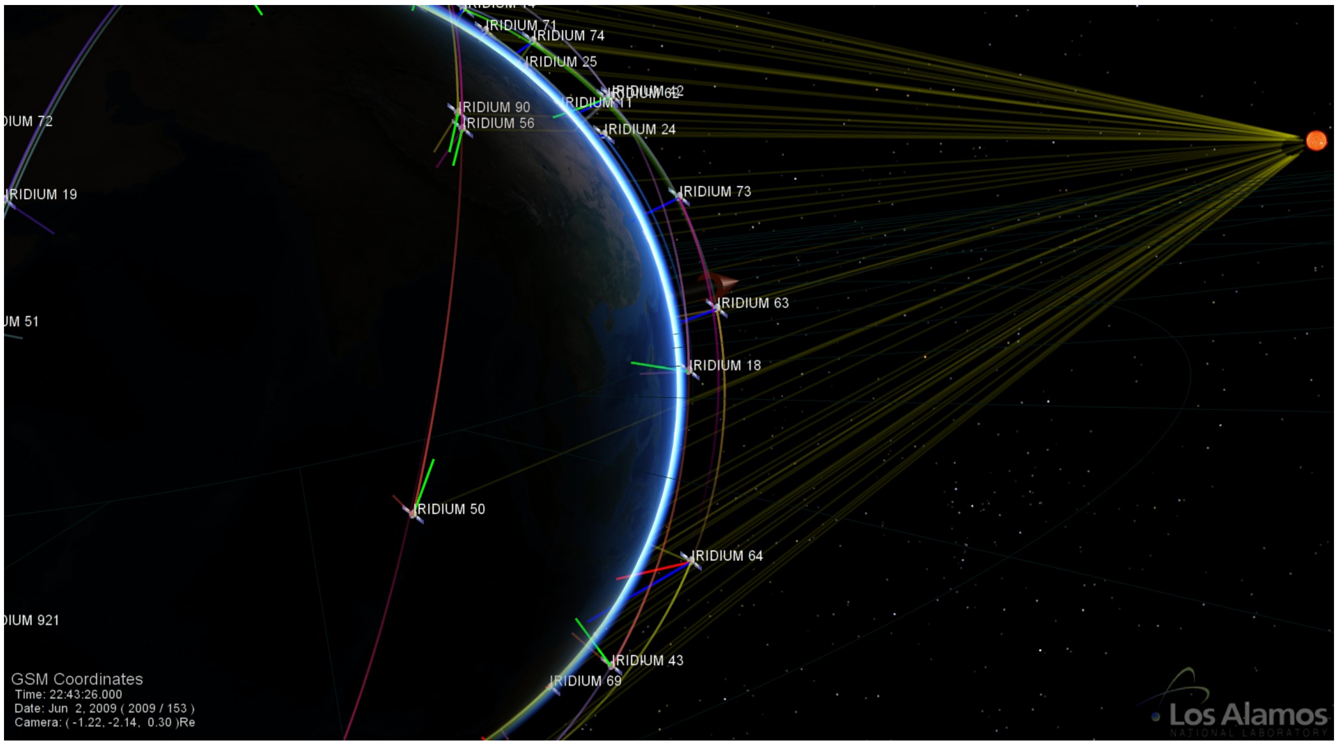
Documentation and Reference Manual for

LanlGeoMag Library
Version 1.5.6 (August 9, 2010)



Michael G. Henderson
Los Alamos National Laboratory

Version	Date	Comments
1.5.0	05/09/09	Added User Manual.
1.5.1	01/25/10	Updated DGRF/IGRF models to 11th Generation. This added IGRF 2010.0 and updated coefficients for 2005.0
1.5.2	03/01/10	Re-ran example programs for documentation. (Many of the quantities relating to IGRF change slightly due to updated IGRF models.
1.5.3	05/27/10	Significant changes to structuring of time conversions
1.5.4	06/17/10	Migrated build/install process to autotools in order to make it easier to install on different platforms. (i.e. It adds the whole GNU ./configure; make ; make install formalism).
1.5.5	07/15/10	Lots of bug fixes. Improved Lstar calculations. Tested on dipole (can get solutions accurate to at least 1e-7 in most cases). Changed some autoconf things. Header files installation now implemented as a subdir of the Makefile in the libLanlGeoMag directory (so it will install properly if you make from only the lib directory.)
1.5.6	08/09/10	Sped up IGRF model using a different recurrence relation. Added AE8/AP8 models.

Table of Contents

1 Overview

The LanlGeoMag library is roughly divided into four main sections which provide routines and utilities for;

1. Coordinate system definitions and transformations (e.g. ITRF, PEF, TOD, MOD, GEI2000, GSM, SM, GSE, CDMAG, Geodetic, etc...)
2. Magnetic field model calculations (e.g. Centered dipole, Eccentric dipole, IGRF, T87, T89, etc...)
3. Magnetic field line tracing (e.g. Trace to given altitude, trace to Min-B, trace to mirror point, etc...)
4. Adiabatic invariant calculations (e.g. I, K, L*).

2 Installation

The package uses autotools to enhance the portability and ease of installation on various platforms. On Linux and Mac installation is done as follows. Move the tarball into the directory where you want to build from.

```
% cd <build directory>
% tar xvzf lanlgeomag-1.5.5.tar.gz
% ./configure
% make
% make install
```

3 Compiling User Programs

Information about installed packages can be obtained by running the pkg-config program which should be available on most systems. For example, to determine the version of LanlGeoMag that is installed, run:

```
% pkg-config --modversion lgm
1.5.5
```

To obtain the gcc library-related arguments needed to compile a program, run:

```
% pkg-config --libs lgm
-fopenmp -L/usr/local/lib -lLanlGeoMag -lgsl -lgslcblas -lm
```

The gcc cflags options are similarly obtained:

```
% pkg-config --cflags lgm
-g -Wall -O3 -funroll-all-loops -ffast-math -fopenmp
-I/usr/local/include/Lgm
```

Using this capability, it is easy to compile programs as follows:

```
% gcc `pkg-config –cflags --libs lgm` MyProgram.c -o MyProgram
```

The back-quotes mean, replace with the results of running the contents of the string.

4 Basic Data Types

In order to start doing coordinate transformations, we will first introduce a few basic data structures that are used extensively throughout the LanlGeoMag library routines.

4.1 The Lgm_CTrans Structure and Initialization

Many of the functions and routines in the coordinate transformation parts of the LanlGeoMag library pass information through an **Lgm_CTrans** structure which must be initialized before use. The initialization procedure allocates space for the structure and it sets some basic parameters. The user must also be careful to free the structure when it is no longer needed. The following code example illustrates how to perform the initialization.

```
#include <Lgm_CTrans.h> // A required core LanlGeoMag include file
int main( ) {
      Lgm_CTrans *c;
      // Initialize Ctrans structure.
      c = Lgm_init_ctrans( 0 );

      // Do other things here ...
      // E.g. Coordinate transformations, etc..

      // free the structure
      free( c );
      return(0);
}
```

The argument to **`Lgm_init_ctrans()`** controls the verbosity level of subsequent calls to the library functions and routines. The verbosity level can also be changed any time after initialization by setting the **`Verbosity`** variable in the structure that gets returned by Lgm_init_ctrans(). For example,

```
    c->Verbosity = 1;
```

Other variables contained in the structure will be described in more detail in subsequent sections.

4.2 The Lgm_Vector Data Type

The LanlGeoMag library defines a number of new data types (the **`Lgm_CTrans`** structure is one) – the definitions of which are contained in the various include files. We will describe these in detail in subsequent sections. However, to get started with simple coordinate transformations we need to describe a simple data type that we use to represent vectors – the **`Lgm_Vector`** type. This is a small data structure that contains three variables representing the x, y, and z components of a vector. It it defined in **`LgmVec.h`** as follows,

```
typedef struct Lgm_Vector {
      double x;
      double y;
      double z;
} Lgm_Vector;
```

The **`LgmVec.h`** file is included when you include the **`Lgm_CTrans.h`** include file. Declaring vectors or pointers to vectors is then simply done as follows;

```
Lgm_Vector  Ugsm, Usm;  // Vectors
Lgm_Vector  *u, *v;     // Pointers to vectors
```

5 Coordinate Transformations – A Quick Start

The quickest way to start doing coordinate transformations is to show a simple example. In the following example code, we will first define a date, a time (in UTC) and a position in GSM coordinates. Then we will transform the vector into SM coordinates. Note that in this example we use units of Earth-radii, or Re.

```
#include <Lgm_CTrans.h>
int main( ) {
    Lgm_CTrans  *c = Lgm_init_ctrans( 0 ); // more compact declaration
    Lgm_Vector   Ugsm, Usm;
    long int     Date;
    double       UTC;

    Date = 20040812;    // August 12, 2004
    UTC  = 12.34567;    // Universal Time Coordinated (in decimal hours)
    Ugsm.x = -6.6; Ugsm.y = 3.4; Ugsm.z = -2.3; // Set a vector in GSM coordinates

    // Set up all the necessary variables to do transformations for this
    // Date and UTC
    Lgm_Set_Coord_Transforms( Date, UTC, c );

    // Do the transformation from GSM->SM
    Lgm_Convert_Coords( &Ugsm, &Usm, GSM_TO_SM, c );

    // Print out the final results
    printf("Date = %8ld\n", Date);
    printf("UTC  = %lf\n", UTC);
    printf("Ugsm = %.8lf %.8lf %.8lf Re\n", Ugsm.x, Ugsm.y, Ugsm.z);
    printf("Usm  = %.8lf %.8lf %.8lf Re\n", Usm.x, Usm.y, Usm.z);

    free( c ); // free the structure
    return(0);
}
```

Save this as **`Quickstart.c`**, then compile and run as follows,

```
% gcc QuickStart.c -lLanlGeoMag -lgsl -lgfortran -o QuickStart
% ./QuickStart

Date = 20040812
UTC  = 12.345670
Ugsm = -6.60000000 3.40000000 -2.30000000 Re
Usm  = -5.53490381 3.40000000 -4.26788471 Re
```

