial X} &= \frac{XX^T}{r^2}
\end{aligned} \tag{5-2}$$

$$\frac{\partial \varepsilon}{\partial \underline{X}}\frac{\partial \varepsilon^T}{\partial \underline{X}} = \frac{x_3^2}{r^6}\underline{X}\underline{X}^T - \frac{x_3}{r^4}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) + \frac{1}{r^2}\underline{\alpha}\underline{\alpha}^T$$

$$\frac{\partial^2 r}{\partial \underline{X}^2} = \frac{1}{r}\left(I - \frac{\underline{X}\underline{X}^T}{r^2}\right)$$

$$\frac{\partial^2 \varepsilon}{\partial \underline{X}^2} = \frac{3x_3}{r^5}\underline{X}\underline{X}^T - \frac{1}{r^3}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) - \frac{x_3}{r^3}I \tag{5-3}$$

$$\frac{\partial^2 B_{n,m}}{\partial \underline{X}^2} = m(m-1)\begin{bmatrix} B_{n,m-2} & -A_{n,m-2} & 0 \\ -A_{n,m-2} & -B_{n,m-2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where I is a 3x3 identity matrix, and

$$\begin{aligned} B_{n,m-2} &\equiv C_{n,m}C_{m-2} + S_{n,m}S_{m-2} \\ A_{n,m-2} &\equiv C_{n,m}S_{m-2} - S_{n,m}C_{m-2} \end{aligned} \tag{5-4}$$

We also note the special combinations

$$\frac{\partial r}{\partial \underline{X}}\frac{\partial \varepsilon^T}{\partial \underline{X}} + \frac{\partial \varepsilon}{\partial \underline{X}}\frac{\partial r^T}{\partial \underline{X}} = \frac{1}{r^2}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) - 2\frac{x_3}{r^4}\underline{X}\underline{X}^T \tag{5-5}$$

and

$$\begin{aligned} \frac{\partial B_{n,m}}{\partial \underline{X}}\frac{\partial r^T}{\partial \underline{X}} + \frac{\partial r}{\partial \underline{X}}\frac{\partial B_{n,m}^T}{\partial \underline{X}} &= m\begin{bmatrix} B_{n,m-1} \\ -A_{n,m-1} \\ 0 \end{bmatrix}\frac{\underline{X}^T}{r} + m\frac{\underline{X}}{r}\begin{bmatrix} B_{n,m-1} & -A_{n,m-1} & 0 \end{bmatrix} \\ &= \frac{m}{r}(\underline{b}\underline{X}^T + \underline{X}\underline{b}^T) \end{aligned} \tag{5-6}$$

where

$$\underline{b} = \begin{bmatrix} B_{n,m-1} \\ -A_{n,m-1} \\ 0 \end{bmatrix} \tag{5-7}$$

and

$$\frac{\partial B_{n,m}}{\partial \underline{X}}\frac{\partial \varepsilon^T}{\partial \underline{X}} + \frac{\partial \varepsilon}{\partial \underline{X}}\frac{\partial B_{n,m}^T}{\partial \underline{X}} = \frac{m}{r}\left((\underline{b}\underline{\alpha}^T + \underline{\alpha}\underline{b}^T) - \frac{x_3}{r^3}(\underline{b}\underline{X}^T + \underline{X}\underline{b}^T)\right) \tag{5-8}$$

Putting all these into eq(5-1) leads to

$$\frac{\partial^2 U}{\partial \underline{X}^2} = \frac{\mu}{r^3}\{2 + \sum\sum (\frac{\alpha_e}{r})^n (m+n+1) \frac{(m+n+2)}{r^m} P_n^m B_{n,m}\} \frac{\underline{X}\underline{X}^T}{r^2}$$

$$+ \frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n P_n^{m+2} \frac{B_{n,m}}{r^m}\left[\frac{x_3^2}{r^4}\underline{X}\underline{X}^T \frac{x_3}{r^2}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) + \underline{\alpha}\underline{\alpha}^T\right]$$

$$-\frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n \frac{(n+m+1)}{r^m} P_n^{m+1} B_{n,m}\left[\frac{1}{r}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) - 2\frac{x_3}{r^3}\underline{X}\underline{X}^T\right]$$

$$+ \frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n \frac{P_n^{m+1}}{r^{m-1}} m\left[(\underline{b}\underline{\alpha}^T + \underline{\alpha}\underline{b}^T) - \frac{x_3}{r^2}(\underline{b}\underline{X}^T + \underline{X}\underline{b}^T)\right]$$

$$-\frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n \frac{(n+m+1)}{r^{m-1}} P_n^m \frac{m}{r}[(\underline{b}\underline{X}^T + \underline{X}\underline{b}^T)]$$

$$-\frac{\mu}{r^3}\left(1 + \sum\sum (\frac{\alpha_e}{r})^n \frac{(n+m+1)}{r^m} P_n^m B_{n,m}\right)\left[I - \frac{\underline{X}\underline{X}^T}{r^2}\right]$$

$$+ \frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n P_n^{m+1} \frac{B_{n,m}}{r^m}\left[3\frac{x_3}{r^3}\underline{X}\underline{X}^T - \frac{1}{r}(\underline{X}\underline{\alpha}^T + \underline{\alpha}\underline{X}^T) - \frac{x_3}{r}I\right]$$

$$+ \frac{\mu}{r^3}\sum\sum (\frac{\alpha_e}{r})^n \frac{P_n^m}{r^{m-2}} m(m-1) \begin{bmatrix} B_{n,m-2} & -A_{n,m-2} & 0 \\ -A_{n,m-2} & -B_{n,m-2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (5\text{-}9)$$

If we now define

$$L \equiv 2 + \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=0}^{n} (n+m+1)(n+m+2) P_n^m \frac{B_{n,m}}{r^m}$$

$$M \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=0}^{n} \frac{P_n^{m+2}}{r^m} B_{n,m}$$

$$N \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=2}^{n} P_n^m m(m-1) \frac{(C_{n,m}C_{m-2} + S_{n,m}S_{m-2})}{r^{m-2}}$$

$$\Omega \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=2}^{n} P_n^m m(m-1) \frac{(C_{n,m}S_{m-2} - S_{n,m}C_{m-2})}{r^{m-2}}$$

$$P \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=0}^{n} P_n^{m+1} (m+n+1) \frac{B_{n,m}}{r^m} \qquad (5\text{-}10)$$

$$Q \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=1}^{n} P_n^{m+1} m \frac{(C_{n,m}C_{m-1} + S_{n,m}S_{m-1})}{r^{m-1}}$$

$$R \equiv -\sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=1}^{n} P_n^{m+1} m \frac{(C_{n,m}S_{m-1} - S_{n,m}C_{m-1})}{r^{m-1}}$$

$$S \equiv \sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=1}^{n} (m+n+1) P_n^m m \frac{(C_{n,m}S_{m-1} - S_{n,m}C_{m-1})}{r^{m-1}}$$

$$T \equiv -\sum_{n=2}^{\infty} \left(\frac{\alpha_e}{r}\right)^n \sum_{m=1}^{n} (m+n+1) P_n^m m \frac{(C_{n,m}S_{m-1} - S_{n,m}C_{m-1})}{r^{m-1}}$$

then with these definitions

$$\underline{\alpha} \equiv \begin{bmatrix} Q \\ R \\ 0 \end{bmatrix}$$

$$Y \equiv \begin{bmatrix} S \\ T \\ 0 \end{bmatrix} \qquad (5\text{-}11)$$

we can write eq(5-1) as

$$\frac{\partial^2 U}{\partial \underline{X}^2} = \frac{\mu}{r^3} L \frac{\underline{X}\underline{X}^T}{r^2} + \frac{\mu}{r^3} M \left[\frac{x_3^2}{r^4} \underline{X}\underline{X}^T - \frac{x_3}{r^2} (\underline{X}\underline{\Omega}^T + \underline{\Omega}\underline{X}^T) + \underline{\Omega}\underline{\Omega}^T \right]$$
$$- \frac{\mu}{r^3} P \left[\frac{1}{r} (\underline{X}\underline{\Omega}^T + \underline{\Omega}\underline{X}^T) - 2 \frac{x_3}{r^3} \underline{X}\underline{X}^T \right] + \frac{\mu}{r^3} \left[(\underline{\Omega}\underline{\Omega}^T + \underline{\Omega}\underline{\Omega}^T) - \frac{x_3}{r^3} (\underline{\Omega}\underline{X}^T + \underline{X}\underline{\Omega}^T) \right]$$
$$- \frac{\mu}{r^3} \left[\frac{\underline{Y}\underline{X}^T + \underline{X}\underline{Y}^T}{r} \right] - \frac{\mu}{r^3} \Gamma \left[I - \frac{\underline{X}\underline{X}^T}{r^2} \right] + \frac{\mu}{r^3} H \left[\frac{3x_3}{r^3} \underline{X}\underline{X}^T - \frac{1}{r} (\underline{X}\underline{\Omega}^T + \underline{\Omega}\underline{X}^T) - \frac{x_3}{r} I \right] \qquad (5\text{-}12)$$
$$+ \frac{\mu}{r^3} \begin{bmatrix} N & -\Omega & 0 \\ -\Omega & -N & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Collecting like terms, we get

$$\frac{\partial^2 U}{\partial \underline{X}^2} = \frac{\mu}{r^3} \left(L + M \frac{x_3^2}{r^2} + 2P \frac{x_3}{r} + \Gamma + 3H \frac{x_3}{r} \right) \left(\frac{\underline{X}\underline{X}^T}{r^2} \right)$$
$$- \frac{\mu}{r^3} (M \frac{x_3}{r} + P + H) \left[\frac{\underline{X}\underline{\Omega}^T + \underline{\Omega}\underline{X}^T}{r} \right]$$
$$+ \frac{\mu}{r^3} M \underline{\Omega}\underline{\Omega}^T - \frac{\mu}{r^3} (\Gamma + \frac{x_3}{r} H) I$$

$$+ \frac{\mu}{r^3} \left\{ (\underline{\Omega}\underline{\Omega}^T + \underline{\Omega}\underline{\Omega}^T) - \frac{x_3}{r^2} (\underline{\Omega}\underline{X}^T + \underline{X}\underline{\Omega}^T) - (\frac{\underline{Y}\underline{X}^T + \underline{X}\underline{Y}^T}{r}) + \begin{bmatrix} N & -\Omega & 0 \\ -\Omega & -N & 0 \\ 0 & 0 & 0 \end{bmatrix} \right\} \qquad (5\text{-}13)$$

Recalling that

$$\Lambda = (\Gamma + \varepsilon H)$$
$$\hat{\underline{X}} = \frac{\underline{X}}{r}$$

and defining

$$F \equiv L + \varepsilon (M\varepsilon + 2 (P + H)) + \Lambda$$
$$G \equiv -(M\varepsilon + P + H) \qquad (5\text{-}14)$$
$$\underline{d} \equiv \varepsilon \underline{\Omega} + \underline{\Upsilon}$$

we have finally

$$\frac{\partial^2 U}{\partial \underline{X}^2} = \frac{\mu}{r^3} \left\{ [\hat{\underline{X}} | \underline{\Omega}] \begin{bmatrix} F & G \\ G & M \end{bmatrix} \begin{bmatrix} \hat{\underline{X}}^T \\ \underline{\Omega}^T \end{bmatrix} \right.$$
$$\left. + [\hat{\underline{X}} | \underline{d}] \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \hat{\underline{X}}^T \\ \underline{d}^T \end{bmatrix} + \begin{bmatrix} N - \Lambda & -\Omega & Q \\ -\Omega & -(N + \Lambda) & R \\ Q & R & -\Lambda \end{bmatrix} \right\} \qquad (5\text{-}15)$$

Note that the final result for the second partial derivative of the potential is a rather simple, compact, symmetric matrix equation.

6.0 Computational Considerations

If the derived Legendre functions are arranged in a table such as Table 1 below, it is rather easy to see that to the right of the diagonal all terms are zero. Along the diagonal only pure numbers appear, and immediately to the left of the diagonal, the diagonal term appears multiplied by ε. This means that a number of the derived Legendre functions that are needed in the computation of gravity can be computed once only and stored, and do not need to be computed using the recursion relationships.

This can save a great deal of time. If in addition, all coefficients in the recursion relationships are computed and stored as functions of n, another slight savings can be gained. It was found that these two taken together save about 15 - 20% of the time normally taken, depending on the size of the coefficient array used.

In Ref [1] and [2], the gravity coefficients were placed in a single array in an attempt to avoid the time it takes to manage a two dimensional array on a computer. In the new code given in Appendix A, it was found that by having an array of pointers to arrays the code is more clear and just as fast.

	m=0	1	2	3	4	5	6	7
n = 0	$P_0^0 = 1$	$P_0^1 = 0$	$P_0^2 = 0$	etc.				
1	$P_1^0 = \varepsilon$	$P_1^1 = 1$	$P_1^2 = 0$	$P_1^3 = 0$	etc.			
2	P_2^0	$P_2^1 = 3\varepsilon$	$P_2^2 = 3$	$P_2^3 = 0$	$P_2^4 = 0$	etc.		
3	P_3^0	P_3^1	$P_3^2 = 15\varepsilon$	$P_3^3 = 15$	$P_3^4 = 0$	$P_3^5 = 0$	etc.	
4	P_4^0	P_4^1	P_4^2	$P_4^3 = 105\varepsilon$	$P_4^4 = 105$	$P_4^5 = 0$	$P_4^6 = 0$	etc.
5	P_5^0	P_5^1	P_5^2	P_5^3	$P_5^4 = 945\varepsilon$	$P_5^5 = 945$	$P_5^6 = 0$	$P_5^7 = 0$

Table 1. Table of Derived Legendre Functions

In the code given in Appendix A, the "fast" gravity model takes advantage of all of the things mentioned in this section. The normalized model is also "fast" in the same sense.

Since no difference in the computation of the gravitational acceleration could be detected out through a 30x30 model using both eq(3-13) or eq(3-14) to generate the P_n^m, and since eq(3-14) requires more multiplications and additions than eq(3-13), eq(3-13) was used for the "fast" model. Since normalization makes more sense as model size, n, increases, it was decided to use eq(3-14) in constructing the normalized algorithm. (Section 7).

7.0 First and Second Partials Using Normalized Coefficients

Following [3] and [4], define the normalized derived Legendre functions,

$$\bar{P}_n^m \equiv N(n,m)\,P_n^m \tag{7-1}$$

where N(n,m) is given by eq(3-11). Note that this definition means that

$$\begin{aligned} \bar{P}_n^m \bar{C}_{n,m} &= P_n^m C_{n,m} \\ \bar{P}_n^m \bar{S}_{n,m} &= P_n^m S_{n,m} \end{aligned} \tag{7-2}$$

Note also that if m = 0,

$$N(n,0) = (2n+1)^{\frac{1}{2}} \tag{7-3}$$

and if $m \geq 1$

$$N(n,m) = \left(\frac{(n-m)!\,(2n+1)\,2}{(n+m)!}\right)^{\frac{1}{2}} \tag{7-4}$$

The recursion relationships given earlier for the derived Legendre functions are reproduced below for the sake of convenience

$$\begin{gathered} P_n^m = P_{n-2}^m + (2n-1)\,P_{n-1}^{m-1}, (m \geq 1), \text{ or} \\ P_n^m = \left((2n-1)\,\varepsilon P_{n-1}^m - (n+m-1)\,P_{n-2}^m\right)/(n-m), \; (m<n) \\ P_n^0 = P_n = \left((2n-1)\,\varepsilon P_{n-1} - (n-1)\,P_{n-2}\right)/n \\ P_n^n = (2n-1)\,P_{n-1}^{n-1} \\ P_n^{n-1} = \varepsilon P_n^n \\ P_0^0 = 1 \qquad P_0^1 = 0 \\ P_1^0 = \varepsilon \qquad P_1^1 = 1 \end{gathered} \tag{7-5}$$

Either of the formulae for computing the P_n^m may be used. The first is faster, and the second is more stable numerically. The difference in the resulting algorithm is slight, as shall be seen. Taking the definition given in eq(7-1) and applying it to the first of eq(7-5) gives

$$\bar{P}_n^m = N(n,m)\,P_n^m = N(n,m)\frac{N(n-2,m)}{N(n-2,m)}P_{n-2}^m + (2n-1)\,N(n,m)\frac{N(n-1,m-1)}{N(n-1,m-1)}P_{n-1}^{m-1} \tag{7-6}$$

or,

$$\bar{P}_n^m = \frac{N(n,m)}{N(n-2,m)}\bar{P}_{n-2}^m + (2n-1)\frac{N(n,m)}{N(n-1,m-1)}\bar{P}_{n-1}^{m-1} \tag{7-7}$$

Similarly,

$$\bar{P}_n^m = N(n,m)\,P_n^m = \varepsilon\frac{(2n-1)}{(n-m)}\frac{N(n,m)}{N(n-1,m)}\bar{P}_{n-1}^m - \frac{(n+m-1)}{(n-m)}\frac{N(n,m)}{N(n-2,m)}\bar{P}_{n-2}^m\Big|_{m<n}$$

$$\bar{P}_n^0 = N(n,0)\,P_n^0 = \varepsilon\frac{(2n-1)}{n}\frac{N(n,0)}{N(n-1,0)}\bar{P}_{n-1}^0 - \frac{(n-1)}{n}\frac{N(n,0)}{N(n-2,0)}\bar{P}_{n-2}^0$$

$$\bar{P}_n^n = N(n,n)\,P_n^n = (2n-1)\frac{N(n,n)}{N(n-1,n-1)}\bar{P}_{n-1}^{n-1} = \left(\frac{2n+1}{2n}\right)^{\frac{1}{2}}\bar{P}_{n-1}^{n-1}\Big|_{n\geq 2} \tag{7-8}$$

$$\bar{P}_n^{n-1} = N(n,n-1)\,P_n^{n-1} = \varepsilon\frac{N(n,n-1)}{N(n,n)}\bar{P}_n^n = \varepsilon(2n-1)\frac{N(n,n-1)}{N(n-1,n-1)}\bar{P}_{n-1}^{n-1}$$

$$\bar{P}_0^0 = 1 \qquad \bar{P}_0^1 = 0$$

$$\bar{P}_1^0 = \sqrt{3}\varepsilon \qquad \bar{P}_1^1 = \sqrt{3}$$

These define the recursion relationships for the $\bar{P}_n^m$, at least symbolically. Computationally, these relationships can be simplified. Defining

$$\bar{\xi}(n,m) \equiv \frac{N(n,m)}{N(n-2,m)}\Big|_{m\geq 1} = \left(\frac{(n-m)(n-m-1)(2n+1)}{(n+m)(n+m-1)(2n-3)}\right)^{\frac{1}{2}}$$

$$\bar{\eta}(n,m) \equiv \frac{(2n-1)N(n,m)}{N(n-1,m-1)}\Big|_{m\geq 1} = \left(\frac{2(2n+1)(2n-1)}{(n+m)(n+m-1)(2-\delta_{0,m-1})}\right)^{\frac{1}{2}} \tag{7-9}$$

and

$$\xi(n,m) \equiv \frac{(2n-1)}{(n-m)}\frac{N(n,m)}{N(n-1,m)}\Big|_{m<n} = \left(\frac{(2n-1)(2n+1)}{(n-m)(n+m)}\right)^{\frac{1}{2}}$$

$$\eta(n,m) \equiv \frac{(n+m-1)}{(n-m)}\frac{N(n,m)}{N(n-2,m)}\Big|_{m<n} = \left(\frac{(n+m-1)(2n+1)(n-m-1)}{(n+m)(n-m)(2n-3)}\right)^{\frac{1}{2}} \tag{7-10}$$

the recursion relationship for $\bar{P}_n^m$ becomes either

$$\bar{P}_n^m = \bar{\xi}(n,m)\,\bar{P}_{n-2}^m + \bar{\eta}(n,m)\,\bar{P}_{n-1}^{m-1} \tag{7-11}$$

or

$$\bar{P}_n^m = \xi(n,m)\,\varepsilon\bar{P}_{n-1}^m - \eta(n,m)\,\bar{P}_{n-2}^m \tag{7-12}$$

Note that $\bar{\xi}(n,m)$ and $\bar{\eta}(n,m)$ or $\xi(n,m)$ and $\eta(n,m)$ are constant functions of n and m and need be computed only once. There is no need to use both eq(7-11) and eq(7-12), either one will do. The code for the normalized model contained in Appendix A is based on eq(7-12). Another model was developed based on eq(7-11), but that code is not included in Appendix A. The test cases in Appendix B labeled "norm_I" came from the model using eq(7-11), and test cases labeled "norm_II" came from the model using eq(7-12). The only difference in the code is the use of $\bar{\xi}(n,m)$ and $\bar{\eta}(n,m)$ in lieu of $\xi(n,m)$ and $\eta(n,m)$, and eq(7-11) in lieu of eq(7-12). In either case, everything that follows will be exactly the same. Since roughly $\frac{nm}{2}$ more multiplications are required if eq(7-12) is used, it was anticipated that the "norm_I" model would run faster than the "norm_II" model. The difference in run time turned out to be less than the noise in the timer. This

means that "norm_II" is the way to go, since it is based on a more stable recursion formula.

Going back to eq(7-8) and defining

$$\alpha(n) \equiv \frac{(2n-1)N(n,0)}{nN(n-1,0)} = \frac{\sqrt{(2n+1)(2n-1)}}{n}$$
$$\beta(n) \equiv \frac{(n-1)N(n,0)}{nN(n-2,0)} = \frac{(n-1)}{n}\left(\frac{2n+1}{2n-3}\right)^{\frac{1}{2}} \tag{7-13}$$

the equations for the normalized derived Legendre polynomials become

$$\bar{P}_n^0 = \varepsilon\alpha(n)\bar{P}_{n-1}^0 - \beta(n)\bar{P}_{n-2}^0 \tag{7-14}$$

Note that $\alpha(n)$ and $\beta(n)$ need be computed only once and stored since they are only functions of n and are not functions of the state.

Going back to eq(7-8) and defining the inner diagonal term, $\delta(n)$, as

$$\delta(n) \equiv (2n-1)\frac{N(n,n-1)}{N(n-1,n-1)}\bar{P}_{n-1}^{n-1} = (2n+1)^{\frac{1}{2}}\bar{P}_{n-1}^{n-1} \tag{7-15}$$

The inner diagonal, P_n^{n-1} can be computed as

$$P_n^{n-1} = \varepsilon\delta(n) \tag{7-16}$$

Note that $\delta(n)$ need be computed only once and stored since it is only a function of n and is a not function of the state. The use of $\delta(n)$ will speed up the computation since only a single multiply is required to build the inner diagonal term.

Looking now at eqs(3-29) given earlier for J_n, K_n, Γ_n, and H_n, note that P_n^m multiplies $C_{n,m}$ and $S_{n,m}$ everywhere except in H_n. Therefore, P_n^m, $C_{n,m}$ and $S_{n,m}$ may be replaced by $\bar{P}_n^m$, $\bar{C}_{n,m}$ and $\bar{S}_{n,m}$ everywhere except in H_n where the terms $P_n^{m+1}C_{n,m}$ and $P_n^{m+1}S_{n,m}$ appear. Multiplying and dividing by N(n,m) and N(n,m+1) leads to

$$P_n^{m+1}C_{n,m} = \frac{N(n,m)}{N(n,m)}\frac{N(n,m+1)}{N(n,m+1)}P_n^{m+1}C_{n,m} = \frac{N(n,m)}{N(n,m+1)}\bar{P}_n^{m+1}\bar{C}_{n,m} \tag{7-17}$$

Defining

$$\zeta(n,m) \equiv \frac{N(n,m)}{N(n,m+1)} = \left(\frac{(n-m)(2-\delta_{0,m})(n+m+1)}{(2-\delta_{0,m+1})}\right)^{\frac{1}{2}} \tag{7-18}$$

note that

$$\zeta(n,0) = \left(\frac{n(n+1)}{2}\right)^{\frac{1}{2}}$$
$$\zeta(n,m)\,)_{m\geq 1} = ((n-m)(n+m+1))^{\frac{1}{2}} \tag{7-19}$$

These last two may also be computed and stored and then used to compute H_n. This will allow the computation of gravitational acceleration without the necessity of unnormalizing the gravity coefficients. This makes the algorithm more portable, since for large gravity models, some computers can't handle the large numbers involved in the unnormalizing process.

An examination of L, M, N, Ω, P, Q, R, S, and T will show that the only other term needed in order to compute the second partials using normalized coefficients is in the term, M, and involves $P_n^{m+2}C_{n,m}$ and $P_n^{m+2}S_{n,m}$. Consequently, form

$$P_n^{m+2}C_{n,m} = \frac{N(n,m)}{N(n,m)}\frac{N(n,m+2)}{N(n,m+2)}P_n^{m+2}C_{n,m} = \frac{N(n,m)}{N(n,m+2)}\bar{P}_n^{m+2}\bar{C}_{n,m} \tag{7-20}$$

As before, define

$$\Upsilon(n,m) \equiv \frac{N(n,m)}{N(n,m+2)} = \left(\frac{(n-m)\,(n-m-1)\,(2-\delta_{0,m})\,(n+m+1)\,(n+m+2)}{(2-\delta_{0,m+2})}\right)^{\frac{1}{2}} \tag{7-21}$$

Note that

$$\Upsilon(n,0) = \left(\frac{n(n-1)\,(n+1)\,(n+2)}{2}\right)^{\frac{1}{2}}$$

$$\Upsilon(n,m)\,)_{m\geq 1} = \left((n-m)\,(n-m-1)\,(n+m+1)\,(n+m+2)\right)^{\frac{1}{2}} \tag{7-22}$$

Again, these may be computed and stored and used to compute M. This will allow the complete computation of the first and second partials of the potential using normalized coefficients.

One comment, the simplification in Table 1 that allowed the inner diagonal to be the main diagonal multiplied by ε is no longer valid. In the normalized case, the inner diagonal term is computed by multiplying $\delta(n)$ by ε.

All the normalized equations have been coded in ASDS [9] and verified against previously published [2] test cases, and found to agree exactly. The code for the normalized gravity model using the recursion relation from [3] is given in Appendix A, and the test case data is given in Appendix B.

8.0 Gravity Gradient Torque

The gravity gradient torque on a spacecraft is derived twice here. The first derivation uses a point mass gravity model and assumes that $1/|\bar{r}+\delta\bar{r}|^3$ is approximated by the first term of the binomial expansion.

The second derivation uses a full n x m gravity model and assumes that gravity varies linearly about the center of mass, i.e., that gravity in the vicinity of the center of mass is given by $\bar{g} = \bar{g}_{cg} + \frac{\partial \bar{g}}{\partial \bar{r}}\delta\bar{r}$, where both $\bar{g}_{cg}$ and $\frac{\partial \bar{g}}{\partial \bar{r}}$ are computed using a general gravity model subroutine such as the models discussed in previous sections, and given in Appendix A. This derivation shows that the eigenvectors in [7] are not needed, as was pointed out in [8].

It is then shown that the point mass derivation and the general derivation give identical results when the harmonic coefficients of the full gravity model are set to zero.

It is shown in Appendix B that when only J2 ($-C_{20}$) is used, the general formulation gives the same torque as that given by Roithmayr's [5] model.

It is anticipated that the use of the full potential model in the calculation of the gravity gradient torques will lead to more accurate attitude simulations.

8.1 Point Mass Gravity Model

In Figure 1, the vector $\bar{\rho}$ describes the position of a particle of mass, *dm*, in the body axis system.

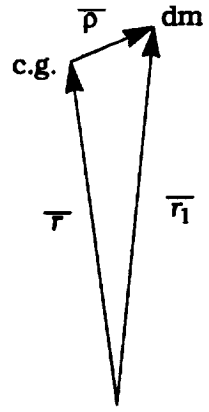


Figure 1 Position of a Particle of Mass in the Body Axis System

Assume the matrix, B, relates the body axis system to the system in which $\bar{r}$ and $\bar{r}_1$ are defined and $\bar{g}$ is computed. Then,

$$\bar{r}_1 = \bar{r} + B\bar{\rho} \tag{8-1}$$

The gravitational force on *dm* is

$$d\bar{F} = -\mu \frac{\bar{r}_1}{r_1^3} dm \tag{8-2}$$

Rotating this force back into the body system yields

$$d\bar{f} = B^T d\bar{F} = -\mu B^T \frac{\bar{r}_1}{r_1^3} dm = -\frac{\mu (B^T \bar{r} + \bar{\rho})}{r_1^3} dm \tag{8-3}$$

We first examine $r_1{}^3$

$$\begin{aligned} r_1^3 &= [\bar{r}_1^T \bar{r}_1]^{3/2} \\ &= [(\bar{r}^T + \bar{\rho}^T B^T)(\bar{r} + B\bar{\rho})]^{3/2} \\ &= [r^2 + \rho^2 + 2\bar{\rho}^T B^T \bar{r}]^{3/2} \end{aligned} \tag{8-4}$$

We note that the term $B^T \bar{r}$ appears in both the numerator and denominator of the expression for $d\bar{f}$ and rewrite it as

$$B^T \bar{r} = r B^T \hat{r} \tag{8-5}$$

where $\hat{r}$ is the unit vector along $\bar{r}$.