Note that none of the calls to the library routines produced any output (the only information printed was from **`main()`** in the **`QuickStart.c`** program). That is because the verbosity level was set to 0. If we had set it to 1 (e.g. by changing the argument to **`Lgm_init_ctrans()`** **)** the output would have looked like this;

```
Time Quantitites:
    fYear              = 2004.616160
    Date               = 20040812
    UTC                = 12.34567000000000 (  12$^h$ 20$^m$ 44$^s$.41200000 )
    UT1                = 12.34567000000000 (  12$^h$ 20$^m$ 44$^s$.41200000 )
    TAI                = 12.35455888888889 (  12$^h$ 21$^m$ 16$^s$.41200000 )
    TT = TAI+32.184s   = 12.36349888888889 (  12$^h$ 21$^m$ 48$^s$.59600000 )
    TDB                = 12.36349887715793 (  12$^h$ 21$^m$ 48$^s$.59595777 )
    DUT1 = UT1-UTC     = 0.0000000 seconds
    DAT  = TAI-UTC     = 32.0000000 seconds
    JD_UTC             = 2453230.014402917 days
    JD_UT1             = 2453230.014402917 days
    JD_TT              = 2453230.015145787 days
    T_UT1              =    0.0461331801 Julian Centuries of UT1
    T_TT               =    0.0461332004 Julian Centuries of TT
    year               = 2004
    month              = 8
    day                = 12
    doy                = 225
    dow                = 4
    dowstr             = Thu
    gmst (hours)       = 9.7650452 (  09$^h$ 45$^m$ 54$^s$.163 )
    gmst (degrees)    = 146.4756776 (  146° 28′ 32″.439 )
    gast (hours)      = 9.7648988 (  09ʰ 45ᵐ 53ˢ.636 )
    gast (degrees)    = 146.4734825 (  146° 28′ 24″.537 )

Eccentricity and Obliquity:
    eccentricity                    = 0.01670717
    epsilon mean (obliq. of ecliptic) = 23.43869119 (  23° 26′ 19″.288 )
    epsilon true (obliq. of ecliptic) = 23.44075681 (  23° 26′ 26″.725 )

Precession Quantities:
    Zeta              = 0.0295538 (  00° 01′ 46″.394 )
    Zee               = 0.0295543 (  00° 01′ 46″.396 )
    Theta             = 0.0256845 (  00° 01′ 32″.464 )

Nutation Quantities:
    dPsi (w.o. corrections)         = -0.00239301 ( -00° 00′ 08″.615 )
    dEps (w.o. corrections)         = 0.00206562 (  00° 00′ 07″.436 )
    ddPsi (EOP correction)          = 0.00000000 (  00° 00′ 00″.000 )
    ddEps (EOP correction)          = 0.00000000 (  00° 00′ 00″.000 )
    dPsi (w. corrections)           = -0.00239301 ( -00° 00′ 08″.615 )
    dEps (w. corrections)           = 0.00206562 (  00° 00′ 07″.436 )
    epsilon true (obliq. of ecliptic) = 23.44075681 (  23° 26′ 26″.725 )
    Equation of the Equinox         = -0.00219510 ( -00° 00′ 07″.902 )

Low Accuracy Position of Sun:
    lambda_sun      =      140.123365  (  140° 07′ 24″.113 )
    earth_sun_dist  =    23763.937342 Re
    beta_sun        =             0   (  00° 00′ 00″.000 )
    RA_sun  (MOD)   =      142.529761  (  09ʰ 30ᵐ 07ˢ.143 )
    DEC_sun (MOD)   =       14.774971  (  14° 46′ 29″.896 )
    RA_sun  (TOD)   =      142.527845  (  09ʰ 30ᵐ 06ˢ.683 )
    DEC_sun (TOD)   =       14.776983  (  14° 46′ 37″.139 )
    RA_sun  (J2000) =      142.466525  (  09ʰ 29ᵐ 51ˢ.966 )
    DEC_sun (J2000) =       14.795347  (  14° 47′ 43″.251 )

High Accuracy Position of Sun:
    lambda_sun_ha   =      140.124771  (  140° 07′ 29″.175 )
    r_sun_ha        =    23765.146671 Re
    beta_sun_ha     =    0.000142757   (  00° 00′ 00″.514 )
    RA_sun  (MOD)   =      142.531187  (  09ʰ 30ᵐ 07ˢ.485 )
    DEC_sun (MOD)   =       14.774663  (  14° 46′ 28″.785 )
    RA_sun  (TOD)   =      142.529271  (  09ʰ 30ᵐ 07ˢ.025 )
    DEC_sun (TOD)   =       14.776675  (  14° 46′ 36″.029 )
    RA_sun  (J2000) =      142.467952  (  09ʰ 29ᵐ 52ˢ.309 )
    DEC_sun (J2000) =       14.795039  (  14° 47′ 42″.142 )

Sun vector and Ecliptic Pole in GEI2000:
    Sun               = (-0.767427, 0.588234, 0.255023)
    EcPole            = (0.000000, -0.397768, 0.917486)

Geo-dipole tilt angle:
    psi                    = 18.417974  (  18° 25′ 04″.708 )
    sin_psi                = 0.315947
    cos_psi                = 0.948777
    tan_psi                = 0.333004

Position of Moon:
   RA_moon                     = 101.551918  (  06ʰ 46ᵐ 12ˢ.460 )
   DEC_moon                    = 27.668707  (  27° 40′ 07″.345 )
   EarthMoonDistance           = 63.520553
   MoonPhase                   = 0.115766

IGRF-derived quantities:
    M_cd              = 30040.633779
    M_cd_McIllwain    = 31165.300000
    CD_gcolat         = 10.267478 (deg.)  (  10° 16′ 02″.919 )
    CD_glon           = -71.786745 (deg.)  ( -71° 47′ 12″.284 )
    ED_x0             = -0.062921  Re  (-401.319329 km)
    ED_y0              = 0.049572  Re  (316.174073 km)
    ED_z0              = 0.032618  Re  (208.039137 km)

Transformation Matrices:
                         [     -0.76742666       0.58823413       0.25502339 ]
    Amod_to_gse        = [     -0.64113674      -0.70410339      -0.30525742 ]
                         [      0.00000000      -0.39776755       0.91748623 ]

                         [     -0.76742666       0.58823413       0.25502339 ]
    Amod_to_gsm        = [     -0.56385377      -0.80855732       0.16823787 ]
                         [      0.30516429      -0.01468567       0.95218648 ]

                         [     -0.83308118       0.55315065       0.00033982 ]
    Agei_to_wgs84      = [     -0.55315058      -0.83308125       0.00026865 ]
                         [      0.00043170       0.00003584       0.99999991 ]

                         [     -0.76742666      -0.64113674       0.00000000 ]
    Agse_to_mod        = [      0.58823413      -0.70410339      -0.39776755 ]
                         [      0.25502339      -0.30525742       0.91748623 ]

                         [      1.00000000      -0.00000000       0.00000000 ]
    Agse_to_gsm        = [     -0.00000000       0.87945946       0.47597380 ]
                         [      0.00000000      -0.47597380       0.87945946 ]

                         [     -0.83308118      -0.55315058       0.00043170 ]
    Awgs84_to_gei      = [      0.55315065      -0.83308125       0.00003584 ]
                         [      0.00033982       0.00026865       0.99999991 ]

                         [     -0.76742666      -0.56385377       0.30516429 ]
    Agsm_to_mod        = [      0.58823413      -0.80855732      -0.01468567 ]
                         [      0.25502339       0.16823787       0.95218648 ]

                         [      0.94877694       0.00000000      -0.31594669 ]
    Agsm_to_sm         = [      0.00000000       1.00000000       0.00000000 ]
                         [      0.31594669       0.00000000       0.94877694 ]

                         [      1.00000000      -0.00000000       0.00000000 ]
    Agsm_to_gse        = [     -0.00000000       0.87945946      -0.47597380 ]
                         [      0.00000000       0.47597380       0.87945946 ]

                         [      0.94877694       0.00000000       0.31594669 ]
    Asm_to_gsm         = [      0.00000000       1.00000000       0.00000000 ]
                         [     -0.31594669       0.00000000       0.94877694 ]

                         [      0.99999937      -0.00103163      -0.00044828 ]
    Agei_to_mod        = [      0.00103163       0.99999947      -0.00000023 ]
                         [      0.00044828      -0.00000023       0.99999990 ]

                         [      0.99999937       0.00103163       0.00044828 ]
    Amod_to_gei        = [     -0.00103163       0.99999947      -0.00000023 ]
                         [     -0.00044828      -0.00000023       0.99999990 ]

                         [      1.00000000       0.00003832       0.00001661 ]
    Amod_to_tod        = [     -0.00003832       1.00000000      -0.00003605 ]
                         [     -0.00001661      -0.00000023       1.00000000 ]

                         [      1.00000000      -0.00003832      -0.00001661 ]
    Atod_to_mod        = [      0.00003832       1.00000000       0.00003605 ]
                         [      0.00001661      -0.00003605       1.00000000 ]

                         [     -0.83363029       0.55232286       0.00000000 ]
    Atod_to_pef        = [     -0.55232286      -0.83363029       0.00000000 ]
                         [      0.00000000       0.00000000       1.00000000 ]

                         [     -0.83363029      -0.55232286       0.00000000 ]
    Apef_to_tod        = [      0.55232286      -0.83363029       0.00000000 ]
                         [      0.00000000       0.00000000       1.00000000 ]
```

```
                    [ -8.33651446e-01   5.52290925e-01   0.00000000e+00 ]
    Ateme_to_pef      = [ -5.52290925e-01  -8.33651446e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [ -8.33651446e-01  -5.52290925e-01   0.00000000e+00 ]
    Apef_to_teme      = [  5.52290925e-01  -8.33651446e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [  1.00000000e+00   0.00000000e+00  -0.00000000e+00 ]
    Awgs84_to_pef     = [  0.00000000e+00   1.00000000e+00   0.00000000e+00 ]
                    [  0.00000000e+00  -0.00000000e+00   1.00000000e+00 ]

                    [  1.00000000e+00   0.00000000e+00   0.00000000e+00 ]
    Apef_to_wgs84     = [  0.00000000e+00   1.00000000e+00  -0.00000000e+00 ]
                    [ -0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

Date = 20040812
UTC  = 12.345670
Ugsm = -6.60000000 3.40000000 -2.30000000 Re
Usm  = -5.53525042 3.40000000 -4.26743515 Re
```

This ends up printing out a large portion of the computed variables in the **`Lgm_Ctrans`** structure (although not all of them) which is obviously much more verbosity than we would want to have during normal calculations, but it is very useful for debugging. Note that some of the angles printed out have embedded Unicode (UTF-8) symbols to represent degrees, arc-minutes, arc-seconds while some of the times have superscripted h, m, s symbols to represent hours minutes seconds. The LanlGeoMag library has routines to print times and angles out in this enhanced UTF-8 format and we will describe these later.

Some other thing to note from the verbose printout is that a reasonably large number of transformation matrices are computed. These matrices allow us to transform between any two of the defined coordinate systems in any direction. Conversions between the Terrestrial system and Celestial inertial system used in this version of LanlGeoMag are based on the IAU-1976 theory of precession and the IAU-1980 theory of nutation for coordinate reductions. To get to the J2000 epoch Geocentric Equatorial Inertial system starting from an earth centered, Earth fixed system (sometimes the acronym ECEF is used for such a system) in which the rotation axis is fixed in the crust of the Earth, a number of standard transformations must be applied. The earth fixed system we use here is the WGS84 system (World Geodetic System 1984) and is the one used by most GPS equipment (we also refer to this system simply as GEO coordinates). From WGS84 coordinates, we transform to Pseudo-Earth-Fixed (PEF) coordinates (a system in which the pole is not exactly fixed to the crust of the Earth – but its close). From PEF we transform to True-Of-Date (or TOD) coordinates. Then from TOD we go to Mean-Of-Date (or MOD) coordinates. Then finally we can transform from MOD to GEI2000 coordinates. Note that these are all standard coordinate transformations that are generally referred to as IAU-76/FK-5 reduction theory (e.g. See *Vallado [2007]*). Some of the coordinate transformations require additional data that must be updated on a regular basis. These additional data are referred to as Earth Orientation Parameters (EOP). For high accuracy results, the transformation between WGS84 and PEF requires the polar motion EOP parameters, xp and yp and the transformations between TOD and MOD require additional corrections to the nutation parameters (the so-called ddPsi and ddEps parameters). These are all set to zero by default, but can be set either manually or by calls to other LanlGeoMag library routines. Additional coordinate systems are also defined including GSM, SM, CD, ED, GSE. We will describe these coordinate systems in more detail later, but for now we note that we can transform from any one to any other using the **`Lgm_Convert_Coords()`** routine. Transformation from one system to another is specified by using pre-defined symbols such as **`GSM_TO_SM`** or **`WGS84_TO_MOD`** (these symbols are all defined in the **`Lgm_CTrans.h`** include file. An additional non-standard coordinate system called TEME (True Equator, Mean Equinox) is also defined. It is commonly thought (e.g. See *Vallado [2007]*) that this coordinate system is the one in which results of the SGP4 orbit propagator (which predicts satellite positions based on Two-Line-Elements) are output (the coordinate system arises implicitly as a result of certain assumptions made in SGP4 and the TLE determinations). Transformations to and from TEME are also provided as well as an implementation of the SGP4 code itself and a TLE reader.

6 Time Formats and Conversions

In the example above, we saw how to transform a vector from GSM to SM at a given moment in time that we defined in terms of a Date and a UTC. The date was in the form of **`YYYYMMDD`** (where **`YYYY`** is a 4 digit year, **`MM`** is a two digit month (January is 01) and **`DD`** is a 2-digit day of Month). The time of day is given in terms of decimal hours of UTC (Universal Time Coordinated). UTC is the most common form of time keeping and is the basis for most legal trade (its essentially our standard clock time – although for regions around the world local time is computed by add/subtracting integral numbers of hours or half hours to UTC.)

Other forms of date can be used as well. In most places in the LanlGeoMag library, the date can be either **`YYYYMMDD`** or **`YYYYDDD`** (where **`DDD`** is a 3-digit day of year). Routines like **`Lgm_Set_Coord_Transforms()`** will decide autonomously which type of date it gets. In addition to these date formats, LanlGeoMag also uses Julian Date (JD), Modified Julian Date (MJD), and a number of variations of “Julian Centuries” since some epoch or another.

Other forms of time used include UTC (Universal Time Coordinated), UT1, TAI (international Atomic Time), and TT (Terrestrial Time – also referred to as Terrestrial Dynamical Time (TDT) or Ephemeris Time (ET)). A description of time values is given below:

UT0	Uncorrected Universal Time based on observed rotation of the Earth. Not generally used in this form.
UT1	A corrected version of UT0. This is the most common form of Universal Time.
UTC	Coordinated Universal Time. UTC is an approximation to UT1. UTC is kept to within 0.9s of UT1 by adding or subtracting leap seconds when the difference gets too large. UTC was introduced in 1972.
TAI	International Atomic Time (Temps Atomique International) is an average of times determined by a large number of atomic clocks around the world. TAI-UT1 was approximately 0 on 1958 Jan 1. From 1972 onward, TAI differs from UTC by an integral number of seconds (i.e.

leap seconds).

ET	Ephemeris Time. Theoretical calculations of orbits, etc. require a uniform timescale. ET was introduced for such pruposes. It was replaced by TDT in 1984.
TDT	Terrestial Dynamical Time. Used from 1984-2000. Replaced ET in 1984 and was replaced by TT in 2001.
TT	Terrestial Time. Replaced TDT in 2001. We can regard ET, TDT, and TT to be continuous time scales for our purposes. TT = TAI + 32.184s.
TDB	Barycentric Dynamical Time. A dynamical time system based on observed motions of bodies relative to the solar system barycenter.
DUT1	ΔUT1. Difference between UTC and UT1 (DUT1 = UT1-UTC). Observed (and predicted) values of DUT1 are part of the Earth Orientation Parameters (EOP). These values are stored in an auxiliary file called **`LgmEop.dat`** covering the time from Jan 1 1962 to approximately 1 year into the future.
DAT	ΔAT. Difference between TAI and UTC (DAT = TAI-UTC). Integral number of leap seconds. (Although its not integral prior to introduction of leap seconds on Jan 1 1972.) These values are stored in an auxiliary file called **`LgmEop.dat`** covering the time from Jan 1 1962 to approximately 1 year into the future.

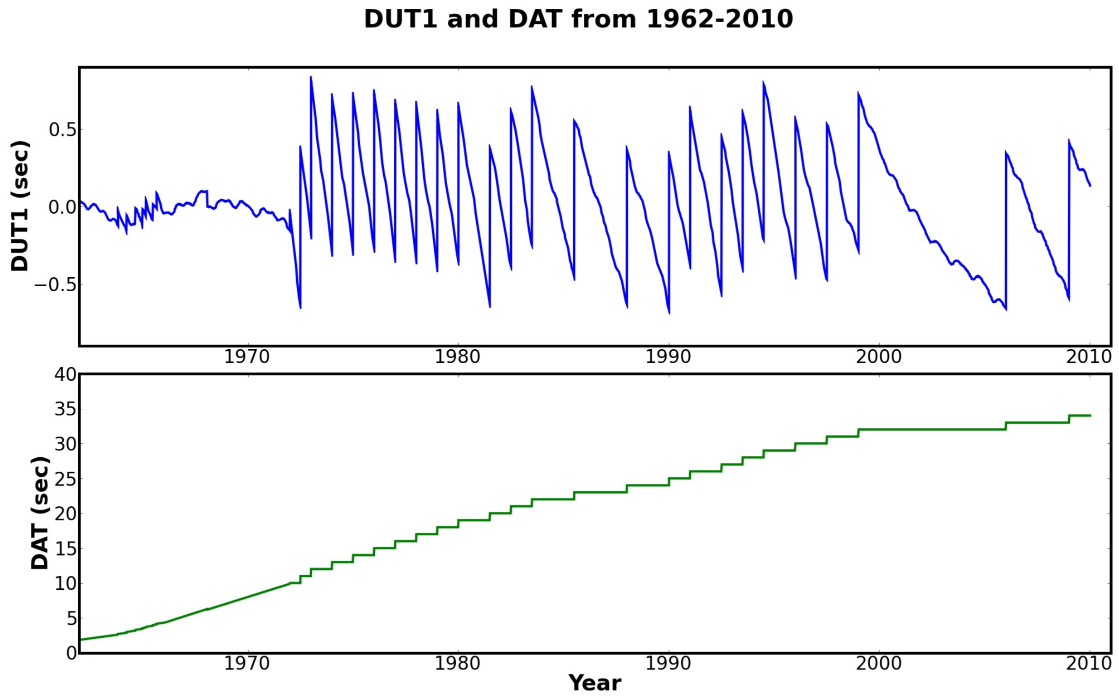


Figure 1: DUT1 and DAT from 1962 to 2010. DUT1 is UT1-UTC and DAT is TAI-UTC. The sudden jumps in DUT1 and DAT after 1972, are due to the introduction of leap seconds. A leap second is introduced whenever DUT1 deviates too far from zero. By convention, they are added (or subtracted if the need ever arises) so that the magnitude of DUT1 is never more than 0.9 seconds.

In the LanlGeoMag library we use, UTC, UT1, TAI, TT, DUT1, and DAT, and these can be seen in the verbose printout from the demo code shown in section 4. Note however that DUT1 is set to zero in that example which makes UT1 the same as UTC. DUT1 needs to be set from Earth Orientation Parameter (EOP) datasets or predictions (in the form of tables or prediction series). If the EOP corrections are not set, then DUT1 (as well as other EOP corrections like xp, yp, ddEps, ddPsi) defaults to zero. DAT, on the other hand is set even if EOP corrections are neglected. Figure 1 shows DUT1 and DAT from 1962 to 2010 and Figure 2 shows how the various timescales are related by plotting the time minus TAI.

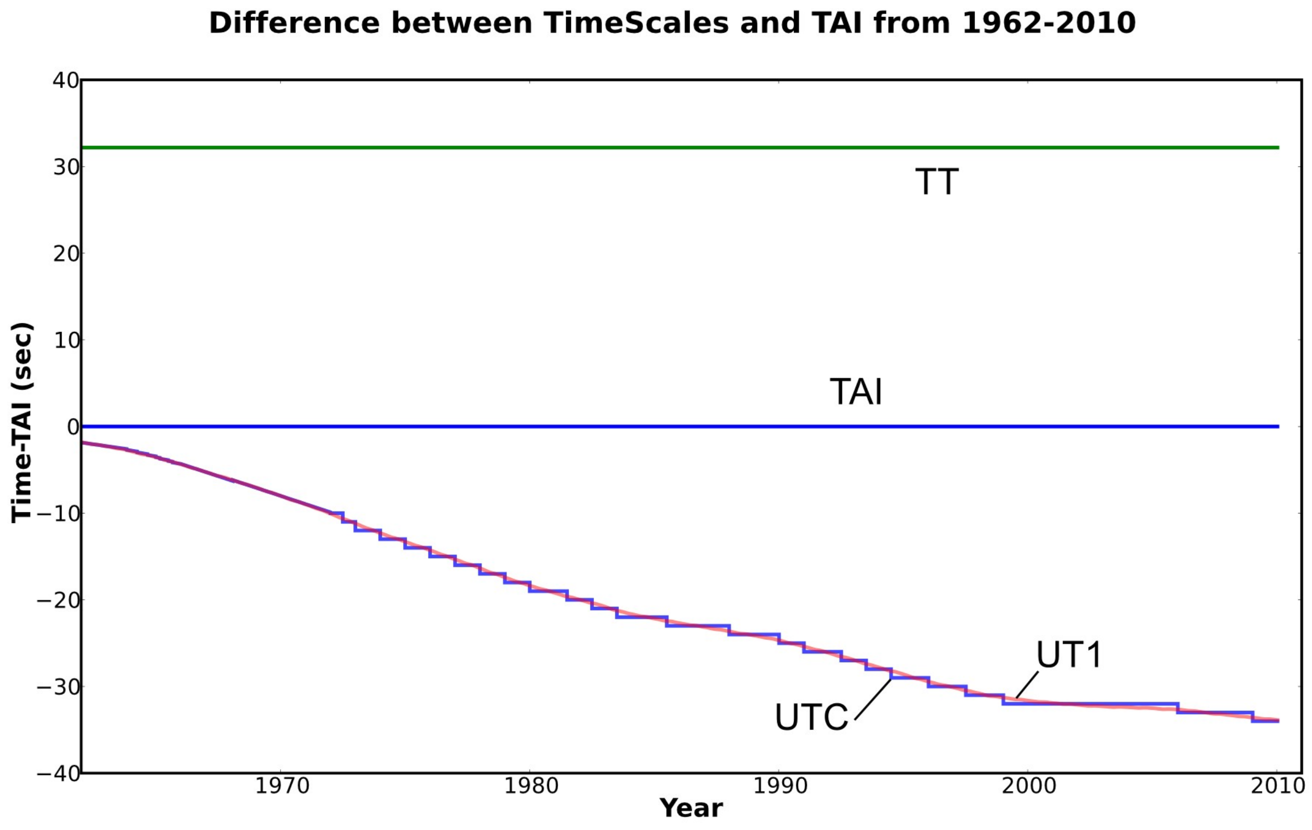


Figure 2: Difference between various timescales and TAI (Time-TAI). UT1 is not a uniform timescale since it is based on the non-uniform rotation period of the Earth. UTC is a step-wise uniform approximation to UT1. The steps in the UTC-TAI curve are due to leap seconds. Note that UTC-TAI is just -DAT and UT1-UTC is DUT1-DAT.

As can be seen in the printout from the demo code shown in section 4, the different times will get set for you when you call `Lgm_Convert_Coords()`. There are, however routines accessible to do the conversions in different ways.

6.1 Leap Second Routines

`int Lgm_LoadLeapSeconds( Lgm_LeapSeconds  *l )`

This routine loads data from the `Lgm_LeapSecondDates.dat` file. This file contains the date and Julian Date that each leap second was added since 1972 as well as the running total of leap seconds that resulted from the addition. Also defines values before 1972 (these are non-integral values that are designed to make the different time systems consistent with each other in the past).

`int Lgm_IsLeapSecondDay( long int Date, Lgm_LeapSeconds *l )`

Given a date (e.g. 20050315), this routine returns 1 if a leap second was added on that date and a zero otherwise. This is required to compute Julian Date accurately on leap second days.

`double Lgm_GetLeapSeconds( double JulianDate, Lgm_LeapSeconds *l )`

Returns number of leap seconds defined for the given Julian Date. Return value is a double. A double is returned rather than an integer because prior to 1972, DAT is non-integral.