Now define the unit vector $\hat{b}$ in the body axis system

$$\hat{b} \equiv B^T \hat{r} \tag{8-6}$$

and write

$$d\bar{f} = -\frac{\mu (r\hat{b} + \bar{\rho})\, dm}{(r^2 + \rho^2 + 2r\bar{\rho}^T \hat{b})^{3/2}} \tag{8-7}$$

The moment about the center of mass due to $d\bar{f}$ is

$$d\bar{\tau} = \bar{\rho} \times d\bar{f} = -\frac{\mu r \bar{\rho} \times \hat{b} dm}{(r^2 + \rho^2 + 2r\bar{\rho}^T \hat{b})^{3/2}} \tag{8-8}$$

If we factor out r^2 and ignore ρ^2 compared to one, we get

$$d\bar{\tau} = -\frac{\mu r \bar{\rho} \times \hat{b} dm}{r^3 \left(1 + 2\frac{\bar{\rho}^T \hat{b}}{r}\right)^{3/2}} \tag{8-9}$$

Now, using the binomial theorem and again ignoring ρ^2 terms, we get

$$d\bar{\tau} = -\frac{\mu}{r^2} \bar{\rho} \times \hat{b} \left(1 - 3\frac{\bar{\rho}^T \hat{b}}{r}\right) dm \tag{8-10}$$

Integrating over mass, we get

$$\bar{\tau} = \int d\bar{\tau} \quad = -\frac{\mu}{r^2} \int \bar{\rho} dm \times \hat{b} + 3\frac{\mu}{r^3} \int (\bar{\rho} \times \hat{b})(\bar{\rho}^T \hat{b})\, dm \tag{8-11}$$

but $\int \bar{\rho} dm = 0$ since $\bar{\rho}$ is measured from the center of mass; hence

$$\bar{\tau} = \frac{3\mu}{r^3} \int (\bar{\rho} \times \hat{b})\, (\bar{\rho}^T \hat{b})\, dm \tag{8-12}$$

where

$$\bar{\rho}^T \hat{b} = \rho_x b_1 + \rho_y b_2 + \rho_z b_3 \tag{8-13}$$

and

$$\bar{\rho} \times \hat{b} = \begin{vmatrix} i & j & k \\ \rho_x & \rho_y & \rho_z \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{bmatrix} \rho_y b_3 - \rho_z b_2 \\ \rho_z b_1 - \rho_x b_3 \\ \rho_x b_2 - \rho_y b_1 \end{bmatrix} \tag{8-14}$$

therefore,

$$(\bar{\rho} \times \hat{b})\, (\bar{\rho}^T \hat{b}) = \begin{bmatrix} \rho_x \rho_y b_1 b_3 & \rho_y^2 b_2 b_3 & \rho_y \rho_z b_3^2 & -\rho_x \rho_z b_1 b_2 & -\rho_y \rho_z b_2^2 & -\rho_z^2 b_2 b_3 \\ \rho_x \rho_z b_1^2 & \rho_y \rho_z b_1 b_2 & \rho_z^2 b_1 b_3 & -\rho_x^2 b_1 b_3 & -\rho_x \rho_y b_2 b_3 & -\rho_x \rho_z b_3^2 \\ \rho_x^2 b_1 b_2 & \rho_x \rho_y b_2^2 & \rho_x \rho_z b_2 b_3 & -\rho_x \rho_y b_1^2 & -\rho_y^2 b_1 b_2 & -\rho_y \rho_z b_1 b_3 \end{bmatrix} \tag{8-15}$$

thus

$$\bar{\tau} = \frac{3\mu}{r^3} \begin{bmatrix} b_2 b_3 (I_{zz} - I_{yy}) & + b_1 b_3 I_{xy} & - b_1 b_2 I_{xz} & + I_{yz} (b_3^2 - b_2^2) \\ b_1 b_3 (I_{xx} - I_{zz}) & - b_2 b_3 I_{xy} & + b_1 b_2 I_{yz} & + I_{xz} (b_1^2 - b_3^2) \\ b_1 b_2 (I_{yy} - I_{xx}) & + b_2 b_3 I_{xz} & - b_1 b_3 I_{yz} & + I_{xy} (b_2^2 - b_1^2) \end{bmatrix} \tag{8-16}$$

where we define the moments and products of inertia to be

$$\begin{aligned} I_{xx} &= \int (\rho_y^2 + \rho_z^2)\, dm \\ I_{yy} &= \int (\rho_x^2 + \rho_z^2)\, dm \\ I_{zz} &= \int (\rho_x^2 + \rho_y^2)\, dm \\ I_{xy} &= \int (\rho_x \rho_y)\, dm \\ I_{xz} &= \int (\rho_x \rho_z)\, dm \\ I_{yz} &= \int (\rho_y \rho_z)\, dm \end{aligned} \tag{8-17}$$

8.2 General Gravity Model

The force of gravity at the particle, *dm*, is now assumed to be computed from

$$\bar{g} = \bar{g}_{cg} + \frac{\partial \bar{g}}{\partial \bar{r}} \delta \bar{r} \tag{8-18}$$

Referring back to Figure 1, we recall that B rotates $\bar{\rho}$ back into the system where $\bar{g}$ is computed, i.e., that

$$\delta\bar{r} = B\bar{\rho} \tag{8-19}$$

and therefore the force on the particle is given by

$$d\bar{R} = (\bar{g}_{cg} + \frac{\partial\bar{g}}{\partial\bar{r}} B\bar{\rho})\, dm \tag{8-20}$$

Rotating this force back into the body system yields

$$d\bar{f} = B^T d\bar{F} = (B^T\bar{g}_{cg} + B^T\frac{\partial\bar{g}}{\partial\bar{r}} B\bar{\rho})\, dm \tag{8-21}$$

The moment about the center of mass due to $d\bar{f}$ is

$$d\bar{\tau} = \bar{\rho}\times d\bar{f} = (\bar{\rho}\times B^T\bar{g}_{cg} + \bar{\rho}\times B^T\frac{\partial\bar{g}}{\partial\bar{r}} B\bar{\rho})\, dm \tag{8-22}$$

Integrating over mass, we get

$$\bar{\tau} = \int d\bar{\tau} = \int\bar{\rho}dm\times B^T\bar{g}_{cg} + \int\bar{\rho}\times B^T\frac{\partial\bar{g}}{\partial\bar{r}} B\bar{\rho}\, dm \tag{8-23}$$

The integral $\int\bar{\rho}dm = 0$ since $\bar{\rho}$ is measured from the center of mass. Hence,

$$\bar{\tau} = \int\bar{\rho}\times B^T\frac{\partial\bar{g}}{\partial\bar{r}} B\bar{\rho}\, dm \tag{8-24}$$

Next, define

$$G \equiv B^T\frac{\partial\bar{g}}{\partial\bar{r}} B^{\dagger} \tag{8-25}$$

then

$$\bar{\tau} = \int\bar{\rho}\times G\bar{\rho}\, dm \tag{8-26}$$

† It must be pointed out that $\frac{\partial\bar{g}}{\partial\bar{r}}$ is computed in an equatorial or planet-fixed system and must be rotated into the body system. The simplest choice for B is body-to-equatorial if only spherical or zonals are considered, and body-to-earth-fixed if tesserals are considered.

The operation, $[\bar{\rho}\times\ ]$, may be considered a matrix, where

$$[\bar{\rho}\times\ ] \equiv \begin{bmatrix} 0 & -\rho_z & +\rho_y \\ +\rho_z & 0 & -\rho_x \\ -\rho_y & +\rho_x & 0 \end{bmatrix} \tag{8-27}$$

then

$$\bar{\rho}\times G = \begin{bmatrix} 0 & -\rho_z & +\rho_y \\ +\rho_z & 0 & -\rho_x \\ -\rho_y & +\rho_x & 0 \end{bmatrix}\begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} g_{31}\rho_y - g_{21}\rho_z & g_{32}\rho_y - g_{22}\rho_z & g_{33}\rho_y - g_{23}\rho_z \\ g_{11}\rho_z - g_{31}\rho_x & g_{12}\rho_z - g_{32}\rho_x & g_{13}\rho_z - g_{33}\rho_x \\ g_{21}\rho_x - g_{11}\rho_y & g_{22}\rho_x - g_{12}\rho_y & g_{23}\rho_x - g_{13}\rho_y \end{bmatrix} \tag{8-28}$$

and finally

$$\bar{\rho}\times G\bar{\rho} = \begin{bmatrix} g_{31}\rho_y\rho_x - g_{21}\rho_z\rho_x + g_{32}\rho_y^2 - g_{22}\rho_z\rho_y + g_{33}\rho_y\rho_z - g_{23}\rho_z^2 \\ g_{11}\rho_z\rho_x - g_{31}\rho_x^2 + g_{12}\rho_z\rho_y - g_{32}\rho_x\rho_y + g_{13}\rho_z^2 - g_{33}\rho_x\rho_z \\ g_{21}\rho_x^2 - g_{11}\rho_y\rho_x + g_{22}\rho_x\rho_y - g_{12}\rho_y^2 + g_{23}\rho_x\rho_z - g_{13}\rho_y\rho_z \end{bmatrix} \tag{8-29}$$

Since the G matrix is always *symmetric*[2], we may write

$$\bar{\tau} = \int \bar{\rho}\times G\bar{\rho}\,dm = \begin{bmatrix} g_{23}(I_{zz} - I_{yy}) + g_{13}I_{xy} - g_{12}I_{xz} + I_{yz}(g_{33} - g_{22}) \\ g_{13}(I_{xx} - I_{zz}) - g_{23}I_{xy} + g_{12}I_{yz} + I_{xz}(g_{11} - g_{33}) \\ g_{12}(I_{yy} - I_{xx}) + g_{23}I_{xz} - g_{13}I_{yz} + I_{xy}(g_{22} - g_{11}) \end{bmatrix} \tag{8-30}$$

8.3 Formulation Validation

As a check on the general form given by eq(8-30), consider the gravity vector (assuming spherical planet) given by

$$\bar{g} = -\mu\frac{\bar{r}}{r^3} \tag{8-31}$$

and consequently,

$$\frac{\partial\bar{g}}{\partial\bar{r}} = -\frac{\mu}{r^3}I + 3\frac{\mu}{r^5}\bar{r}\bar{r}^T \tag{8-32}$$

Therefore,

$$G = B^T\frac{\partial\bar{g}}{\partial\bar{r}}B = -\frac{\mu I}{r^3} + 3\frac{\mu}{r^5}B^T\bar{r}\bar{r}^TB \tag{8-33}$$

but we defined b to be equal to $B^T\bar{r}$. Therefore,

$$G = -\frac{\mu I}{r^3} + 3\frac{\mu}{r^5}bb^T \tag{8-34}$$

or,

$$G = \frac{3\mu}{r^3}\begin{bmatrix} b_1^2 - \frac{1}{3} & b_1 b_2 & b_1 b_3 \\ b_1 b_2 & b_2^2 - \frac{1}{3} & b_3 b_2 \\ b_1 b_3 & b_2 b_3 & b_3^2 - \frac{1}{3} \end{bmatrix} \tag{8-35}$$

Substituting the G elements from eq(8-35) into eq(8-30) gives

$$\bar{\tau} = \frac{3\mu}{r^3}\begin{bmatrix} b_2 b_3 (I_{zz} - I_{yy}) & + b_1 b_3 I_{xy} & -b_1 b_2 I_{xz} & + I_{yz}(b_3^2 - b_2^2) \\ b_1 b_3 (I_{xx} - I_{zz}) & -b_2 b_3 I_{xy} & + b_1 b_2 I_{yz} & + I_{xz}(b_1^2 - b_3^2) \\ b_1 b_2 (I_{yy} - I_{xx}) & + b_2 b_3 I_{xz} & -b_1 b_3 I_{yz} & + I_{xy}(b_2^2 - b_1^2) \end{bmatrix} \tag{8-36}$$

which is identical to eq(8-16). This lends confidence that the general expression is correct and that the spherical case is contained in the more general expansion as a special case where all higher gravitational harmonics are zero.

The formulae given here were coded in ASDS [9] and checked against a FORTRAN program containing Roithmayr's method for n=2, m=0 (J2 only). The data used and the results obtained are given in Appendix B.

Further data was run for a 4x4 gravity model. Note that the gravity gradient does change as the model size increases. This result is expected to be especially important during control when longitude effects will vary much more quickly than those due to latitude.

9.0 Geomagnetic Field

The magnetic field for the earth is also defined in terms of Legendre functions. It was felt that it would be useful to include a derivation for the magnetic field in this report, since so much of the gravity algorithm can be used.

The potential function, V, for the magnetic field is given [10], [11] as

$$V = a\sum_{n=1}^{\infty}\sum_{m=0}^{n}\left(\frac{a}{r}\right)^{n+1} P_n^m(\cos\theta)\,(g_n^m\cos m\lambda + h_n^m\sin m\lambda) \tag{9-1}$$

where a is the mean radius of the earth, r is position magnitude, θ is the geocentric colatitude, g_n^m and h_n^m are the spherical harmonic coefficients, and λ is the longitude. The P_n^m in eq(9-1) are not derived Legendre functions, rather they are Schmidt normalized associated Legendre functions of degree n and order m, defined by

$$P_n^m(\cos\theta) \equiv \left[\frac{2\,(n-m)!}{(n+m)!} - \delta_{0m}\right]^{\frac{1}{2}} P_{n,m}(\cos\theta) \tag{9-2}$$

The cosine of the colatitude is the same as the sine of the latitude, which was denoted by ε in eq(3-3). Also, define

$$\begin{aligned} C_{n,m} &\equiv \left[\frac{2\,(n-m)!}{(n+m)!} - \delta_{0m}\right]^{\frac{1}{2}} g_n^m \\ S_{n,m} &\equiv \left[\frac{2\,(n-m)!}{(n+m)!} - \delta_{0m}\right]^{\frac{1}{2}} h_n^m \end{aligned} \tag{9-3}$$

It is now possible to write eq(9-1) as

$$V = \sum_{n=1}^{\infty}\sum_{m=0}^{n}\frac{a^2}{r}\left(\frac{a}{r}\right)^n P_{n,m}(\varepsilon)\,(C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda) \tag{9-4}$$

This is not quite the form of the gravitational potential since the sum on n started at 2 in eq(3-5). Note also the appearance of a^2 in eq(9-4) in lieu of μ in eq(3-5).