6.2 Lgm_DateTime Structure

The Lgm_DateTime structure holds all of the bits and pieces that make up a date and time in a particular time system. The structure is defined as follows;

```
typedef struct Lgm_DateTime {

    long int    Date;       // In basic ISO format (YYYYMMDD or YYYYDDD)
                            // Represented as a single long int

    int         Year;       // 4-digit year

    int         Month;      // [1-12]

    int         Day;        // Day Of Month [1-31]

    int         Doy;        // Day Of Year [1-31]

    double      Time;       // Decimal value of time in hours

    int         Hour;       // Hours [0-23]

    int         Minute;     // Minutes [0-59]

    double      Second;     // Seconds [0-60] (the 60 accommodates leap seconds)

    int         Week;       // ISO Week number [1-53]

    int         Dow;        // ISO Day Of Week number [1-7]

    char        DowStr[10]; // ISO Day Of Week number [1-7]

    double      fYear;      // Decimal year (e.g. 2004.2345)

    double      JD;         // Julian Date

    double      T;          // Julian Centuries since J2000
                            // for this time system

    double      DaySeconds; // Number of seconds in the day.

    int         TimeSystem; // e.g. LGM_UTC, LGM_UT1, LGM_TAI, LGM_GPS, LGM_TT, LGM_TDB,
LGM_TCG, etc..

} Lgm_DateTime;
```

`Lgm_DateTime *Lgm_DateTime_Create( int Year, int Month, int Day, double Time, int TimeSystem, Lgm_CTrans *c )`

Allocates and returns an Lgm_DateTime structure based on the given Year, Month, Day, Time and TimeSystem information.

```
int Lgm_Make_UTC( long int Date, double Time, Lgm_DateTime *UTC, Lgm_CTrans *c )
```

Fills an existing Lgm_DateTime structure based on the given Year, Month, Day, Time and TimeSystem information.

6.3 Time Conversion Routines

```
void Lgm_UTC_to_TAI( Lgm_DateTime *UTC, Lgm_DateTime *TAI, Lgm_CTrans *c )
void Lgm_TAI_to_UTC( Lgm_DateTime *TAI, Lgm_DateTime *UTC, Lgm_CTrans *c )
```

Converts between UTC and TAI.

```
void Lgm_UTC_to_TT( Lgm_DateTime *UTC, Lgm_DateTime *TT, Lgm_CTrans *c )
void Lgm_TT_to_UTC( Lgm_DateTime *TT, Lgm_DateTime *UTC, Lgm_CTrans *c )
```

Converts between UTC and TT.

```
void Lgm_UTC_to_TDB( Lgm_DateTime *UTC, Lgm_DateTime *TDB, Lgm_CTrans *c )
void Lgm_TDB_to_UTC( Lgm_DateTime *TDB, Lgm_DateTime *UTC, Lgm_CTrans *c )
```

Converts between TDB and UTC.

```
void Lgm_TT_to_TDB( Lgm_DateTime *TT, Lgm_DateTime *TDB, Lgm_CTrans *c )
void Lgm_TDB_to_TT( Lgm_DateTime *TDB, Lgm_DateTime *TT, Lgm_CTrans *c )
```

Converts between TDB and TT.

```
void Lgm_TT_to_TAI( Lgm_DateTime *TT, Lgm_DateTime *TAI, Lgm_CTrans *c )
void Lgm_TAI_to_TT( Lgm_DateTime *TAI, Lgm_DateTime *TT, Lgm_CTrans *c )
```

Converts between UTC and TAI.

```
double  Lgm_TAI_to_TaiSeconds( Lgm_DateTime *TAI )
void Lgm_TaiSeconds_to_TAI( double TaiSeconds, Lgm_DateTime *TAI )
```

Converts a TAI structure into a so-called TaiSeconds time and backwards. TaiSeconds is the number of SI seconds since 0h January 1, 1958 UTC.

```
void Lgm_TaiSeconds_to_UTC( double TaiSeconds, Lgm_DateTime *UTC, Lgm_CTrans *c )
double Lgm_UTC_to_TaiSeconds( Lgm_DateTime *UTC, Lgm_CTrans *c )
```

Converts a TAI structure into a so-called TaiSeconds time and backwards. TaiSeconds is the number of SI seconds since 0h January 1, 1958 UTC.

double Lgm_GPS_to_GpsSeconds(Lgm_DateTime *GPS)
void Lgm_GpsSeconds_to_GPS(double GpsSeconds, Lgm_DateTime *GPS)

Converts a GPS structure into a so-called GpsSeconds time and backwards. GpsSeconds is the number of SI seconds since 0h January 6, 1980 UTC.

void Lgm_GpsSeconds_to_UTC(double GpsSeconds, Lgm_DateTime *UTC, Lgm_CTrans *c)
double Lgm_UTC_to_TaiSeconds(Lgm_DateTime *UTC, Lgm_CTrans *c)

Converts a UTC structure into a so-called GpsSeconds time and backwards. GpsSeconds is the number of SI seconds since 0h January 6, 1980 UTC.

void Lgm_TdbSeconds_to_UTC(double TdbSeconds, Lgm_DateTime *UTC, Lgm_CTrans *c)
double Lgm_UTC_to_TdbSeconds(Lgm_DateTime *UTC, Lgm_CTrans *c)

Converts a UTC structure into a so-called TdbSeconds time and backwards. TdbSeconds is the number of SI seconds since J2000 epoch (12:00:00 TT on Jan 1 2000). This should be the same as “ephemeris time” in the SPICE package.

In order to convert between UT1 and UTC, the EOP DUT1 value must be set. If DUT1 is known, the conversion is simply UT1 = UTC+DUT1.

Other time/date routines and utilities included in LanlGeoMag library include (there are more undocumented ones also);

int Lgm_LeapYear(int Year)
Returns true (1) or false (0) if the 4-digit year is a leap year.

double Lgm_JD(int Year, int Month, int Day, double UTC, Lgm_CTrans *c)
Computes Julian Date from 4-digit Year, 2-digit Month, 2-digit Day of Month, and UTC in decimal hours. An initialized **Lgm_CTrans** structure is given because we need to know whether the date is a leap second day.

long int Lgm_JD_to_Date(double jd, int *ny, int *nm, int *nd, double *UTC)
Breaks down a Julian Date into Year, Month, Day of Month, UTC values. Also returns the Date in YYYYMMDD format. Currently does not deal correctly with leap second dates.

double Lgm_hour24(double);
Puts a decimal hour into the range 0-24.

int Lgm_DayofYear(int Year, int Month, int Day);
Computes the day of the year given the Year, Month and Day of Month.

int Lgm_DayofWeek(int Year, int Month, int Day, char *Str);
Computes the day of the week given the Year, Month and Day of Month.

`int Lgm_IsValidDate( long int Dsate );`

Tests to see if the given date is a valid one (e.g. 19980231 is not valid).

`int Lgm_Doy( long int Date, int *Year, int *Month, int *Day, int *DOY );`

Breaks down a Date into Year, Month, Day of Month, Day of Year values.

```
void Lgm_jd_to_ymdh( double JD, long int *Date,
                  int *year, int *month, int *day, double *UTC );
```

Converts Julian date back to Date, Year, Month, Day of Month, UTC values.

`double Lgm_GetCurrentJD( Lgm_CTrans *c );`

Gets the current Julian date (based on current time of the computer).

```
void Lgm_UT_to_hmsms( double UT, int *HH, int *MM, int *SS, int *MilliSec );
void Lgm_UT_to_HMS( double UT, int *HH, int *MM, int *SS );
void Lgm_UT_to_HMSd( double UT, int *sgn, int *HH, int *MM, double *SS );
```

Converts decimal hours to hour, minute, seconds, and/or millisecond format. The first two forms return integers for all values. The last one returns the full precision decimals seconds.

```
void Lgm_Print_HMS( double d );
void Lgm_Print_HMSd( double d );
```

Prints decimal hours in HMS format with UTF-8 symbols for hms. The first form prints no decimals in the seconds field. The second one does. We should unify these so the number of values after the decimal can be user-defined. Also, we need versions that fill a string passed in by the user. (Then we would need to be careful to use glib utf8 string routines on the resulting strings.)

`void Lgm_Print_HMSdp( double d, int UnicodeHMS, int p )`

A more generalized version of **`Lgm_Print_HMSd()`**. If **`UnicodeHMS`** is **`TRUE`**, prints decimal hours in HMS format with UTF-8 symbols for hms; otherwise uses colons as separators. The number of decimal places to print on the seconds field is given in last argument.

7 IAU-1976/FK-5 Reduction Theory

The IAU-1976/FK-5 reduction theory is a well documented, standardized and reproducible methodology for transforming between Earth-fixed coordinate systems and celestial inertial systems. Vallado's 2007 text on astrodynamics and Seidelmann's 2005 “Explanatory Supplement to the Astronomical Almanac” are excellent references on this topic and in the latest version of the code, we follow their conventions fairly closely. We briefly describe the relevant frames below and show how they are related to each other in Figure 1.

ITRF	International Terrestrial Reference Frame. The frame is fixed with respect to the Earth (literally, it is fixed in the crust of the Earth.) The Z-axis points in the direction of the instantaneous rotation axis of the Earth defined by IERS (its called the IRP or IERS Reference Pole). X-axis in instantaneous equatorial plane along Greenwich meridian. Y-axis completes and righ-handed system.
PEF	Pseudo-Earth Fixed Frame. This is very close to ITRF. The difference is that the Z-axis points in the direction of the "Celestial Intermediate Pole" or CIP. This pole actually moves with respect to the Earth's crustal features. The difference between the IRP and the CIP locations are given by the xp and yp EOP parameters.
TOD	True Of Date inertial frame. This is a Geocentric Equatorial Inertial (GEI) frame. Z-axis is same as PEF, but X-axis points to the vernal equinox (which is where the ecliptic crosses the equator in spring). The "True Of Date" means that this system uses the True Equator of Date. This is related to PEF by a rotation around Z-axis by an angle defined by the Greenwich Apparent Sidereal Time (GAST).
MOD	Mean Of Date inertial frame. This is the same as TOD, except that the Mean Equator of Date is used when determining the equinox location. The Mean and True equators differ due to periodic perturbations that result in Nutation. Accurate transformation between TOD and MOD is probably the most involved and for precise work, EOP corrections are needed as well.
GCRF	Geocentric Celestial Reference Frame. This is the same as MOD except for the Epoch J2000. To go from MOD to GCRF requires calculation of changes in orientation due to precession.

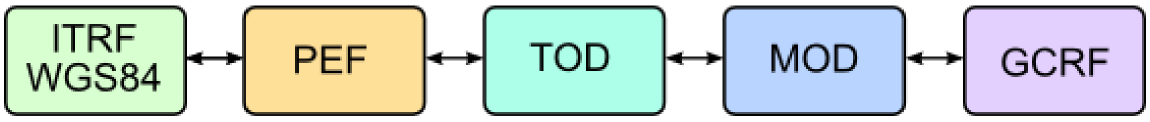


Figure 3: Relationship between coordinate systems in the IAU-1976/FK-5 reduction theory.

7.1 ITRF to PEF (Polar Motion Matrix, W)

International Terrestrial Reference Frame (ITRF) to Pseudo-Earth Fixed (PEF) frame.

$$\begin{aligned}\vec{r}_{PEF} = R_x(y_p)R_y(x_p)\vec{r}_{ITRF} &= \begin{bmatrix}\cos(x_p) & 0 & -\sin(x_p)\\ \sin(x_p)\sin(y_p) & \cos(y_p) & \cos(x_p)\sin(y_p)\\ \sin(x_p)\cos(y_p) & -\sin(y_p) & \cos(x_p)\cos(y_p)\end{bmatrix}\vec{r}_{ITRF}\\ &\approx \begin{bmatrix}1 & 0 & -x_p\\ 0 & 1 & y_p\\ x_p & -y_p & 1\end{bmatrix}\vec{r}_{ITRF} = [\![W]\!]\ \vec{r}_{ITRF}\end{aligned} \tag{1}$$

$$[\![W]\!] = \begin{bmatrix}1 & 0 & -x_p\\ 0 & 1 & y_p\\ x_p & -y_p & 1\end{bmatrix}$$

In the **`Lgm_Ctrans`** structure, the **W** matrix is called **`Awgs84_to_pef.`**

7.2 PEF to TOD (Sidereal Time Matrix, R)

$$Eq_{equinox1982} = \Delta\Psi_{1980}\cos(\bar{\epsilon})+0.00264"\sin(\Omega_{moon})+0.000063"\sin(2\,\Omega_{moon})$$

$$\theta_{GAST1982} = \theta_{GMST1982}+Eq_{equinox1982}$$

$$\begin{aligned}\vec{r}_{TOD} &= R_z(-\theta_{GAST1982})\ \vec{r}_{PEF}\\ &= \begin{bmatrix}\cos(\theta_{GMST1982}) & -\sin(\theta_{GMST1982}) & 0\\ \sin(\theta_{GMST1982}) & \cos(\theta_{GMST1982}) & 0\\ 0 & 0 & 1\end{bmatrix}\vec{r}_{PEF}\end{aligned} \tag{2}$$

$$[\![R]\!] = \begin{bmatrix}\cos(\theta_{GMST1982}) & -\sin(\theta_{GMST1982}) & 0\\ \sin(\theta_{GMST1982}) & \cos(\theta_{GMST1982}) & 0\\ 0 & 0 & 1\end{bmatrix}$$

In the **`Lgm_Ctrans`** structure, the **R** matrix is called **`Apef_to_tod.`**

7.3 TOD to MOD (Nutation Matrix, N)

Mean obliquity of the ecliptic;

$$\bar{\epsilon} = 84381.448" - 46.8150\ T_{TT}-0.00059\ T_{TT}^2 + 0.001813\ T_{TT}^3 \tag{3}$$

Delauney parameters;

$$\begin{aligned}
M_{moon} &= 134.96340251° + (1325r + 198.8675605)\ T_{TT}\ +\ 0.0088553\ T_{TT}^2\ +\ 1.4343e\text{-}5\ T_{TT}^3 \\
M_{moon} &= 357.52910918° + (99r + 359.0502911)\ T_{TT}\ -\ 0.000153\ T_{TT}^2\ +\ 3.8e\text{-}8\ T_{TT}^3 \\
u_{Mmoon} &= 93.27209062°\ +\ (1342r + 82.0174577)\ T_{TT}\ -\ 0.0035420\ T_{TT}^2\ -\ 2.88e\text{-}7\ T_{TT}^3 \\
D_{sun} &= 297.85019547°\ +\ (1236r + 307.1114469)\ T_{TT}\ -\ 0.0017696\ T_{TT}^2\ +\ 1.831e\text{-}6\ T_{TT}^3 \\
\Omega_{moon} &= 125.04455501°\ -\ (5r + 134.1361851)\ T_{TT}\ +\ 0.0020756\ T_{TT}^2\ +\ 2.139e\text{-}6\ T_{TT}^3
\end{aligned} \tag{4}$$

Compute nutation in longitude, $\Delta\Psi_{1980}$ and nutation in obliquity, $\Delta\epsilon_{1980}$ from the 106-term series and the observed or predicted EOP corrections ($\delta\Delta\Psi_{1980}$, $\delta\Delta\epsilon_{1980}$);

$$\begin{aligned}
\Delta\Psi_{1980} &= \sum_{i=1}^{106}\left(A_{p_i} + A_{p1_i} T_{TT}\right)\sin\left(a_{p_i}\right) \\
\Delta\epsilon_{1980} &= \sum_{i=1}^{106}\left(A_{e_i} + A_{e1_i} T_{TT}\right)\cos\left(a_{p_i}\right)
\end{aligned} \tag{5}$$

$$a_{p_i} = a_{1i} M_{moon} + a_{2i} M_{sun} + a_{3i} u_{Mmoon} + a_{4i} D_{sun} + a_{5i}\Omega_{moon}$$

$$\begin{aligned}
\Delta\Psi_{1980\ EopCorr} &= \Delta\Psi_{1980} + \delta\Delta\Psi_{1980} \\
\Delta\epsilon_{1980\ EopCorr} &= \Delta\epsilon_{1980} + \delta\Delta\epsilon_{1980}
\end{aligned}$$

Compute the true obliquity of the ecliptic, $\bar{\epsilon}$;

$$\epsilon = \bar{\epsilon} + \Delta\epsilon_{1980\ EopCorr} \tag{6}$$

Finally, the transformation is given by;

$$\begin{aligned}\vec{r}_{MOD} &= R_x(-\bar{\epsilon})R_z(\Delta\Psi_{1980\ \ EopCorr})Rot_x(\epsilon)\ \ \vec{r}_{TOD}\\ &= \begin{bmatrix} \cos(\Delta\Psi) & \sin(\Delta\Psi)\cos(\epsilon) & \sin(\Delta\Psi)\sin(\epsilon)\\ -\sin(\Delta\Psi)\cos(\bar{\epsilon}) & \cos(\epsilon)\cos(\bar{\epsilon})\cos(\Delta\Psi)+\sin(\bar{\epsilon})\sin(\epsilon) & \sin(\epsilon)\cos(\bar{\epsilon})\cos(\Delta\Psi)-\sin(\bar{\epsilon})\cos(\epsilon)\\ -\sin(\Delta\Psi)\sin(\bar{\epsilon}) & \cos(\epsilon)\sin(\bar{\epsilon})\cos(\Delta\Psi)-\cos(\bar{\epsilon})\sin(\epsilon) & \sin(\epsilon)\sin(\bar{\epsilon})\cos(\Delta\Psi)+\cos(\bar{\epsilon})\cos(\epsilon)\end{bmatrix}\end{aligned} \tag{7}$$

$$[\![N]\!] = \begin{bmatrix} \cos(\Delta\Psi) & \sin(\Delta\Psi)\cos(\epsilon) & \sin(\Delta\Psi)\sin(\epsilon)\\ -\sin(\Delta\Psi)\cos(\bar{\epsilon}) & \cos(\epsilon)\cos(\bar{\epsilon})\cos(\Delta\Psi)+\sin(\bar{\epsilon})\sin(\epsilon) & \sin(\epsilon)\cos(\bar{\epsilon})\cos(\Delta\Psi)-\sin(\bar{\epsilon})\cos(\epsilon)\\ -\sin(\Delta\Psi)\sin(\bar{\epsilon}) & \cos(\epsilon)\sin(\bar{\epsilon})\cos(\Delta\Psi)-\cos(\bar{\epsilon})\sin(\epsilon) & \sin(\epsilon)\sin(\bar{\epsilon})\cos(\Delta\Psi)+\cos(\bar{\epsilon})\cos(\epsilon)\end{bmatrix}$$

In the **`Lgm_Ctrans`** structure, the **N** matrix is called **`Atod_to_mod.`**

7.4 MOD to GCRF (Precession Matrix, P)

$$\begin{aligned}\zeta &= 2306.2181''\ T_{TT} + 0.30188''\ T_{TT}^2 + 0.017998''\ T_{TT}^3\\ \Theta &= 2004.3109''\ T_{TT} - 0.42665''\ T_{TT}^2 - 0.041833''\ T_{TT}^3\\ z &= 2306.2181''\ T_{TT} + 1.09468''\ T_{TT}^2 + 0.018203''\ T_{TT}^3\end{aligned} \tag{8}$$

$$\vec{r}_{GCRF} = R_z(\zeta)R_y(-\Theta)R_z(z)\ \ \vec{r}_{MOD} \tag{9}$$

$$\begin{aligned}\vec{r}_{GCRF} &= \begin{bmatrix} \cos(\Theta)\cos(z)\cos(\zeta)-\sin(z)\sin(\zeta) & \sin(z)\cos(\Theta)\cos(\zeta)+\sin(\zeta)\cos(z) & \sin(\Theta)\cos(\zeta)\\ -\sin(\zeta)\cos(\Theta)\cos(z)-\sin(z)\cos(\zeta) & -\sin(z)\sin(\zeta)\cos(\Theta)+\cos(z)\cos(\zeta) & -\sin(\Theta)\sin(\zeta)\\ -\sin(\Theta)\cos(z) & -\sin(\Theta)\sin(z) & \cos(\Theta)\end{bmatrix}\vec{r}_{MOD}\\ &= [\![P]\!]\ \vec{r}_{MOD}\end{aligned}$$

$$[\![P]\!] = \begin{bmatrix} \cos(\Theta)\cos(z)\cos(\zeta)-\sin(z)\sin(\zeta) & \sin(z)\cos(\Theta)\cos(\zeta)+\sin(\zeta)\cos(z) & \sin(\Theta)\cos(\zeta)\\ -\sin(\zeta)\cos(\Theta)\cos(z)-\sin(z)\cos(\zeta) & -\sin(z)\sin(\zeta)\cos(\Theta)+\cos(z)\cos(\zeta) & -\sin(\Theta)\sin(\zeta)\\ -\sin(\Theta)\cos(z) & -\sin(\Theta)\sin(z) & \cos(\Theta)\end{bmatrix} \tag{10}$$

In the **`Lgm_Ctrans`** structure, the **P** matrix is called **`Amod_to_gei.`**

7.5 ITRF to GCRF

$$\vec{r}_{GCRF} = [\![P]\!][\![N]\!][\![R]\!][\![W]\!]\ \ \vec{r}_{ITRF} \tag{11}$$

In the **`Lgm_Ctrans`** structure, the combined matrix **P N R W** is called **`Awgs84_to_gei.`**

8 Earth Orientation Parameters (EOP)

In the previous section, we saw that some of the transformations needed to go from a terrestrial to celestial coordinate frame involve parameters that cannot be computed or predicted on the fly. These include the polar motion angles xp and yp and the nutation corrections $\delta\Delta\epsilon$ and $\delta\Delta\Psi$. In addition, the values of DUT1 and DAT (see section on time conversions) also cannot be computed on the fly. These values are typically computed via astronomical observations and are part of the so-called Earth Orientation Parameters (EOP). The body responsible for generating the values and making them available to end-users is the International Earth Rotation Service (IERS). In addition, the US Naval Observatory (USNO) provides predictions up to about a year into the future (for some of the EOP values). The definitive (final) values and the short-term and long-term predicted values are all available online form the different agencies, but in separate files that all have slightly different formats.

The LanlGeoMag library contains a perl script called GetEopFiles that can be used to generate a consistent merged EOP datafile that is valid from 1962 to about a year into the future. The resulting file is called **`LgmEop.dat`** and provided that it exists (and is regularly updated – e.g. by running the GetEopFiles script a as a cron job) can be accessed with library routines. A sample of the LgmEop.dat file is listed below (data has been edited out at locations where there is an ellipsis). The column headers are: Date (in YYYYMMDD format), Modified Julian Date, xp and yp polar motion values (in arc-seconds), DUT1 (i.e. UT1-UTC) in seconds, extra number of seconds in the day (Length Of Day or LOD), precession and nutation corrections in arc-seconds (dPsi and dEps) for the IAU-76/IUA-80/FK5 reduction theory, dX and dY corrections in arc-seconds for the IAU-2000A theory (we currently do not use that theory). And finally, DAT (difference in atomic time scales, i.e. DAT = TAI-UTC) which in modern times is the number of leap seconds.