Using the definitions in eq(3-6) and eq(3-9), and separating out the n = 1 term, eq(9-4) can be written

$$V = \frac{a^3}{r^2}\left(P_1^0 C_{1,0} + \left(C_{1,1}\frac{x_1}{r} + S_{1,1}\frac{x_2}{r}\right)P_1^1\right) + \sum_{n=2}^{\infty}\sum_{m=0}^{n}\frac{a^2}{r}\left(\frac{a}{r}\right)^n \frac{P_n^m}{r^m} B_{n,m} \tag{9-5}$$

Where now, the P_n^m are derived Legendre functions. The double sum part of eq(9-5) is now identical in form to the double sum part of eq(3-12), the only difference being that a^2 replaces μ. Using the definitions of P_1^0 and P_1^1 in eq(3-13), the leading term of eq(9-5) can be written

$$V_1 \equiv \frac{a^3}{r^2}\left(P_1^0 C_{1,0} + \left(C_{1,1}\frac{x_1}{r} + S_{1,1}\frac{x_2}{r}\right)P_1^1\right) = \frac{a^3}{r^3}\left(C_{1,1}x_1 + S_{1,1}x_2 + C_{1,0}x_3\right) \tag{9-6}$$

The magnetic potential function can now be expressed as

$$V = V_1 + \sum_{n=2}^{\infty} \sum_{m=0}^{n} \frac{a^2}{r} \left(\frac{a}{r}\right)^n \frac{P_n^m}{r^m} B_{n,m} \tag{9-7}$$

The magnetic field vector, B, is computed as

$$B = -\frac{\partial V}{\partial X} \tag{9-8}$$

The partials of the double sum part of V will look exactly as they did for gravity (assuming a^2 replaces μ). That being the case, the definitions of Γ_n, H_n, J_n, and K_n are the same as given in eq(4-11). This allows $\frac{\partial V}{\partial X}$ to be written

$$\frac{\partial V}{\partial X} = \frac{\partial V_1}{\partial X} - \frac{a^2}{r^2}\left(\sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n \Gamma_n\right)\hat{X} - \frac{a^2}{r^2}\left(\sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n H_n\right)\varepsilon\hat{X} + \frac{a^2}{r^2}\sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n \begin{bmatrix} J_n \\ K_n \\ H_n \end{bmatrix} \tag{9-9}$$

Computing the partial of V_1 results in

$$\frac{\partial V_1}{\partial X} = \frac{a^3}{r^3}\begin{bmatrix} C_{1,1} \\ S_{1,1} \\ C_{1,0} \end{bmatrix} - 3\frac{a^3}{r^4}\left(C_{1,1}x_1 + S_{1,1}x_2 + C_{1,0}x_3\right)\hat{X} \tag{9-10}$$

The negative of this result can be found in [12] as the field resulting from a magnetic dipole. Now, defining

$$\begin{aligned} J &\equiv \frac{a}{r}C_{1,1} + \sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n J_n \\ K &\equiv \frac{a}{r}S_{1,1} + \sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n K_n \\ \Gamma &\equiv \frac{a}{r^2}\left(3C_{1,1}x_1 + 3S_{1,1}x_2 + 2C_{1,0}x_3\right) + \sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n \Gamma_n \\ H &\equiv \frac{a}{r}C_{1,0} + \sum_{n=2}^{\infty} \left(\frac{a}{r}\right)^n H_n \end{aligned} \tag{9-11}$$

The equation for $\frac{\partial V}{\partial X}$ can be written

$$\frac{\partial V}{\partial X} = -\frac{a^2}{r^2}\Gamma\hat{X} - \frac{a^2}{r^2}H\varepsilon\hat{X} + \frac{a^2}{r^2}\begin{bmatrix} J \\ K \\ H \end{bmatrix} \tag{9-12}$$

Finally, then, the equation for the magnetic field vector, B, becomes

$$B = -\frac{\partial V}{\partial X} = \frac{a^2}{r^2}\left\{(\Gamma + \varepsilon H)\hat{X} - \begin{bmatrix} J \\ K \\ H \end{bmatrix}\right\} = \frac{a^2}{r^2}\left\{\Lambda\hat{X} - \begin{bmatrix} J \\ K \\ H \end{bmatrix}\right\} \tag{9-13}$$

Note that eq(9-13) is the same as eq(4-13) with $-\mu$ replaced by a^2. Assuming that equal degree and order were desired, a great deal of time could be saved by computing the gravitational acceleration and the magnetic field vector together, since they both would use the same state, the same derived Legendre functions, and all of the sums would be of exactly the same form. An examination of the code given in Appendix A for the fast_gravity_model and the fast_magnetic_model will show that they are almost entirely the same. The magnetic field vector resulting from IGRF 1985 data at a given position vector is given in Appendix B.

10.0 Conclusions

Derivations of the first and second partials of the gravitational potential have been given in both normalized and unnormalized form. Two different recursion relationships for the derived Legendre functions were considered. Code for both a "fast" unnormalized gravity model and a normalized gravity model using the best recursion relationship were given in Appendix A. Speed comparisons made in [13] indicate that the model in [2] ran at essentially the same speed as other gravity models in general use at JSC. The plots in Appendix B show that the "fast" algorithm is always faster than the model in [2], in some cases by as much as 20%, and consequently should be that much faster than other models in general use at JSC. The normalized models are faster than the model in [2] out through degree and order 27. Beyond that, the extra multiplications inherent in the normalized approach begin to outweigh the savings gained by precomputing all possible terms. Gravitational acceleration computed by the normalized and unnormalized models agreed through the 15th (out of 16) significant digit, out through degree and order 50, for a variety of initial states. For larger models, one would probably be safer using code based on the recursion relationship from [3].

A gravity gradient torque derivation for a general gravity model was given as well as code and data showing that the torque agrees with another model restricted to J2 only.

A derivation of the magnetic field vector was given. The derivation was given in a form that was as close as possible to the form of the gravity derivation. An examination of the resulting code shows that the two algorithms are almost totally alike.

11.0 Acknowledgments

The author wishes to express his appreciation to Mike Fraietta for checking equations, to Doug Neal who first typed the bulk of this document into Framemaker, to Bill Lear for many suggestions related to the gravity description, and to Carlos Roithmayr for his suggestions related to the gravity description and his motivating the author to include a section on the magnetic field vector.

12.0 References

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

2 Gottlieb, R.G., "A Fast Recursive Singularity Free Algorithm for Calculating the First and Second Derivatives of the Geopotential", JSC-23762, July 6, 1990.

3 Lundberg, J.B. and Schutz, B.E., "Recursion Formulas of Legendre Functions for Use with Nonsingular Geopotential Models", J. Guidance, Vol. 11, No. 1, Jan-Feb 1988

4 Lee, T., "Formulation of GTDS Gravity Modeling in Terms of Normalized Associated Legendre Polynomials", GTDS Formulation, GSFC, Nov, 1989

5 Roithmayr, C.M., "Gravitational Moment Exerted by an Oblate Body on a Small Body", Journal of Guidance, Control, and Dynamics, Vol. 12, May-June 1989, pg. 441-444.

6 Gottlieb, R.G., "Gravity Gradient Torque", MDSSC TM-FM8EA-5, 30 September 1988.

7 Glandorf, D.R., "Gravity Gradient Torque for an Arbitrary Potential Function", Journal of Guidance, Vol.9,No.1, Jan-Feb 1986.

8 Wilcox, J. C., Comment on "Gravity Gradient Torque for an Arbitrary Potential Function", J. Guidance, Vol.10, No. 2, March-April 1987

9 Gottlieb, R.G., Fraietta, M.F., et al, "Ada Simulation Development System (ASDS), Version 2.0 Documentation", McDonnell Douglas TM 5.23.08.103, 23 Jan 1992.

10 Peddie, N.W., "International Geomagnetic Reference Field: the Third Generation", J. Geomag. Geoelectr., Vol. 34, 1982, pp 309-326.

11 Barraclough, D.R., "International Geomagnetic Reference Field: the Fourth Generation", Physics of the Earth and Planetary Interiors, Vol. 48, 1987, pp 279-292.

12 Roithmayr, C., "Contributions of Spherical Harmonics to Magnetic and Gravitational Fields", (Draft), NASA JSC, Dec. 1992

13 Lear, W.M., "Subroutines to Compute the Gravitational Acceleration", JSC-25469, February 1992

14 Lerch, F.J., et al, "Gravity Model Improvement using GEOS-3 (GEM-9 and GEM-10)" Journal of Geophysical Research, Vol 84, No B8, July 30, 1979

15 Lerch, F.J., et al, "Geopotential Models of the Earth From Satellite Tracking, Altimeter and Surface Gravity Observations: GEM-T3 and GEM-T3S", NASA Technical Memorandum 104555, GSFC, January 1992.

Appendix A

Ada Code

A.1 Spec of Fast_Gravity_Model

```
with Real_Types;
use Real_Types;
with Vector_Matrix_3;
use Vector_Matrix_3;

package fast_Gravity_Model is

  Max_Gravity_Model_Name_Length : constant Positive := 80;
  max_degree_and_order : constant Positive := 50;

  type Data_Coefficient_Array is
    array (Natural range <>, Natural range <>) of Real;

  type gravity_array is array(0..max_degree_and_order+2) of real;
  type ga_ptr is access gravity_array;
  type gravity_array_2 is array(0..max_degree_and_order) of  ga_ptr;

  type Gravity_Model_Data is private;
  ------------------------------------------------------------------------
  function Create_Gravity_Model (Name_In    : String;
                                 C, S       : Data_Coefficient_Array;
                             Mu, Radius : Real) return Gravity_Model_Data;
  ------------------------------------------------------------------------
  procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                    X        : in  Vector_3;
                    R        : in  Real;
                    Want_Central_Force : in  Boolean;
                    Nax, Max : in  Natural;
                    G        : out Vector_3); --no potential
  ------------------------------------------------------------------------
  procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                    X        : in  Vector_3;
                    R        : in  Real;
                    Want_Central_Force : in  Boolean;
                    Nax, Max : in  Natural;
                    Pot      : out Real;
                    G        : out Vector_3;
                    Dgdx     : out Matrix_3x3);--pot and dgdx
private

  type Gravity_Model_Data is -- defaulted to point mass gem_9
    record
      Name          : String (1 .. Max_Gravity_Model_Name_Length);
      Name_Length   : Integer;
      C             : gravity_array_2;
      S             : gravity_array_2;
      Mu            : Real := 398_600.47E9; -- planet gravitational constant(m**3/s**2)
      Radius        : Real := 6_378_139.0;  -- planet equatorial radius (m)
      Model_Max_Size :Natural;              -- max size current gravity model data
    end record;

end fast_Gravity_Model;
```

A.2 Body of Fast_Gravity_Model

```
with Extended_Range_Combinatoric_Functions;
use Extended_Range_Combinatoric_Functions;
with Exponential_Logarithm_Functions;
use Exponential_Logarithm_Functions;

package body fast_Gravity_Model is

 Default_Gmd              : Gravity_Model_Data;
 Have_Set_Default_Gravity  : Boolean := False;
 Gravity_Model_Name_Too_Long : exception;
 bad_gravity_data : exception;
 twonm1,twonm1on,nm1on : gravity_array;

 P : gravity_array_2 := (others => new gravity_array);
 ---------------------------------------------------------------------

 procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                   X        : in  Vector_3;
                   R        : in  Real;
                   Want_Central_Force : in  Boolean;
                   Nax, Max : in  Natural;
                   G        : out Vector_3) is


  Ri, Xovr, Yovr, Zovr, Ep : Real;
  Muor, Muor2, Reor, Reorn,Sum_Init  : Real;
  ctil, stil : gravity_array;
  Sumh, Sumgam, Sumj, Sumk, Sumh_N,  Lambda : Real;
  pnm,cnm,snm,ctmm1,stmm1 : real;
  Sumgam_N, Sumj_N, Sumk_N, Mxpnm, Npmp1 : Real;
  Bnmtil,n_const : Real;
  Mm1, Mm2, Mp1, Nm1, Lim ,nm2: Integer;
  pn,pnm1,pnm2 :  ga_ptr;
  cn,sn : ga_ptr;

 begin

  Ri   := 1.0 / R;
  Xovr  := X (1) * Ri;
  Yovr  := X (2) * Ri;
  Zovr  := X (3) * Ri;
  Ep    := Zovr;
  Reor  := gmd.Radius * Ri;
  Reorn := Reor;
  Muor  := gmd.Mu * Ri;
  Muor2 := Muor * Ri;

  Case Want_Central_Force is
   When true  => Sum_Init := 1.0;
   When false => Sum_Init := 0.0;
  end case;

  ctil (0) := 1.0; ctil (1) := Xovr;
```

```
stil (0) := 0.0; stil (1) := Yovr;
Sumh     := 0.0;
Sumj     := 0.0;
Sumk     := 0.0;
Sumgam   := Sum_Init;

p(1)(0) := ep;
for N in 2 .. Nax loop
  n_const := twonm1(n);
  Reorn  := Reorn * Reor;
  pn := p(n);
  cn := gmd.c(n);
  sn := gmd.s(n);
  nm1 := n - 1;
  nm2 := n - 2;
  pnm1 := p(nm1);
  pnm2 := p(nm2);
  Pn(nm1) := ep*Pn(n);
  Pn(0) := Twonm1on(n)*Ep*Pnm1(0) - Nm1on(n)*Pnm2(0);
  Pn(1) := Pnm2(1) + n_const * Pnm1(0);
  Sumh_N   := Pn (1) * Cn(0);
  Sumgam_N := Pn (0) * Cn(0) * real(n + 1);

  if Max > 0 then
  for m in 2..nm2 loop
    Pn(m) := Pnm2(m) + n_const * Pnm1(m-1);
  end loop;  --Have all derived Legendre functions
    Sumj_N := 0.0;
    Sumk_N := 0.0;

    ctil (N) := ctil (1) * ctil (Nm1) - stil (1) * stil (Nm1);
    stil (N) := stil (1) * ctil (Nm1) + ctil (1) * stil (Nm1);

    if N < Max then
      Lim := N;
    else
      Lim := Max;
    end if;
    for M in 1 .. Lim loop
      Mm1   := M - 1;
      Mp1   := M + 1;
      Npmp1 := Real (N + Mp1);
      pnm := pn(m);
      cnm := cn(m);
      snm := sn(m);
      ctmm1 := ctil(mm1);
      stmm1 := stil(mm1);

      Mxpnm    := Real (M) * Pnm;
      Bnmtil   := Cnm * ctil (M) + Snm * stil (M);
      Sumh_N   := Sumh_N + Pn(mp1) * Bnmtil;
      Sumgam_N := Sumgam_N + Npmp1 * Pnm * Bnmtil;
      Sumj_N   := Sumj_N + Mxpnm * (Cnm*ctmm1 + Snm*stmm1);
      Sumk_N   := Sumk_N - Mxpnm * (Cnm*stmm1 - Snm*ctmm1);
```

```
      end loop;
      Sumj := Sumj + Reorn * Sumj_N;
      Sumk := Sumk + Reorn * Sumk_N;
    end if;

    ---- SUMS BELOW HERE HAVE VALUES WHEN M := 0

    Sumh   := Sumh + Reorn * Sumh_N;
    Sumgam := Sumgam + Reorn * Sumgam_N;
  end loop;

  Lambda := Sumgam + Ep * Sumh;
  G (1)  := -Muor2 * (Lambda * Xovr - Sumj);
  G (2)  := -Muor2 * (Lambda * Yovr - Sumk);
  G (3)  := -Muor2 * (Lambda * Zovr - Sumh);

end Gotpot;

-----------------------------------------------------------------
procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                  X        : in  Vector_3;
                  R        : in  Real;
                  Want_Central_Force : in  Boolean;
                  Nax, Max : in  Natural;
                  Pot      : out Real;
                  G        : out Vector_3;
                  Dgdx     : out Matrix_3x3) is

  Mu     : Real          renames Gmd.Mu;
  Radius : Real          renames Gmd.Radius;

  Ri, Xovr, Yovr, Zovr, Ep,Sum_Init,n_const : Real;
  Muor, Muor2, Muor3, Reor, Reorn, Sumv, Gg, Ff, D1, D2 : Real;
  ctil, stil : gravity_array;
  Sumh, Sumgam, Sumj, Sumk, Sumh_N, Np1, Lambda : Real;
  Suml, Summ, Sumn, Sumo, Sump, Sumq, Sumr, Sums, Sumt : Real;
  Suml_N, Summ_N, Sumn_N, Sumo_N, Sump_N, Sumq_N : Real;
  Sumr_N, Sums_N, Sumt_N, Temp : Real;
  Sumgam_N, Sumj_N, Sumk_N, Mxpnm,  Npmp1 : Real;
  Sumv_N, Amntil, Bnmtil, Pnmbnm, Anmtm1, Bnmtm1 : Real;
  Mm1, Mm2, Mp1, Mp2, Nm1, Nm2 , Lim : Integer;
  pnm , pnmp1, cnm,snm, ctmm1, stmm1,cn0 : real;
  pn,pnm1,pnm2,cn,sn : ga_ptr;

begin

  Ri   := 1.0 / R;
  Xovr := X (1) * Ri;
  Yovr := X (2) * Ri;
  Zovr := X (3) * Ri;
  Ep   := Zovr;
  Reor := Radius * Ri;
  Reorn := Reor;
  Muor := Mu * Ri;
```

```
Muor2 := Muor * Ri;

Case Want_Central_Force is
  When true  => Sum_Init := 1.0;
  When false => Sum_Init := 0.0;
end case;

ctil (0) := 1.0; ctil (1) := Xovr;
stil (0) := 0.0; stil (1) := Yovr;
Sumv    := Sum_Init;
Sumh    := 0.0;
Sumj    := 0.0;
Sumk    := 0.0;
Sumgam  := Sum_Init;
Summ    := 0.0;
Sumn    := 0.0;
Sumo    := 0.0;
Sump    := 0.0;
Sumq    := 0.0;
Sumr    := 0.0;
Sums    := 0.0;
Sumt    := 0.0;
Suml    := 2.0 * Sum_Init;

p(1)(0) := ep;
for N in 2 .. Nax loop
  n_const := Twonm1(n);
  Reorn  := Reorn * Reor;
  nm1 := n - 1;
  nm2 := n - 2;
  pn := p(n);
  pnm1 := p(nm1);
  pnm2 := p(nm2);
  Pn(nm1) := ep*Pn(n);
  Pn(0) := Twonm1on(n)*Ep*Pnm1(0) - Nm1on(n)*Pnm2(0);
  Pn(1) := Pnm2(1) + n_const * Pnm1(0);
  cn := gmd.c(n);
  sn := gmd.s(n);
  np1 := real(n+1);
  Cn0 := Cn(0);
  Sumv_N   := Pn (0) * Cn0;
  Sumh_N   := Pn (1) * Cn0;
  Summ_N   := Pn (2) * Cn0;
  Sumgam_N := Sumv_N * Np1;
  Sump_N   := Sumh_N * Np1;
  Suml_N   := Sumgam_N * (Np1 + 1.0);

  if Max > 0 then
  for m in 2..nm2 loop
    Pn(m) := Pnm2(m) + n_const * Pnm1(m-1);
  end loop;

    Sumj_N := 0.0;
    Sumk_N := 0.0;
```

```
Sumn_N := 0.0;
Sumo_N := 0.0;
Sumq_N := 0.0;
Sumr_N := 0.0;
Sums_N := 0.0;
Sumt_N := 0.0;

nm1 := n - 1;
ctil (N) := ctil (1) * ctil (Nm1) - stil (1) * stil (Nm1);
stil (N) := stil (1) * ctil (Nm1) + ctil (1) * stil (Nm1);

if N < Max then
  Lim := N;
else
  Lim := Max;
end if;
for M in 1 .. Lim loop
  Mm1   := M - 1;
  Mp1   := M + 1;
  Mp2   := M + 2;
  Npmp1 := Real (N + Mp1);

  pnm := pn(m);
  pnmp1 := pn(mp1);
  cnm := cn(m);
  snm := sn(m);
  ctmm1 := ctil(mm1);
  stmm1 := stil(mm1);

  Mxpnm    := Real (M) * pnm;
  Bnmtil   := cnm * ctil (M) + snm * stil (M);
  Pnmbnm   := Pnm * Bnmtil;
  Sumv_N   := Sumv_N + Pnmbnm;
  Bnmtm1   := CNm * ctMm1 + SNm * stMm1;
  Anmtm1   := CNm * stMm1 - SNm * ctMm1;
  Sumh_N   := Sumh_N + Pn (Mp1) * Bnmtil;
  Sumgam_N := Sumgam_N + Npmp1 * Pnmbnm;
  Sumj_N   := Sumj_N + Mxpnm * Bnmtm1;
  Sumk_N   := Sumk_N - Mxpnm * Anmtm1;
  Suml_N   := Suml_N + Npmp1 * (Real (Mp1) + Np1) *pnmBnm;
  Summ_N   := Summ_N + Pn(Mp2) * Bnmtil;
  Sump_N   := Sump_N + Npmp1 * PnMp1 * Bnmtil;
  Sumq_N   := Sumq_N + Real (M) * PnMp1 * Bnmtm1;
  Sumr_N   := Sumr_N - Real (M) * PnMp1 * Anmtm1;
  Sums_N   := Sums_N + Npmp1 * Mxpnm * Bnmtm1;
  Sumt_N   := Sumt_N - Npmp1 * Mxpnm * Anmtm1;
  if (M >= 2) then
    Mm2    := M - 2;
    Sumn_N := Sumn_N + Real (Mm1) * Mxpnm *
                  (CNm * ctil (Mm2) + SNm*stil(Mm2));
    Sumo_N := Sumo_N + Real (Mm1) * Mxpnm *
                  (CNm * stil (Mm2) - SNm*ctil(Mm2));
  end if;
```

```
      end loop;
      Sumj := Sumj + Reorn * Sumj_N;
      Sumk := Sumk + Reorn * Sumk_N;
      Sumn := Sumn + Reorn * Sumn_N;
      Sumo := Sumo + Reorn * Sumo_N;
      Sumq := Sumq + Reorn * Sumq_N;
      Sumr := Sumr + Reorn * Sumr_N;
      Sums := Sums + Reorn * Sums_N;
      Sumt := Sumt + Reorn * Sumt_N;
    end if;
    ---- SUMS BELOW HERE HAVE VALUES WHEN M := 0
    Sumv   := Sumv + Reorn * Sumv_N;
    Sumh   := Sumh + Reorn * Sumh_N;
    Sumgam := Sumgam + Reorn * Sumgam_N;
    Suml   := Suml + Reorn * Suml_N;
    Summ   := Summ + Reorn * Summ_N;
    Sump   := Sump + Reorn * Sump_N;
  end loop;

  Pot    := Muor * Sumv;
  Lambda := Sumgam + Ep * Sumh;
  G (1)  := -Muor2 * (Lambda * Xovr - Sumj);
  G (2)  := -Muor2 * (Lambda * Yovr - Sumk);
  G (3)  := -Muor2 * (Lambda * Zovr - Sumh);

  -- Need to construct second partial matrix_3x3
  Gg         := -(Summ * Ep + Sump + Sumh);
  Ff         := Suml + Lambda + Ep * (Sump + Sumh - Gg);
  D1         := Ep * Sumq + Sums;
  D2         := Ep * Sumr + Sumt;
  Muor3      := Muor2 * Ri;
  Dgdx (1, 1) := Muor3 * ((Ff * Xovr - 2.0 * D1) * Xovr - Lambda + Sumn);
  Dgdx (2, 2) := Muor3 * ((Ff * Yovr - 2.0 * D2) * Yovr - Lambda - Sumn);
  Dgdx (3, 3) := Muor3 * ((Ff * Zovr + 2.0 * Gg) * Zovr - Lambda + Summ);
  Temp       := Muor3 * ((Ff * Yovr - D2) * Xovr - D1 * Yovr - Sumo);
  Dgdx (1, 2) := Temp;
  Dgdx (2, 1) := Temp;
  Temp       := Muor3 * ((Ff * Xovr - D1) * Zovr + Gg * Xovr + Sumq);
  Dgdx (1, 3) := Temp;
  Dgdx (3, 1) := Temp;
  Temp       := Muor3 * ((Ff * Yovr - D2) * Zovr + Gg * Yovr + Sumr);
  Dgdx (2, 3) := Temp;
  Dgdx (3, 2) := Temp;

end Gotpot;

-------------------------------------------------------------------------
function Create_Gravity_Model (Name_In    : String;
                               C, S       : Data_Coefficient_Array;
                               Mu, Radius : Real) return Gravity_Model_Data is
  Gmd    : Gravity_Model_Data;
  Coef   : Real;
  n_max : Integer := C'Last (1);
begin
```

```
    if n_max < 2 then raise bad_gravity_data;end if;

    gmd.c := (others => new gravity_array);
    gmd.s := (others => new gravity_array);
    -- Unnormalize gravity model coefficients
    for N in C'Range  loop
     for M in 0 .. N loop
       if M = 0 then
         Gmd.C (N)(0) := Sqrt(Real (2 * N + 1)) * C (N, 0) * 1.0E-6;
         Gmd.S (N)(0) := 0.0;
       else
         Coef         :=
           Sqrt (Real (2 * (2 * N + 1)) * Factorial_Ratio (N - M, N + M)) *
             1.0E-6;
         Gmd.C (N)( M) := Coef * C (N, M);
         Gmd.S (N)( M) := Coef * S (N, M);
       end if;
     end loop;
    end loop;
    Gmd.Mu          := Mu;
    Gmd.Radius      := Radius;
    Gmd.Name_Length := Name_In'Length;
    if Gmd.Name_Length > Max_Gravity_Model_Name_Length then
     raise Gravity_Model_Name_Too_Long;
    end if;
    Gmd.Name := (others => Ascii.Nul);
    Gmd.Name (1 .. Gmd.Name_Length) := Name_In;
    Gmd.Model_Max_Size := n_max;

    if Have_Set_Default_Gravity then
     return Gmd;
    else
     Default_Gmd := Gmd;
     return Gmd;
    end if;
  end Create_Gravity_Model;

  ------------------------------------------------------------------------
begin


  p(0)(0) := 1.0; p(0)(1) := 0.0; p(0)(2) := 0.0;
  p(1)(1) := 1.0; p(1)(2) := 0.0; p(1)(3) := 0.0;
  for n in 2..Max_Degree_And_Order loop
     p(n)(n) := p(n-1)(n-1)*real(2*n-1);
     p(n)(n+1) := 0.0;
     p(n)(n+2) := 0.0;
     twonm1(n) := real(2*n - 1);
     twonm1on(n) := twonm1(n)/real(n);
     nm1on(n) := real(n - 1)/real(n);
  end loop;

end fast_Gravity_Model;
```