Note that DAT is non-integral prior to Jan 1, 1972. This is because leap seconds were only introduced in 1972 and the value of DAT prior to that time is constructed to be consistent with the system in place then. The DAT values are computed using the values given in the **`tai-utc.dat`** file`.ls`

```
#
#                                ************************************************
#                                 LanlGeoMag Merged Earth Orientation Parameters
#                                   FILE CREATED: Sat May 29 2009 13:27:10 UTC
#                                        (Julian Date: 2454828.06053241 )
#                                ************************************************
#
# Data extracted from the following files with modification times as indicated;
#
#   (Last mod.: Jul 07 2008 09:38) (JD: 2454440.90138889) (Age: 387.159143518191 days) http://maia.usno.navy.mil/ser7/tai-utc.dat
#   (Last mod.: May 28 2009 10:47) (JD: 2454826.94930556) (Age: 1.11122685158625 days)
http://hpiers.obspm.fr/eoppc/eop/eopc04_05/eopc04.62-now
#   (Last mod.: May 28 2009 10:47) (JD: 2454826.94930556) (Age: 1.11122685158625 days)
http://hpiers.obspm.fr/eoppc/eop/eopc04_05/eopc04_IAU2000.62-now
#   (Last mod.: May 28 2009 14:17) (JD: 2454827.09513889) (Age: 0.96539351856336 days) http://maia.usno.navy.mil/ser7/finals.daily
#   (Last mod.: May 28 2009 14:17) (JD: 2454827.09513889) (Age: 0.96539351856336 days) http://maia.usno.navy.mil/ser7/finals2000A.daily
#   (Last mod.: May 28 2009 14:17) (JD: 2454827.09513889) (Age: 0.96539351856336 days) http://maia.usno.navy.mil/ser7/finals.all
#   (Last mod.: May 28 2009 14:17) (JD: 2454827.09513889) (Age: 0.96539351856336 days) http://maia.usno.navy.mil/ser7/finals2000A.all
#
#                   INTERNATIONAL EARTH ROTATION AND REFERENCE SYSTEMS SERVICE
#                        EARTH ORIENTATION PARAMETERS
#                          EOP (IERS) 05 C04
#
```

```
# Date      MJD        xp          yp       UT1-UTC       LOD         dPsi        dEps         dX          dY       DAT
#(0 UTC)                "           "          s           s            "           "           "           "         s
19620101   37665   -0.012700    0.213000   0.0326338   0.0017230    0.064041    0.006305    0.000000    0.000000 1.845858
19620102   37666   -0.015900    0.214100   0.0320547   0.0016690    0.063758    0.006529    0.000000    0.000000 1.846981
19620103   37667   -0.019000    0.215200   0.0315526   0.0015820    0.063649    0.006754    0.000000    0.000000 1.848104
19620104   37668   -0.021999    0.216301   0.0311435   0.0014960    0.063673    0.006812    0.000000    0.000000 1.849228
...
19711228   41313    0.044399    0.022099  -0.1534953   0.0028140    0.047916    0.005364    0.000000    0.000000 9.881874
19711229   41314    0.040899    0.021199  -0.1535992   0.0025860    0.048007    0.005189    0.000000    0.000000 9.884466
19711230   41315    0.037399    0.020299  -0.1535161   0.0024550    0.048313    0.004982    0.000000    0.000000 9.887058
19711231   41316    0.033899    0.019499  -0.1533590   0.0024410    0.048711    0.004914    0.000000    0.000000 9.889650
19720101   41317    0.030400    0.018700  -0.0454859   0.0025390    0.049089    0.005042    0.000000    0.000000 10.000000
19720102   41318    0.026900    0.017900  -0.0481008   0.0027080    0.049412    0.005275    0.000000    0.000000 10.000000
19720103   41319    0.023300    0.017200  -0.0509077   0.0028970    0.049657    0.005467    0.000000    0.000000 10.000000
...
19720104   41320    0.019801    0.016501  -0.0538936   0.0030650    0.049763    0.005512    0.000000    0.000000 10.000000
19720105   41321    0.016301    0.015901  -0.0570195   0.0031770    0.049692    0.005393    0.000000    0.000000 10.000000
19720106   41322    0.012801    0.015301  -0.0602134   0.0032200    0.049503    0.005178    0.000000    0.000000 10.000000
...
20090524   54975    0.007052    0.526269   0.2601449   0.0009686   -0.060572   -0.009021   -0.000003    0.000009 34.000000
20090525   54976    0.010772    0.527501   0.2592603   0.0008034   -0.060842   -0.009203   -0.000003    0.000010 34.000000
20090526   54977    0.013856    0.528842   0.2585041   0.0007300   -0.060797   -0.009456   -0.000002    0.000009 34.000000
20090527   54978    0.016587    0.530152   0.2577603   0.0007844   -0.060503   -0.009551   -0.000003    0.000009 34.000000
20090528   54979    0.019357    0.531606   0.2569037   0.0009138   -0.060237   -0.009481   -0.000002    0.000009 34.000000
# starting predicted values from USNO finals.daily and finals2000A.daily files
20090529   54980    0.022099    0.532677   0.2557866   0.0000000   -0.060004   -0.009248    0.000094    0.000088 34.000000
20090530   54981    0.024918    0.533882   0.2544299   0.0000000   -0.060219   -0.009160    0.000084    0.000099 34.000000
20090531   54982    0.027824    0.534859   0.2529257   0.0000000   -0.060525   -0.009065    0.000070    0.000123 34.000000
20090601   54983    0.030871    0.535680   0.2514033   0.0000000   -0.060754   -0.008964    0.000057    0.000153 34.000000
20090602   54984    0.034023    0.536423   0.2499738   0.0000000   -0.060843   -0.008890    0.000046    0.000176 34.000000
...
20090824   55067    0.251261    0.435542   0.2372823   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090825   55068    0.252525    0.432840   0.2369542   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090826   55069    0.253743    0.430119   0.2368680   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090827   55070    0.254913    0.427380   0.2369791   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
# starting predicted values from USNO finals.all and finals2000A.all files
20090828   55071    0.256036    0.424623   0.2372197   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090829   55072    0.257112    0.421849   0.2375147   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090830   55073    0.258139    0.419059   0.2377900   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090831   55074    0.259119    0.416253   0.2379786   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20090901   55075    0.260051    0.413433   0.2380263   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
...
20100603   55350   -0.018322    0.469387  -0.0395401   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20100604   55351   -0.016155    0.470821  -0.0401183   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
20100605   55352   -0.013969    0.472222  -0.0408013   0.0000000    0.000000    0.000000    0.000000    0.000000 34.000000
```

9 EOP Routines

The coordinate transformations in the LanlGeoMag library use the EOP corrections if they are set correctly. In general this is a non-trivial task because the values must be read from auxiliary files or predicted from a constantly changing prediction model. In this section, we describe a number of routines that can be used to populate the EOP variables in the `Lgm_CTrans` structure.

9.1 NGA EOP prediction routines

On a weekly basis, The National Geospatial Intelligence Agency (NGA) generates coefficients for a series-based EOP prediction model. Note, however, that the model only predicts values for xp, yp, and DUT1. LanlGeoMag library has two routines for making use of the NGA prediction model:

```
int Lgm_ReadNgaEopp( Lgm_NgaEopp *e, int Verbosity );
void Lgm_NgaEoppPred( double t, Lgm_EopOne *eop, Lgm_NgaEopp *e );
```

These routines allow the user to read NGA EOP prediction coefficient files and compute predicted values for xp, yp, and DUT1 based on the NGA prediction series model. The latest coefficients are stored in a file called XXXXX (cant remember of hand right now).

9.2 IERS final values and USNO predicted values

The International Earth Rotation Service (IERS) creates files that contain final values of the Earth Orientation Parameters. And the US Naval Observatory (USNO) produces predicted values that extend about a year into the future. All these values are merged together into the LgmEop.dat file.

```
Lgm_Eop *Lgm_init_eop( int Verbose );
void     Lgm_destroy_eop( Lgm_Eop *e );
void     Lgm_read_eop( Lgm_Eop *e );
void     Lgm_get_eop_at_JD( double JD, Lgm_EopOne *eop, Lgm_Eop *e );
void     Lgm_set_eop( Lgm_EopOne *eop, Lgm_CTrans *c );
void     Lgm_unset_eop( Lgm_EopOne *eop, Lgm_CTrans *c );
```

These routines initialize/destroy an Lgm_Eop structure, read data into it from the `LgmEop.dat` file, and interpolate the value to a given time. The following modification to the **QuickStart.c** code shows how to set the EOP values for a given date.

```
#include <Lgm_CTrans.h>
#include <Lgm_Eop.h>
int main( ) {
    Lgm_CTrans      *c = Lgm_init_ctrans( 1 ); // more compact declaration
    Lgm_Eop         *e = Lgm_init_eop( 1 );
    Lgm_EopOne      eop;
    Lgm_Vector      Ugsm, Usm;
    long int        Date;
    double          UTC, JD;

    Date = 20040812;                  // August 12, 2004
    UTC  = 12.34567;                  // Universal Time Coordinated (in decimal hours)
    JD = Lgm_date_to_jd( Date, UTC ); // Compute JD
    Ugsm.x = -6.6; Ugsm.y = 3.4; Ugsm.z = -2.3; // Set a vector in GSM coordinates

    // Read in the EOP vals
    Lgm_read_eop( e );

    // Get (interpolate) the EOP vals from the values in the file at the given Julian Date
    Lgm_get_eop_at_JD( JD, &eop, e );

    // Set the EOP vals in the CTrans structure.
    Lgm_set_eop( &eop, c );


    // Set up all the necessary variables to do transformations for this
    // Date and UTC
    Lgm_Set_Coord_Transforms( Date, UTC, c );

    // Do the transformation from GSM->SM
    Lgm_Convert_Coords( &Ugsm, &Usm, GSM_TO_SM, c );

    // Print out the final results
    printf("Date = %8ld\n", Date);
    printf("UTC  = %lf\n", UTC);
    printf("Ugsm = %.8lf %.8lf %.8lf Re\n", Ugsm.x, Ugsm.y, Ugsm.z);
    printf("Usm  = %.8lf %.8lf %.8lf Re\n", Usm.x, Usm.y, Usm.z);

    free( c ); // free the structure
    Lgm_destroy_eop( e );
    return(0);

}
```