A.3 Body of Normalized_Gravity_Model

```
with mathematical_constants;
with exponential_logarithm_functions;
use exponential_logarithm_functions;

package body Normalized_Gravity_Model is

 sqrt3:constant real := mathematical_constants.square_root_of_three;
 Gravity_Model_Name_Too_Long : exception;
 bad_gravity_data : exception;

 P     : gravity_array_2 := (others => new gravity_array);
 p1 :ga_ptr := p(1);

 xi    : gravity_array_2 := (others => new gravity_array);
 eta   : gravity_array_2 := (others => new gravity_array);
 zeta  : gravity_array_2 := (others => new gravity_array);
 upsilon: gravity_array_2 := (others => new gravity_array);
 alpha : gravity_array  ;
 beta  : gravity_array  ;
 nrdiag : gravity_array  ;
 num,den: integer;

-----------------------------------------------------------------
function Create_Gravity_Model (Name_In   : String;
                               C, S      : Data_Coefficient_Array;
                           Mu, Radius : Real) return Gravity_Model_Data is
 Gmd   : Gravity_Model_Data;
  Coef  : Real;
  n_max : Integer := C'Last (1);
 begin
   if n_max < 2 then raise bad_gravity_data;end if;

  gmd.c := (others => new gravity_array);
  gmd.s := (others => new gravity_array);
  -- scale gravity model coefficients
  for N in C'Range  loop
   for M in 0 .. N loop
     Gmd.C (N)( M) := 1.0e-6 * C (N, M);--Just scale coefficients
     Gmd.S (N)( M) := 1.0e-6 * S (N, M);--Just scale coefficients
   end loop;
  end loop;
  Gmd.Mu          := Mu;
  Gmd.Radius      := Radius;
  Gmd.Name_Length := Name_In'Length;
  if Gmd.Name_Length > Max_Gravity_Model_Name_Length then
   raise Gravity_Model_Name_Too_Long;
  end if;
  Gmd.Name := (others => Ascii.Nul);
  Gmd.Name (1 .. Gmd.Name_Length) := Name_In;
  Gmd.Model_Max_Size := n_max;

  return Gmd;
```

```
  end Create_Gravity_Model;

-------------------------------------------------------------------
 procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                   X        : in  Vector_3;
                   R        : in  Real;
                   Want_Central_Force : in  Boolean;
                   Nax, Max : in  Natural;
                   G        : out Vector_3) is


  Ctil, Stil : gravity_array;
  Ri, Xovr, Yovr, Zovr, Ep,Sum_Init : Real;
  Muor, Muor2,  Reor, Reorn : Real;
  Sumh,  Sumgam,  Sumj,  Sumk,  Np1,  Lambda : Real;
  Sumh_N,Sumgam_N, Sumj_N, Sumk_N, Mxpnm, Npmp1 : Real;
  Bnmtil,pnm,snm,cnm,ctmm1,stmm1 : Real;
  Mm1, Mp1, Nm1, nm2,Lim: Integer;
  pn,pnm1,pnm2 :  ga_ptr;
  cn,sn,zn,xin,etn : ga_ptr;

 begin

  Ri   := 1.0 / R;
  Xovr  := X (1) * Ri;
  Yovr  := X (2) * Ri;
  Zovr  := X (3) * Ri;
  Ep    := Zovr;
  Reor  := gmd.Radius * Ri;
  Reorn := Reor;
  Muor  := gmd.Mu * Ri;
  Muor2 := Muor * Ri;

  Case Want_Central_Force is
   When true  => Sum_Init := 1.0;
   When false => Sum_Init := 0.0;
  end case;

  Ctil (0) := 1.0; Ctil (1) := Xovr;
  Stil (0) := 0.0; Stil (1) := Yovr;
  Sumh     := 0.0;
  Sumj     := 0.0;
  Sumk     := 0.0;
  Sumgam   := Sum_Init;

  p1(0) := sqrt3*ep;
  for N in 2 .. Nax loop
   Reorn  := Reorn * Reor;
   pn := p(n);
   cn := gmd.c(n);
   sn := gmd.s(n);
   zn := zeta(n);
   xin := xi(n);
   etn := eta(n);
```

```
nm1 := n-1;
nm2 := n-2;
pnm1 := p(nm1);
pnm2 := p(nm2);
Pn(0) := alpha(n)*Ep*Pnm1(0) - beta(n)*Pnm2(0);
Pn(nm1) := ep*nrdiag(n);
Pn(1) :=  xin(1)*ep*Pnm1(1) - etn(1)* Pnm2(1);
Sumh_N   := zn(0)*Pn(1) * Cn(0);
Sumgam_N := Pn(0) * Cn(0) * real(n + 1);

if Max > 0 then
for m in 2..nm2 loop
  Pn(m) :=  xin(m)*ep*Pnm1(m) - etn(m)* Pnm2(m);
end loop;  --got all the Legendre functions now

  Sumj_N := 0.0;
  Sumk_N := 0.0;

  Ctil (N) := Ctil (1) * Ctil (Nm1) - Stil (1) * Stil (Nm1);
  Stil (N) := Stil (1) * Ctil (Nm1) + Ctil (1) * Stil (Nm1);


  if N < Max then
    Lim := n;
  else
    Lim := Max;
  end if;
  for M in 1 .. Lim loop
    Mm1   := M - 1;
    Mp1   := M + 1;
    Npmp1 := Real (N + Mp1);
    pnm := pn(m);
    cnm := cn(m);
    snm := sn(m);
    ctmm1 := ctil(mm1);
    stmm1 := stil(mm1);

    Mxpnm    := Real (m) * Pnm;
    Bnmtil   := Cnm * Ctil (M) + Snm * Stil (M);
    Sumh_N   := Sumh_N + Pn(mp1) * Bnmtil*zn(m);
    Sumgam_N := Sumgam_N + Npmp1 * Pnm * Bnmtil;
    Sumj_N   := Sumj_N + Mxpnm*(Cnm*ctmm1 + Snm*stmm1);
    Sumk_N   := Sumk_N - Mxpnm*(Cnm*stmm1 - Snm*ctmm1);
  end loop;
  Sumj := Sumj + Reorn * Sumj_N;
  Sumk := Sumk + Reorn * Sumk_N;
end if;

---- SUMS BELOW HERE HAVE VALUES WHEN M := 0

Sumh   := Sumh + Reorn * Sumh_N;
Sumgam := Sumgam + Reorn * Sumgam_N;

end loop;
```

```
  Lambda := Sumgam + Ep * Sumh;
  G (1)  := -Muor2 * (Lambda * Xovr - Sumj);
  G (2)  := -Muor2 * (Lambda * Yovr - Sumk);
  G (3)  := -Muor2 * (Lambda * Zovr - Sumh);

end Gotpot;

-------------------------------------------------------------
procedure Gotpot (Gmd      : in  Gravity_Model_Data;
                  X        : in  Vector_3;
                  R        : in  Real;
                  Want_Central_Force : in  Boolean;
                  Nax, Max : in  Natural;
                  Pot      : out Real;
                  G        : out Vector_3;
                  Dgdx     : out Matrix_3x3) is --pot and dgdx

  Ctil, Stil : gravity_array;

  Ri, Xovr, Yovr, Zovr, Ep,Sum_Init : Real;
  Muor, Muor2, Muor3, Reor, Reorn, Sumv, Gg, Ff, D1, D2 : Real;
  Sumh, Sumgam, Sumj, Sumk, Sumh_N, Np1, Lambda : Real;
  Suml, Summ, Sumn, Sumo, Sump, Sumq, Sumr, Sums, Sumt : Real;
  Suml_N, Summ_N, Sumn_N, Sumo_N, Sump_N, Sumq_N : Real;
  Sumr_N, Sums_N, Sumt_N, Temp : Real;
  Sumgam_N, Sumj_N, Sumk_N, Mxpnm, Npmp1 : Real;
  Sumv_N, Amntil, Bnmtil, Pnmbnm, Anmtm1, Bnmtm1 : Real;
  Mm1, Mm2, Mp1, Mp2, Nm1, Nm2 , Lim : Integer;
  pnm,pnmp1,cnm,snm, ctmm1, stmm1,z_pnmp1,cn0 : real;
  pn,pnm1,pnm2 :  ga_ptr;
  cn,sn,zn,upsn,xin,etn : ga_ptr;


begin

  Ri   := 1.0 / R;
  Xovr  := X (1) * Ri;
  Yovr  := X (2) * Ri;
  Zovr  := X (3) * Ri;
  Ep    := Zovr;
  Reor  := gmd.Radius * Ri;
  Reorn := Reor;
  Muor  := gmd.Mu * Ri;
  Muor2 := Muor * Ri;
  Muor3 := Muor2 * Ri;

  Case Want_Central_Force is
   When true  => Sum_Init := 1.0;
   When false => Sum_Init := 0.0;
  end case;

  ctil (0) := 1.0; ctil (1) := Xovr;
  stil (0) := 0.0; stil (1) := Yovr;
```

```
Sumv    := Sum_Init;
Sumh    := 0.0;
Sumj    := 0.0;
Sumk    := 0.0;
Sumgam  := Sum_Init;
Summ    := 0.0;
Sumn    := 0.0;
Sumo    := 0.0;
Sump    := 0.0;
Sumq    := 0.0;
Sumr    := 0.0;
Sums    := 0.0;
Sumt    := 0.0;
Suml    := 2.0 * Sum_Init;

p(1)(0) := sqrt3*ep;
for N in 2 .. Nax loop
  Reorn  := Reorn * Reor;
  pn := p(n);
  cn := gmd.c(n);
  sn := gmd.s(n);
  zn := zeta(n);
  xin := xi(n);
  etn := eta(n);
  nm1 := n - 1;
  nm2 := n - 2;
  pnm1 := p(nm1);
  pnm2 := p(nm2);
  Pn(0) := alpha(n)*Ep*Pnm1(0) - beta(n)*Pnm2(0);
  Pn(nm1) := ep*nrdiag(n);
  Pn(1) :=  xin(1)*ep*Pnm1(1) - etn(1)* Pnm2(1);
  upsn := upsilon(n);
  np1 := real(n+1);
  Cn0 := Cn(0);

  Sumv_N   := Pn (0) * Cn0;
  Sumh_N   := Pn (1) * Cn0  *zn(0);
  Summ_N   := Pn (2) * Cn0*upsn(0);
  Sumgam_N := Sumv_N * Np1;
  Sump_N   := Sumh_N * Np1;
  Suml_N   := Sumgam_N * (Np1 + 1.0);

  if Max > 0 then
    for m in 2..nm2 loop
      Pn(m) :=  xin(m)*ep*Pnm1(m) - etn(m)* Pnm2(m);
    end loop;  --got all the Legendre functions now

    Sumj_N := 0.0;
    Sumk_N := 0.0;
    Sumn_N := 0.0;
    Sumo_N := 0.0;
    Sumq_N := 0.0;
    Sumr_N := 0.0;
    Sums_N := 0.0;
```

```
  Sumt_N := 0.0;

  ctil (N) := ctil (1) * ctil (Nm1) - stil (1) * stil (Nm1);
  stil (N) := stil (1) * ctil (Nm1) + ctil (1) * stil (Nm1);

  if N < Max then
   Lim := N;
  else
   Lim := Max;
  end if;

  for M in 1 .. Lim loop
   Mm1  := M - 1;
   Mp1  := M + 1;
   Mp2  := M + 2;
   Npmp1 := Real (N + Mp1);

   pnm := pn(m);
   pnmp1 := pn(mp1);
   cnm := cn(m);
   snm := sn(m);
   ctmm1 := ctil(mm1);
   stmm1 := stil(mm1);

   Mxpnm   := Real (m) * Pnm;
   Bnmtil  := Cnm * Ctil (M) + Snm * Stil (M);
   Bnmtm1  := Cnm * ctMm1 + Snm * stMm1;
   Anmtm1  := Cnm * stMm1 - Snm * ctMm1;

   Pnmbnm  := Pnm * Bnmtil;
   Sumv_N  := Sumv_N + Pnmbnm;
  if m < n then
   z_pnmp1 := zn(m)*Pn(mp1);
   Sumh_N  := Sumh_N + z_pnmp1 * Bnmtil;
   Sump_N  := Sump_N + Npmp1   * z_PnMp1 * Bnmtil;
   Sumq_N  := Sumq_N + Real (M) * z_PnMp1 * Bnmtm1;
   Sumr_N  := Sumr_N - Real (M) * z_PnMp1 * Anmtm1;
  end if;
   Sumgam_N := Sumgam_N + Npmp1 * Pnmbnm;
   Sumj_N  := Sumj_N + Mxpnm*bnmtm1;
   Sumk_N  := Sumk_N - Mxpnm*anmtm1;
   Suml_N  := Suml_N + Npmp1 * (Real (Mp1) + Np1) *pnmBnm;
   Summ_N  := Summ_N + Pn(Mp2) * Bnmtil*upsn(m);
   Sums_N  := Sums_N + Npmp1 * Mxpnm * Bnmtm1;
   Sumt_N  := Sumt_N - Npmp1 * Mxpnm * Anmtm1;
   if (M >= 2) then
    Mm2   := M - 2;
    Sumn_N := Sumn_N + Real (Mm1) * Mxpnm *
                  (Cnm * ctil (Mm2) + Snm*stil(Mm2));
    Sumo_N := Sumo_N + Real (Mm1) * Mxpnm *
                  (Cnm * stil (Mm2) - Snm*ctil(Mm2));
   end if;

  end loop;
```

```
      Sumj := Sumj + Reorn * Sumj_N;
      Sumk := Sumk + Reorn * Sumk_N;
      Sumn := Sumn + Reorn * Sumn_N;
      Sumo := Sumo + Reorn * Sumo_N;
      Sumq := Sumq + Reorn * Sumq_N;
      Sumr := Sumr + Reorn * Sumr_N;
      Sums := Sums + Reorn * Sums_N;
      Sumt := Sumt + Reorn * Sumt_N;
    end if;

    ---- SUMS BELOW HERE HAVE VALUES WHEN M := 0

    Sumv   := Sumv + Reorn * Sumv_N;
    Sumh   := Sumh + Reorn * Sumh_N;
    Sumgam := Sumgam + Reorn * Sumgam_N;
    Suml   := Suml + Reorn * Suml_N;
    Summ   := Summ + Reorn * Summ_N;
    Sump   := Sump + Reorn * Sump_N;

  end loop;

  Pot    := Muor * Sumv;
  Lambda := Sumgam + Ep * Sumh;
  G (1)  := -Muor2 * (Lambda * Xovr - Sumj);
  G (2)  := -Muor2 * (Lambda * Yovr - Sumk);
  G (3)  := -Muor2 * (Lambda * Zovr - Sumh);

  -- Need to construct second partial matrix_3x3
  Gg         := -(Summ * Ep + Sump + Sumh);
  Ff         := Suml + Lambda + Ep * (Sump + Sumh - Gg);
  D1         := Ep * Sumq + Sums;
  D2         := Ep * Sumr + Sumt;
  Dgdx (1, 1) := Muor3 * ((Ff * Xovr - 2.0 * D1) * Xovr - Lambda + Sumn);
  Dgdx (2, 2) := Muor3 * ((Ff * Yovr - 2.0 * D2) * Yovr - Lambda - Sumn);
  Dgdx (3, 3) := Muor3 * ((Ff * Zovr + 2.0 * Gg) * Zovr - Lambda + Summ);
  Temp       := Muor3 * ((Ff * Yovr - D2) * Xovr - D1 * Yovr - Sumo);
  Dgdx (1, 2) := Temp;
  Dgdx (2, 1) := Temp;
  Temp       := Muor3 * ((Ff * Xovr - D1) * Zovr + Gg * Xovr + Sumq);
  Dgdx (1, 3) := Temp;
  Dgdx (3, 1) := Temp;
  Temp       := Muor3 * ((Ff * Yovr - D2) * Zovr + Gg * Yovr + Sumr);
  Dgdx (2, 3) := Temp;
  Dgdx (3, 2) := Temp;

 end Gotpot;

-------------------------------------------------------------------

BEGIN

 for n in 2..max_degree_and_order loop
  for m in 0 .. n-1 loop
   num := (2*n-1)*(2*n+1);
```

```
   den := (n+m)*(n-m);
    xi(n)(m) := sqrt( real(num)/real(den));
 end loop;
end loop;

for n in 2..max_degree_and_order loop
 for m in 0 .. n-1 loop
  num := (2*n+1)*(n+m-1)*(n-m-1);
  den := (n+m)*(n-m)*(2*n-3);
  if num = 0 then
   eta(n)(m) := 0.0;
  else
   eta(n)(m) := sqrt( real(num)/real(den));
  end if;
 end loop;
end loop;

for n in 2..max_degree_and_order loop
 for m in 0 .. n loop
  if m = 0 then
   num := n*(n+1);
   zeta(n)(0) := sqrt( real(num)/2.0);
  else
   num := (n-m)*(n+m+1);
   if num = 0 then
    zeta(n)(m) := 0.0;
   else
    zeta(n)(m) := sqrt(real(num));
   end if;
  end if;
 end loop;
end loop;

for n in 2..max_degree_and_order loop
 for m in 0 .. n loop
  if m = 0 then
   num := n*(n-1)*(n+1)*(n+2);
   upsilon(n)(0) := sqrt( real(num)/2.0);
  else
   num := (n-m)*(n+m+1)*(n-m-1)*(n+m+2);
   if num = 0 then
    upsilon(n)(m) := 0.0;
   else
    upsilon(n)(m) := sqrt(real(num));
   end if;
  end if;
 end loop;
end loop;

  p(0)(0) := 1.0;     p(0)(1) := 0.0;   p(0)(2) := 0.0;
  p(1)(1) := sqrt3; p(1)(2) := 0.0; p(1)(3) := 0.0;

 for n in 2..Max_Degree_And_Order loop
    p(n)(n) := sqrt( real(2*n+1)/real(2*n))*p(n-1)(n-1);
```

```
    nrdiag(n) := sqrt( real(2*n+1))*p(n-1)(n-1);
    num := (2*n+1)*(2*n-1);
    alpha(n) := sqrt(real(num))/real(n);
    num := (2*n+1);
    den := (2*n-3);
    beta(n) := sqrt(real(num)/real(den))*real(n-1)/real(n);
  end loop;

end Normalized_Gravity_Model;
```