`Save this as QuickStartEop.c, then compile and run as follows,`

```
% gcc QuickStartEop.c -lLanlGeoMag -lgsl -lgfortran -o QuickStartEop
% ./QuickStartEop

    Earth Orientation Parameters
    ----------------------------
      Date   = 20040812
        JD   = 2453230.0144029
       MJD   = 53229.5144029
       UTC   =  12.3456700  (  12ʰ 20ᵐ 44ˢ.412 )
        xp   =   0.1042114 ″
        yp   =   0.5132679 ″
        DUT1 =  -0.4522876 s
        LOD  =  -0.0006407 s
        dPsi =  -0.0626011 ″
        dEps =  -0.0047270 ″
        dX   =   0.0003493 ″
        dY   =   0.0001915 ″
        DAT  =  32.0000000 s

Time Quantitites:
    fYear             = 2004.616160
    Date              = 20040812
    UTC               = 12.34567000000000 (  12ʰ 20ᵐ 44ˢ.41200000 )
    UT1               = 12.34554436454988 (  12ʰ 20ᵐ 43ˢ.95971238 )
    TAI               = 12.35455888888889 (  12ʰ 21ᵐ 16ˢ.41200000 )
    TT = TAI+32.184s  = 12.36349888888889 (  12ʰ 21ᵐ 48ˢ.59600000 )
    TDB               = 12.36349887715793 (  12ʰ 21ᵐ 48ˢ.59595777 )
    DUT1 = UT1-UTC    = -0.4522876 seconds
    DAT  = TAI-UTC    = 32.0000000 seconds
    JD_UTC            = 2453230.014402917 days
    JD_UT1            = 2453230.014397682 days
    JD_TT             = 2453230.015145787 days
    T_UT1             =    0.0461331800 Julian Centuries of UT1
    T_TT              =    0.0461332004 Julian Centuries of TT
    year              = 2004
    month             = 8
    day               = 12
    doy               = 225
    dow               = 4
    dowstr            = Thu
    gmst (hours)      = 9.7649192 (  09ʰ 45ᵐ 53ˢ.709 )
    gmst (degrees)    = 146.4737878 (  146° 28′ 25″.636 )
    gast (hours)      = 9.7647728 (  09ʰ 45ᵐ 53ˢ.182 )
    gast (degrees)    = 146.4715927 (  146° 28′ 17″.734 )

Eccentricity and Obliquity:
    eccentricity                     = 0.01670717
    epsilon mean (obliq. of ecliptic) = 23.43869119 (  23° 26′ 19″.288 )
    epsilon true (obliq. of ecliptic) = 23.44075681 (  23° 26′ 26″.725 )

Precession Quantities:
    Zeta              = 0.0295538 (  00° 01′ 46″.394 )
    Zee               = 0.0295543 (  00° 01′ 46″.396 )
    Theta             = 0.0256845 (  00° 01′ 32″.464 )

Nutation Quantities:
    dPsi (w.o. corrections)           = -0.00239301 ( -00° 00′ 08″.615 )
    dEps (w.o. corrections)           = 0.00206562 (  00° 00′ 07″.436 )
    ddPsi (EOP correction)            = 0.00000000 (  00° 00′ 00″.000 )
    ddEps (EOP correction)            = 0.00000000 (  00° 00′ 00″.000 )
    dPsi (w. corrections)             = -0.00239301 ( -00° 00′ 08″.615 )
    dEps (w. corrections)             = 0.00206562 (  00° 00′ 07″.436 )
    epsilon true (obliq. of ecliptic) = 23.44075681 (  23° 26′ 26″.725 )
    Equation of the Equinox           = -0.00219510 ( -00° 00′ 07″.902 )

Low Accuracy Position of Sun:
    lambda_sun      =      140.123365  (  140° 07′ 24″.113 )
    earth_sun_dist  =    23763.937342 Re
    beta_sun        =               0   (  00° 00′ 00″.000 )
    RA_sun  (MOD)   =      142.529761  (  09ʰ 30ᵐ 07ˢ.143 )
    DEC_sun (MOD)   =       14.774971  (  14° 46′ 29″.896 )
    RA_sun  (TOD)   =      142.527845  (  09ʰ 30ᵐ 06ˢ.683 )
    DEC_sun (TOD)   =       14.776983  (  14° 46′ 37″.139 )
    RA_sun  (J2000) =      142.466525  (  09ʰ 29ᵐ 51ˢ.966 )
    DEC_sun (J2000) =       14.795347  (  14° 47′ 43″.251 )

High Accuracy Position of Sun:
    lambda_sun_ha   =      140.124771  (  140° 07′ 29″.175 )
    r_sun_ha        =    23765.146671 Re
    beta_sun_ha     =    0.000142757   (  00° 00′ 00″.514 )
    RA_sun  (MOD)   =      142.531187  (  09ʰ 30ᵐ 07ˢ.485 )
    DEC_sun (MOD)   =       14.774663  (  14° 46′ 28″.785 )
    RA_sun  (TOD)   =      142.529271  (  09ʰ 30ᵐ 07ˢ.025 )
    DEC_sun (TOD)   =       14.776675  (  14° 46′ 36″.029 )
    RA_sun  (J2000) =      142.467952  (  09ʰ 29ᵐ 52ˢ.309 )
    DEC_sun (J2000) =       14.795039  (  14° 47′ 42″.142 )

Sun vector and Ecliptic Pole in GEI2000:
    Sun               = (-0.767427, 0.588234, 0.255023)
    EcPole            = (0.000000, -0.397768, 0.917486)

Geo-dipole tilt angle:
    psi                      = 18.417656  (  18° 25′ 03″.563 )
    sin_psi                  = 0.315941
    cos_psi                  = 0.948779
    tan_psi                  = 0.332998

Position of Moon:
   RA_moon                         = 101.551918  (  06ʰ 46ᵐ 12ˢ.460 )
   DEC_moon                        = 27.668707  (  27° 40′ 07″.345 )
   EarthMoonDistance               = 63.520553
   MoonPhase                       = 0.115766

IGRF-derived quantities:
    M_cd              = 30040.633779
    M_cd_McIllwain    = 31165.300000
    CD_gcolat         = 10.267478 (deg.)  (  10° 16′ 02″.919 )
    CD_glon           = -71.786745 (deg.)  ( -71° 47′ 12″.284 )
    ED_x0             = -0.062921  Re  (-401.319329 km)
    ED_y0             = 0.049572  Re  (316.174073 km)
    ED_z0             = 0.032618  Re  (208.039137 km)

Transformation Matrices:
                        [     -0.76742666       0.58823413       0.25502339 ]
    Amod_to_gse       = [     -0.64113674      -0.70410339      -0.30525742 ]
                        [      0.00000000      -0.39776755       0.91748623 ]

                        [     -0.76742666       0.58823413       0.25502339 ]
    Amod_to_gsm       = [     -0.56385314      -0.80855735       0.16823982 ]
                        [      0.30516544      -0.01468401       0.95218613 ]

                        [     -0.83306293       0.55317812       0.00034032 ]
    Agei_to_wgs84     = [     -0.55317806      -0.83306300       0.00026617 ]
                        [      0.00043075       0.00003348       0.99999991 ]

                        [     -0.76742666      -0.64113674       0.00000000 ]
    Agse_to_mod       = [      0.58823413      -0.70410339      -0.39776755 ]
                        [      0.25502339      -0.30525742       0.91748623 ]

                        [      1.00000000      -0.00000000       0.00000000 ]
    Agse_to_gsm       = [      0.00000000       0.87945849       0.47597560 ]
                        [      0.00000000      -0.47597560       0.87945849 ]

                        [     -0.83306293      -0.55317806       0.00043075 ]
    Awgs84_to_gei     = [      0.55317812      -0.83306300       0.00003348 ]
                        [      0.00034032       0.00026617       0.99999991 ]

                        [     -0.76742666      -0.56385314       0.30516544 ]
    Agsm_to_mod       = [      0.58823413      -0.80855735      -0.01468401 ]
                        [      0.25502339       0.16823982       0.95218613 ]

                        [      0.94877870       0.00000000      -0.31594143 ]
    Agsm_to_sm        = [      0.00000000       1.00000000       0.00000000 ]
                        [      0.31594143       0.00000000       0.94877870 ]

                        [      1.00000000       0.00000000       0.00000000 ]
    Agsm_to_gse       = [     -0.00000000       0.87945849      -0.47597560 ]
                        [      0.00000000       0.47597560       0.87945849 ]

                        [      0.94877870       0.00000000       0.31594143 ]
    Asm_to_gsm        = [      0.00000000       1.00000000       0.00000000 ]
                        [     -0.31594143       0.00000000       0.94877870 ]

                        [      0.99999937      -0.00103163      -0.00044828 ]
    Agei_to_mod       = [      0.00103163       0.99999947      -0.00000023 ]
                        [      0.00044828      -0.00000023       0.99999990 ]

                        [      0.99999937       0.00103163       0.00044828 ]
    Amod_to_gei       = [     -0.00103163       0.99999947      -0.00000023 ]
                        [     -0.00044828      -0.00000023       0.99999990 ]

                        [      1.00000000       0.00003832       0.00001661 ]
    Amod_to_tod       = [     -0.00003832       1.00000000      -0.00003605 ]
```

```
                        [     -0.00001661      -0.00000023       1.00000000 ]

                        [      1.00000000      -0.00003832      -0.00001661 ]
    Atod_to_mod       = [      0.00003832       1.00000000       0.00003605 ]
                        [      0.00001661      -0.00003605       1.00000000 ]

                        [     -0.83361207       0.55235036       0.00000000 ]
    Atod_to_pef       = [     -0.55235036      -0.83361207       0.00000000 ]
                        [      0.00000000       0.00000000       1.00000000 ]

                        [     -0.83361207      -0.55235036       0.00000000 ]
    Apef_to_tod       = [      0.55235036      -0.83361207       0.00000000 ]
                        [      0.00000000       0.00000000       1.00000000 ]

                        [ -8.33633230e-01   5.52318421e-01   0.00000000e+00 ]
    Ateme_to_pef      = [ -5.52318421e-01  -8.33633230e-01   0.00000000e+00 ]
                        [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                        [ -8.33633230e-01  -5.52318421e-01   0.00000000e+00 ]
    Apef_to_teme      = [  5.52318421e-01  -8.33633230e-01   0.00000000e+00 ]
                        [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                        [  1.00000000e+00   0.00000000e+00  -5.05231114e-07 ]
    Awgs84_to_pef     = [  1.25721365e-12   1.00000000e+00   2.48839317e-06 ]
                        [  5.05231114e-07  -2.48839317e-06   1.00000000e+00 ]

                        [  1.00000000e+00   1.25721365e-12   5.05231114e-07 ]
    Apef_to_wgs84     = [  0.00000000e+00   1.00000000e+00  -2.48839317e-06 ]
                        [ -5.05231114e-07   2.48839317e-06   1.00000000e+00 ]

Date = 20040812
UTC  = 12.345670
Ugsm = -6.60000000 3.40000000 -2.30000000 Re
Usm  = -5.53527410 3.40000000 -4.26740443 Re
```

Figure 4 shows xp, yp, LOD, dPsi, and dEps between January 01, 1990 and December 31 2009 (the last several months are predicted values).

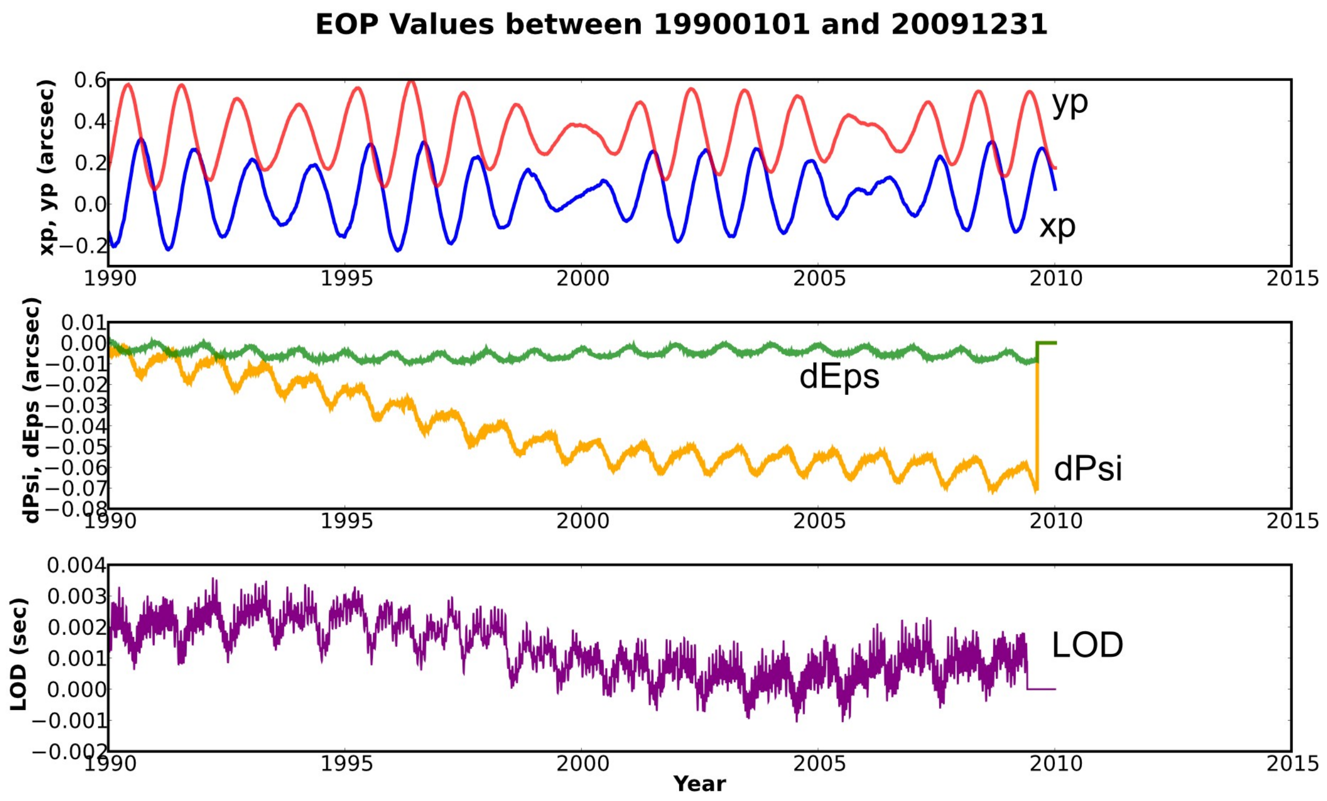


Figure 4: EOP values between January 1, 1990 and December 31 2009.LOD is the number of additional seconds in a day (i.e. above 86400s).