A.4 Body of General_Gravity_Gradient

```
with trigonometric_functions; use trigonometric_functions;

package body general_gravity_gradient is

 function gravity_gradient_torque(mass_tensor, dgdx : matrix_3x3;
                pitch,yaw,roll:real) return vector_3 is

   torque : vector_3;
   g, b : matrix_3x3 ;
   s1,c1,s2,c2,s3,c3,s2s3,s2c3:real;

 begin
     s1 := sin(pitch);
     c1 := cos(pitch);
     s2 := sin(yaw);
     c2 := cos(yaw);
     s3 := sin(roll);
     c3 := cos(roll);
     S2s3 := S2 * S3;
     S2c3 := S2 * C3;

   b := (( C1 * C2, -C1 * S2c3 + S1 * S3, C1 * S2s3 + S1 * C3),
       ( S2     ,  C2 * C3          ,-C2 * S3          ),
       (-S1 * C2,  S1 * S2c3 + C1 * S3,-S1 * S2s3 + C1 * C3));

   g := b**tr * dgdx * b ; --Note: **tr results in transpose

   torque(1) :=  g(2,3)*(mass_tensor(3,3) - mass_tensor(2,2))
             - g(1,3)* mass_tensor(1,2)
             + g(1,2)* mass_tensor(1,3)
             - mass_tensor(2,3)*(g(3,3) - g(2,2)) ;

   torque(2) :=  g(1,3)*(mass_tensor(1,1) - mass_tensor(3,3))
             + g(2,3)* mass_tensor(1,2)
             - g(1,2)* mass_tensor(2,3)
             - mass_tensor(1,3)*(g(1,1) - g(3,3)) ;

   torque(3) :=  g(1,2)*(mass_tensor(2,2) - mass_tensor(1,1))
             - g(2,3)* mass_tensor(1,3)
             + g(1,3)* mass_tensor(2,3)
             - mass_tensor(1,2)*(g(2,2) - g(1,1)) ;

   return torque ;

 end gravity_gradient_torque;

end general_gravity_gradient ;
```

A.5 Spec of Fast_Magnetic_Model

```
with Real_Types;
use Real_Types;
with Vector_Matrix_3;
use Vector_Matrix_3;

package fast_Magnetic_Model is

  Max_Magnetic_Model_Name_Length : constant Positive := 80;
  max_degree_and_order : constant Positive := 20;

  type Data_Coefficient_Array is
    array (Natural range <>, Natural range <>) of Real;
  type magnetic_array is array(0..max_degree_and_order+2) of real;
  type mag_ptr is access magnetic_array;
  type magnetic_array_2 is array(0..max_degree_and_order) of
                                        mag_ptr;
  type Magnetic_Model_Data is private;

  ---------------------------------------------------------------------------

  function Create_Magnetic_Model (Name_In    : String;
                          G, H     : Data_Coefficient_Array;
                          Radius : Real) return Magnetic_Model_Data;

  ---------------------------------------------------------------------
  procedure Maggot (Mmd      : in  Magnetic_Model_Data;
                X       : in  Vector_3;
                R       : in  Real;
                Nax, Max : in  Natural;
                B       : out Vector_3);
  ---------------------------------------------------------------------

private

  type Magnetic_Model_Data is -- defaulted to point mass gem_9
   record
     Name          : String (1 .. Max_magnetic_Model_Name_Length);
     Name_Length    : Integer;
     G             : magnetic_array_2;
     H             : magnetic_array_2;
     Radius        : Real := 6_371_200.0; -- planet mean radius (m)
     Model_Max_Size : Natural;-- max size current gravity model data
   end record;
  ---------------------------------------------------------------------------

end fast_Magnetic_Model;
```

A.6 Body of Fast_Magnetic_Model

```
with Extended_Range_Combinatoric_Functions;
use Extended_Range_Combinatoric_Functions;
with Exponential_Logarithm_Functions;
use Exponential_Logarithm_Functions;

package body Fast_Magnetic_Model is

 Magnetic_Model_Name_Too_Long : exception;
 bad_Magnetic_data : exception;
 twonm1,twonm1on,nm1on : Magnetic_array;

 P : Magnetic_array_2 := (others => new Magnetic_array);
 ------------------------------------------------------------------

 procedure Maggot (Mmd     : in  Magnetic_Model_Data;
                   X       : in  Vector_3;
                   R       : in  Real;
                   Nax, Max : in  Natural;
                   B       : out Vector_3) is


  Ri, Xovr, Yovr, Zovr, Ep : Real;
  Muor, aeor2,  Reor, Reorn  : Real;
  ctil, stil : Magnetic_array;
  Sumh, Sumgam, Sumj, Sumk, Sumh_N,  Lambda : Real;
  pnm,cnm,snm,ctmm1,stmm1 : real;
  Sumgam_N, Sumj_N, Sumk_N, Mxpnm, Npmp1 : Real;
  Bnmtil,n_const : Real;
  Mm1, Mm2, Mp1, Mp2, Nm1, Lim ,nm2: Integer;
  pn,pnm1,pnm2 :  mag_ptr;
  cn,sn : mag_ptr;

begin

  Ri   := 1.0 / R;
  Xovr  := X (1) * Ri;
  Yovr  := X (2) * Ri;
  Zovr  := X (3) * Ri;
  Ep   := Zovr;
  Reor  := Mmd.Radius * Ri;
  Reorn := Reor;
  aeor2 := reor*reor;

  ctil (0) := 1.0; ctil (1) := Xovr;
  stil (0) := 0.0; stil (1) := Yovr;
if Nax < 1 then
  Sumj    := 0.0;
  Sumk    := 0.0;
  Sumh    := 0.0;
  Sumgam  := 0.0;
elsif Max >  0 then
  Sumj    := reor*mmd.g(1)(1);
  Sumk    := reor*mmd.h(1)(1);
```

```
  Sumh     := reor*mmd.g(1)(0);
  Sumgam   := 3.0*(sumj*xovr + sumk*yovr) + 2.0*(sumh*zovr);
 else  -- Max = 0
  Sumj     := 0.0;
  Sumk     := 0.0;
  Sumh     := reor*mmd.g(1)(0);
  Sumgam   := 2.0*sumh*zovr; --note: ep and zovr are the same thing
 end if;

  p(1)(0) := ep;
  for N in 2 .. Nax loop
   n_const := twonm1(n);
   nm1 := n - 1;
   nm2 := n - 2;
   pn := p(n);
   pnm1 := p(nm1);
   pnm2 := p(nm2);
   Pn(nm1) := ep*Pn(n);
   Pn(0) := Twonm1on(n)*Ep*Pnm1(0) - Nm1on(n)*Pnm2(0);
   Pn(1) := Pnm2(1) + n_const * Pnm1(0);
   Reorn  := Reorn * Reor;
   cn := mmd.g(n);
   sn := mmd.h(n);
   Sumh_N   := Pn (1) * Cn(0);
   Sumgam_N := Pn (0) * Cn(0) * real(n + 1);

   if Max > 0 then
   for m in 2..nm2 loop
    Pn(m) := Pnm2(m) + n_const * Pnm1(m-1);
   end loop;

    Sumj_N := 0.0;
    Sumk_N := 0.0;
    nm1 := n - 1;

    ctil (N) := ctil (1) * ctil (Nm1) - stil (1) * stil (Nm1);
    stil (N) := stil (1) * ctil (Nm1) + ctil (1) * stil (Nm1);

    if N < Max then
     Lim := N;
    else
     Lim := Max;
    end if;
    for M in 1 .. Lim loop
     Mm1   := M - 1;
     Mp1   := M + 1;
     Npmp1 := Real (N + Mp1);
     pnm := pn(m);
     cnm := cn(m);
     snm := sn(m);
     ctmm1 := ctil(mm1);
     stmm1 := stil(mm1);

     Mxpnm   := Real (M) * Pnm;
```

```
      Bnmtil  := Cnm * ctil (M) + Snm * stil (M);
      Sumh_N  := Sumh_N + Pn(mp1) * Bnmtil;
      Sumgam_N := Sumgam_N + Npmp1 * Pnm * Bnmtil;
      Sumj_N  := Sumj_N + Mxpnm * (Cnm*ctmm1 + Snm*stmm1);
      Sumk_N  := Sumk_N - Mxpnm * (Cnm*stmm1 - Snm*ctmm1);
    end loop;
    Sumj := Sumj + Reorn * Sumj_N;
    Sumk := Sumk + Reorn * Sumk_N;
  end if;

  ---- SUMS BELOW HERE HAVE VALUES WHEN M := 0

  Sumh  := Sumh + Reorn * Sumh_N;
  Sumgam := Sumgam + Reorn * Sumgam_N;

 end loop;

 Lambda := Sumgam + Ep * Sumh;
 B (1)  := aeor2 * (Lambda * Xovr - Sumj);
 B (2)  := aeor2 * (Lambda * Yovr - Sumk);
 B (3)  := aeor2 * (Lambda * Zovr - Sumh);

end Maggot;

--------------------------------------------------------------------------
function Create_Magnetic_Model (Name_In   : String;
                        g, h      : Data_Coefficient_Array;
                        Radius : Real) return Magnetic_Model_Data is
 Gmd   : Magnetic_Model_Data;
 Coef  : Real;
 n_max : Integer := g'Last (1);
begin
 if n_max < 2 then raise bad_magnetic_data; end if;

 gmd.g := (others => new Magnetic_array);
 gmd.h := (others => new Magnetic_array);
 -- Unnormalize gravity model coefficients
 for N in g'Range(1) loop
  for M in 0 .. N loop
   if M = 0 then
    Gmd.G (N)(0) :=  G (N, 0)*1.0e-9 ;
    Gmd.H (N)(0) := 0.0;
   else
    Coef := Sqrt(2.0*Factorial_Ratio(N - M,N + M))*1.0e-9;
    Gmd.G (N)( M) := Coef * G (N, M);
    Gmd.H (N)( M) := Coef * H (N, M);
   end if;
  end loop;
 end loop;
 Gmd.Radius     := Radius;
 Gmd.Name_Length := Name_In'Length;
 if Gmd.Name_Length > Max_Magnetic_Model_Name_Length then
  raise Magnetic_Model_Name_Too_Long;
 end if;
```

```
    Gmd.Name := (others => Ascii.Nul);
    Gmd.Name (1 .. Gmd.Name_Length) := Name_In;
    Gmd.Model_Max_Size := n_max;

   return Gmd;
  end Create_Magnetic_Model;


begin  --Initialize constant values
-------------------------------------------------------------------

  p(0)(0) := 1.0; p(0)(1) := 0.0; p(0)(2) := 0.0;
  p(1)(1) := 1.0; p(1)(2) := 0.0; p(1)(3) := 0.0;
  for n in 2..Max_Degree_And_Order loop
     p(n)(n) := p(n-1)(n-1)*real(2*n-1);
     p(n)(n+1) := 0.0;
     p(n)(n+2) := 0.0;
     twonm1(n) := real(2*n - 1);
     twonm1on(n) := twonm1(n)/real(n);
     nm1on(n) := real(n - 1)/real(n);
  end loop;

end Fast_Magnetic_Model;
```

Appendix B

Numerical Data & Speed Comparisons

B.1 4 x 4 Gravity Model Test Case from Ref. [2]
Position vector = 5489150.0 , 802222.0 , 3140916.0 (meters)

Gravity Model Data - Gem-10 (See Ref [14])

μ= 398_600.47E9 $\frac{m^3}{s^2}$, r_e= 6_378_139.0 m. Degree and order = 4

**** standard gravity model (As given in Ref. [2])**** Run Time: 2.70 Seconds

Gravitational acceleration from acceleration only call
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00
Gravitational acceleration from acceleration plus dgdx
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00

Potential = 6.25359843440795E+07

Analytic dgdx
1.87779191647821E-06 4.99270741320439E-07 1.96515882331155E-06
4.99270741320439E-07 -1.46519995493325E-06 2.87214112813307E-07
1.96515882331155E-06 2.87214112813307E-07 -4.12591961544953E-07

**** fast gravity model **** Run Time: 2.22 Seconds

Gravitational acceleration from acceleration only call
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00
Gravitational acceleration from acceleration plus dgdx
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00

Potential = 6.25359843440795E+07

Analytic dgdx
1.87779191647821E-06 4.99270741320439E-07 1.96515882331155E-06
4.99270741320439E-07 -1.46519995493325E-06 2.87214112813307E-07
1.96515882331155E-06 2.87214112813307E-07 -4.12591961544953E-07

**** normalized gravity model "norm_II" **** Run Time: 2.39 Seconds

Gravitational acceleration from acceleration only call
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00
Gravitational acceleration from acceleration plus dgdx
-8.44269212018857E+00 -1.23393633785485E+00 -4.84659352346614E+00

Potential = 6.25359843440795E+07

Analytic dgdx
1.87779191647821E-06 4.99270741320439E-07 1.96515882331155E-06
4.99270741320439E-07 -1.46519995493325E-06 2.87214112813307E-07
1.96515882331155E-06 2.87214112813307E-07 -4.12591961544953E-07

Note that the answers are identical to those in Ref. [2].

B.2 5 x 5 Gravity Model Test Case from Ref. [2]
Position vector = 5489150.0 , 802222.0 , 3140916.0 (meters)

Gravity Model Data - Gem-10 (See Ref [14])

μ= 398_600.47E9 $\frac{m^3}{s^2}$, r_e= 6_378_139.0 m. Degree and order = 5

**** standard gravity model (As given in Ref. [2]) **** Run Time: 3.82 Seconds

Gravitational acceleration from acceleration only call
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00
Gravitational acceleration from acceleration plus dgdx
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00

Potential = 6.25358693982450E+07

Analytic dgdx
1.87773230503190E-06 4.99259374934480E-07 1.96507472112557E-06
4.99259374934480E-07 -1.46513564895359E-06 2.87208844531796E-07
1.96507472112557E-06 2.87208844531796E-07 -4.12596656078305E-07

**** fast gravity model **** Run Time: 3.17 Seconds

Gravitational acceleration from acceleration only call
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00
Gravitational acceleration from acceleration plus dgdx
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00

Potential = 6.25358693982450E+07

Analytic dgdx
1.87773230503190E-06 4.99259374934480E-07 1.96507472112557E-06
4.99259374934480E-07 -1.46513564895359E-06 2.87208844531796E-07
1.96507472112557E-06 2.87208844531796E-07 -4.12596656078305E-07

**** normalized gravity model "norm_II" **** Run Time: 3.38 Seconds

Gravitational acceleration from acceleration only
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00
Gravitational acceleration from acceleration plus dgdx
-8.44260633555472E+00 -1.23393243051834E+00 -4.84652486332608E+00

Potential = 6.25358693982450E+07

1.87773230503190E-06 4.99259374934480E-07 1.96507472112557E-06
4.99259374934480E-07 -1.46513564895359E-06 2.87208844531796E-07
1.96507472112557E-06 2.87208844531796E-07 -4.12596656078305E-07

Note that the answers are identical to those in Ref. [2].

B.3 Gravity Gradient Torque Test Case

A comparison of general gravity gradient torque, computed with degree=2 order=0, with Roithmyar (Ref. [5]) J2 only model is given below.

Gravity Model - Gem-10 (See Ref [14])

Position vector = 5489150.0 , 802222.0 , 3140916.0 (meters)

The rotation matrix from body to inertial is derived from

pitch = 20.0 degrees
yaw = 30.0 degrees
roll = 40.0 degrees

Mass tensor = ((477.0 , 63.0 , 0.0),
(63.0 , 770.0, 0.0),
(0.0 , 0.0 , 821.0));

gravity gradient_torque from general gravity model (Spherical 0x0)
-7.38391601519382E-05 -6.34664808264096E-04 3.51747050237606E-04

gravity gradient_torque from general gravity model (2x0)
-7.35430183066330E-05 -6.34049979845033E-04 3.52179052151285E-04

gravity gradient_torque from Roithmayr J2 model
-7.3543018306633D-05 -6.3404997984503D-04 3.5217905215128D-04

gravity gradient_torque from general gravity model (4x4)
-7.35490592401439E-05 -6.34102312449005E-04 3.52209610370063E-04

Note that the torque changes slightly as more terms are included.
The major impact is expected to be control where the effect of
forces that are longitude dependent will be included.

B.4 Magnetic Field Vector

The position vector was (meters)

X = (5489150.0 , 802222.0 , 3140916.0)

Degree and Order = 10

The resulting magnetic field vector was (Tesla)

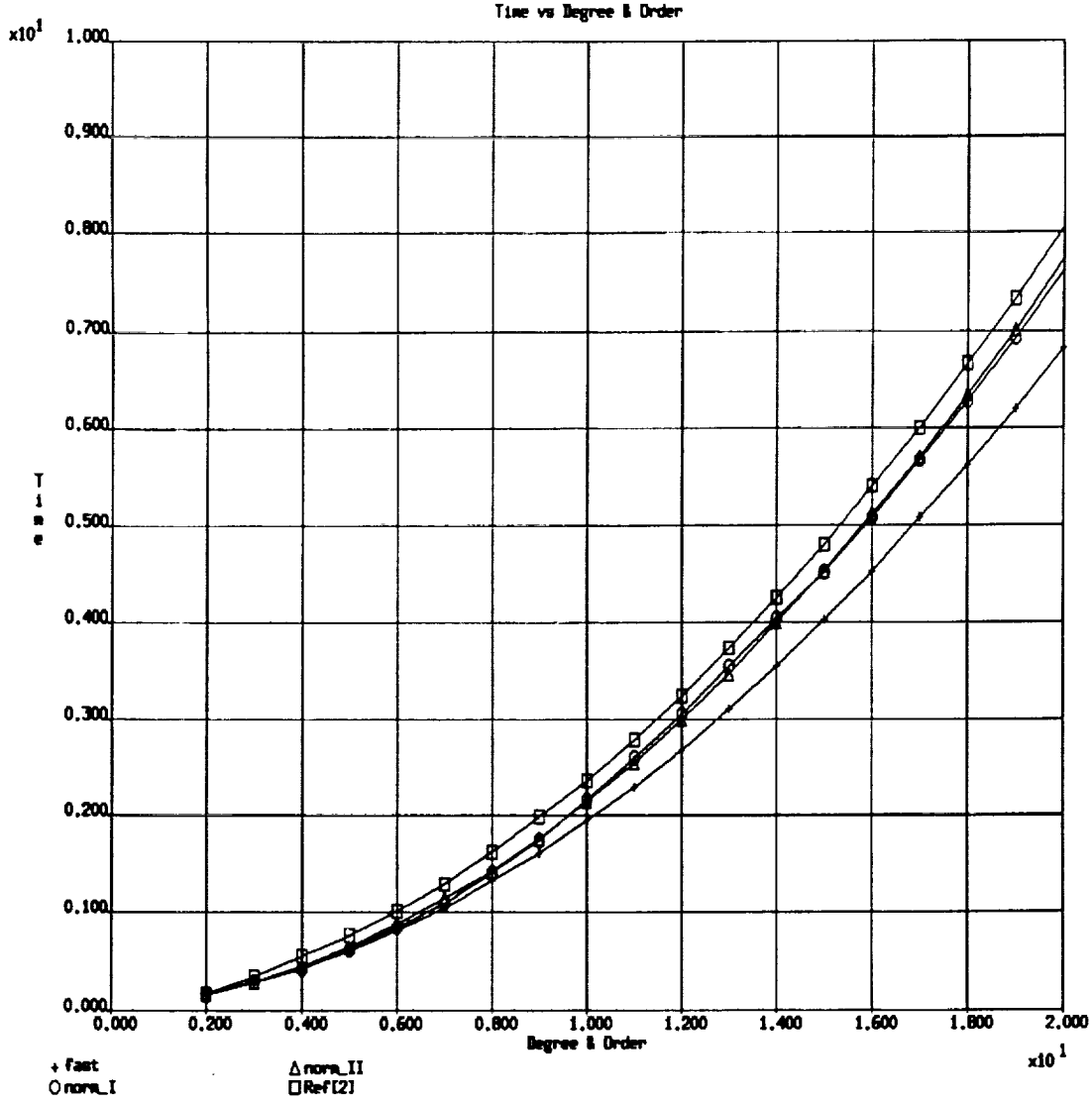
B = (-3.75259753018348E-05, -6.16002442001094E-06, 1.35121172654619E-05)

The mean radius of the earth was 6371.2 km

The (10x10) harmonic data were taken from [11] Table II, IGRF 1985

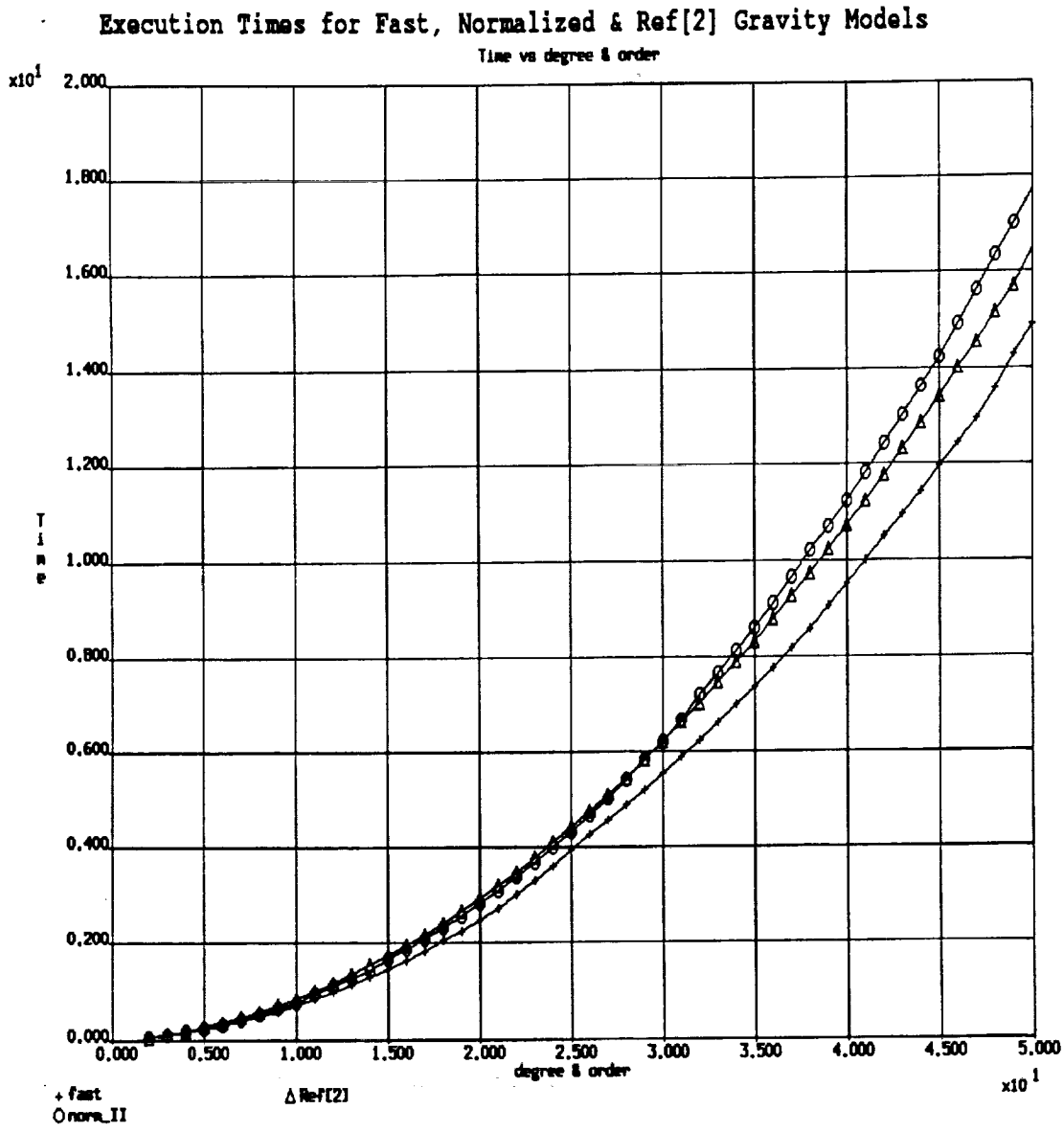
Figure B.1

Execution Times for Fast, Normalized & Ref[2] Gravity Models



This comparison show the relative efficiencies of the various implementations. Note that the normalized models, while less efficient than the unnormalized model, are none the less more efficient than the Ref[2] model. This is a result of the data structures and the precomputing of all possible derived Legendre functions. The simple normalized model (norm_I) and the normalized model (norm_II), which uses the recursion relationship from [3], have very similar speeds. In the next plot, norm_I is not shown. Note also, that the "fast" model is approximately 20% faster than the Ref[2] model.

Figure B.2



This plot, which was generated using the 50x50 GEM-T3 model [15], shows relative efficiency as degree & order increase. The difference in the run times compared to those in Fig B1 are a result of using a faster computer (Sun Sparc II instead of Sun Sparc I). Note that past degree and order 27 the normalized model is slower than the algorithm in [2]. This is a result of the extra multiplication that must be done in the normalized methods finally overcoming the benefit of precomputing some of the data. The "fast" model is always faster than [2]. It is worth noting that, at 50x50, and zero latitude, the gravitational acceleration vector computed using the simple normalized method (norm_I) differed from the other three in the last three (out of 16) places. The benefit of the recursion relationship from [3] is beginning to make itself felt. The other three agreed to 15 places. The slight kinks in the plots are the result of timer noise and not some sudden change in computation speed.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1993	Contractor Report

4. TITLE AND SUBTITLE
Fast Gravity, Gravity Partials, Normalized Gravity, Gravity Gradient Torque and Magnetic Field: Derivation, Code and Data

5. FUNDING NUMBERS
NAS9-17885

6. AUTHOR(S)
Robert G. Gottlieb

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MCDonnell Douglas Space Systems
16055 Space Center Boulevard
Houston, Texas 77062-6208

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Lyndon B. Johnson Space Center
Houston, Texas 77058

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA CR 188243

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 46

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Derivation of first and second partials of the gravitational potential is given in both normalized and unnormalized form. Two different recursion formulas are considered. Derivation of a general gravity gradient torque algorithm which uses the second partial of the gravitational potential is given. Derivation of the geomagnetic field vector is given in a form that closely mimics the gravitational algorithm. Ada code for all algorithms that precomputes all possible data is given. Test cases comparing the new algorithms with previous data are given, as well as speed comparisons showing the relative efficiencies of the new algorithms.

14. SUBJECT TERMS
Gravity, Magnetic Field, Gravitational Acceleration, Gravity Gradient Torque, Normalized Gravity

15. NUMBER OF PAGES
64

16. PRICE CODE

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

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102