10 Coordinate Transformation Routines

Some additional routines are included for coordinate transformations including the following:

```
void Lgm_Set_Coord_Transforms( long int Date, double UTC, Lgm_CTrans *c );
void Lgm_Convert_Coords(Lgm_Vector *u, Lgm_Vector *v,
                              int ConvertFlag, Lgm_CTrans *c );
```

See above examples.

```
void Lgm_WGS84_to_GEOD( Lgm_Vector *uin, double *GeodLat,
                              double *GeodLong, double *GeodHieght );
void Lgm_GEOD_to_WGS84( double GeodLat, double GeodLong,
                                     double GeodHieght, Lgm_Vector *v );
```

These two routines convert between Geocentric and Geodetic coordinates in WGS84. The geocentric Cartesian coordinates are in a `Lgm_Vector` and the geodetic latitude, longitude and height are specified as individual variables.

Describe other routines... e.g. Spherical ↔ Cartesian etc... CDMAG, EDMAG etc.... There are probably more lurking about...

11 Some Validation/Verification Tests

An excellent (and modern) treatment of coordinate transformations and reductions is given by [*Vallado, 2007*]. In particular, numerous worked examples are given in the text, each with detailed intermediate results listed. A number of test codes are provided in LanlGeoMag in order to compare our results with those obtained by [*Vallado, 2007*]. The example listed below shows a comparison with values computed in his Example 3-15 (page 235-236 in the 3rd edition).

```
********************************************************************************************
"Test of Vallado's Example 3-15. Performing IAU-76/FK5 Reduction."
  (See p. 235, Vallado, "Fundamentals of Astrodynamics and Applications", 3rd Ed., 2007.)


Given:
        r_itrf = -1033.4793830 Ihat + u.y =  7901.2952754 Jhat + 6380.3565958 Khat
Find:
        r_gcrf  on April 6, 2004, 07:51:28.386009UTC


********************************************************************************************


    Setting Date and Time.
   -----------------------
    Date: 20040406
     UTC:  7.857885002500   (  07ʰ 51ᵐ 28ˢ.386 )


    Setting Earth Orientation Parameters.
   -------------------------------------
    DUT1:  -0.439961900000   ( -00ʰ 00ᵐ 00ˢ.440 )
      xp:  -0.000039078333   ( -00° 00′ 00″.141 )
      yp:   0.000092585833   (  00° 00′ 00″.333 )
   ddPsi:  -0.000014498611   ( -00° 00′ 00″.052 )
   ddEps:  -0.000001076389   ( -00° 00′ 00″.004 )


    Computing Coordinate Transformations.
   -------------------------------------
Time Quantitites:
    fYear            = 2004.265922
    Date             = 20040406
    UTC              = 7.85788500250000 (  07ʰ 51ᵐ 28ˢ.38600900 )
    UT1              = 7.85776279086111 (  07ʰ 51ᵐ 27ˢ.94604710 )
    TAI              = 7.86677389138889 (  07ʰ 52ᵐ 00ˢ.38600900 )
    TT = TAI+32.184s = 7.87571389138889 (  07ʰ 52ᵐ 32ˢ.57000900 )
    TDB              = 7.87571391055673 (  07ʰ 52ᵐ 32ˢ.57007800 )
    DUT1 = UT1-UTC   = -0.4399619 seconds
    DAT  = TAI-UTC   = 32.0000000 seconds
    JD_UTC           = 2453101.827411875 days
    JD_UT1           = 2453101.827406783 days
    JD_TT            = 2453101.828154746 days
    T_UT1            =    0.0426236114 Julian Centuries of UT1
    T_TT             =    0.0426236319 Julian Centuries of TT
    year             = 2004
    month            = 4
    day              = 6
    doy              = 97
    dow              = 2
    dowstr           = Tue
    gmst (hours)     = 20.8539930 (  20ʰ 51ᵐ 14ˢ.375 )
    gmst (degrees)   = 312.8098943 (  312° 48′ 35″.619 )
    gast (hours)     = 20.8537835 (  20ʰ 51ᵐ 13ˢ.620 )
    gast (degrees)   = 312.8067521 (  312° 48′ 24″.307 )

Eccentricity and Obliquity:
    eccentricity                      = 0.01670732
    epsilon mean (obliq. of ecliptic) = 23.43873683 (  23° 26′ 19″.453 )
    epsilon true (obliq. of ecliptic) = 23.44076738 (  23° 26′ 26″.763 )

Precession Quantities:
    Zeta             = 0.0273055 (  00° 01′ 38″.300 )
    Zee              = 0.0273059 (  00° 01′ 38″.301 )
    Theta            = 0.0237306 (  00° 01′ 25″.430 )

Nutation Quantities:
    dPsi (w.o. corrections)           = -0.00341084 ( -00° 00′ 12″.279 )
    dEps (w.o. corrections)           = 0.00203163 (  00° 00′ 07″.314 )
    ddPsi (EOP correction)            = -0.00001450 ( -00° 00′ 00″.052 )
    ddEps (EOP correction)            = -0.00000108 ( -00° 00′ 00″.004 )
    dPsi (w. corrections)             = -0.00342534 ( -00° 00′ 12″.331 )
    dEps (w. corrections)             = 0.00203056 (  00° 00′ 07″.310 )
    epsilon true (obliq. of ecliptic) = 23.44076738 (  23° 26′ 26″.763 )
    Equation of the Equinox           = -0.00314219 ( -00° 00′ 11″.312 )

Low Accuracy Position of Sun:
    lambda_sun      =       16.860732  (  16° 51′ 38″.635 )
    earth_sun_dist  =    23476.333349 Re
    beta_sun        =               0  (  00° 00′ 00″.000 )
    RA_sun  (MOD)   =       15.539485  (  01ʰ 02ᵐ 09ˢ.476 )
    DEC_sun (MOD)   =        6.625038  (  06° 37′ 30″.138 )
    RA_sun  (TOD)   =       15.536072  (  01ʰ 02ᵐ 08ˢ.657 )
    DEC_sun (TOD)   =        6.624269  (  06° 37′ 27″.370 )
    RA_sun  (J2000) =       15.484137  (  01ʰ 01ᵐ 56ˢ.193 )
    DEC_sun (J2000) =        6.602172  (  06° 36′ 07″.819 )

High Accuracy Position of Sun:
    lambda_sun_ha   =       16.859448  (  16° 51′ 34″.011 )
    r_sun_ha        =    23473.906505 Re
    beta_sun_ha     =    4.19756e-05   (  00° 00′ 00″.151 )
    RA_sun  (MOD)   =       15.538274  (  01ʰ 02ᵐ 09ˢ.186 )
    DEC_sun (MOD)   =        6.624585  (  06° 37′ 28″.505 )
    RA_sun  (TOD)   =       15.534862  (  01ʰ 02ᵐ 08ˢ.367 )
    DEC_sun (TOD)   =        6.623816  (  06° 37′ 25″.738 )
    RA_sun  (J2000) =       15.482927  (  01ʰ 01ᵐ 55ˢ.902 )
    DEC_sun (J2000) =        6.601718  (  06° 36′ 06″.187 )

Sun vector and Ecliptic Pole in GEI2000:
    Sun              = (0.957013, 0.266113, 0.115371)
    EcPole           = (0.000000, -0.397768, 0.917486)

Geo-dipole tilt angle:
    psi                     = -0.615994  ( -00° 36′ 57″.578 )
    sin_psi                 = -0.010751
    cos_psi                  = 0.999942
    tan_psi                  = -0.010752

Position of Moon:
   RA_moon                      = 206.871584  (  13ʰ 47ᵐ 29ˢ.180 )
   DEC_moon                     = -9.751673  ( -09° 45′ 06″.024 )
   EarthMoonDistance            = 57.990581
   MoonPhase                    = 0.989924

IGRF-derived quantities:
    M_cd              = 30046.614225
    M_cd_McIllwain    = 31165.300000
    CD_gcolat         = 10.281863 (deg.)  (  10° 16′ 54″.705 )
    CD_glon           = -71.769985 (deg.)  ( -71° 46′ 11″.947 )
    ED_x0             = -0.062927  Re  (-401.358582 km)
    ED_y0             = 0.049391  Re  (315.021780 km)
    ED_z0             = 0.032532  Re  (207.491698 km)

Transformation Matrices:
                        [      0.95701259       0.26611345       0.11537124 ]
    Amod_to_gse       = [     -0.29004636       0.87804557       0.38066925 ]
                        [      0.00000000      -0.39776828       0.91748591 ]

                        [      0.95701259       0.26611345       0.11537124 ]
    Amod_to_gsm       = [     -0.27987400       0.95167049       0.12646673 ]
                        [     -0.07614091      -0.15331966       0.98523888 ]

                        [      0.67886841      -0.73425991      -0.00023984 ]
    Agei_to_wgs84     = [      0.73425985       0.67886845      -0.00031223 ]
                        [      0.00039208       0.00003586       0.99999992 ]

                        [      0.95701259      -0.29004636       0.00000000 ]
    Agse_to_mod       = [      0.26611345       0.87804557      -0.39776828 ]
                        [      0.11537124       0.38066925       0.91748591 ]

                        [      1.00000000       0.00000000      -0.00000000 ]
    Agse_to_gsm       = [     -0.00000000       0.96492848      -0.26251289 ]
                        [     -0.00000000       0.26251289       0.96492848 ]

                        [      0.67886841       0.73425985       0.00039208 ]
    Awgs84_to_gei     = [     -0.73425991       0.67886845       0.00003586 ]
                        [     -0.00023984      -0.00031223       0.99999992 ]

                        [      0.95701259      -0.27987400      -0.07614091 ]
    Agsm_to_mod       = [      0.26611345       0.95167049      -0.15331966 ]
                        [      0.11537124       0.12646673       0.98523888 ]

                        [      0.99994221       0.00000000       0.01075092 ]
    Agsm_to_sm        = [      0.00000000       1.00000000       0.00000000 ]
                        [     -0.01075092       0.00000000       0.99994221 ]

                        [      1.00000000      -0.00000000      -0.00000000 ]
    Agsm_to_gse       = [      0.00000000       0.96492848       0.26251289 ]
                        [     -0.00000000      -0.26251289       0.96492848 ]

                        [      0.99994221       0.00000000      -0.01075092 ]
    Asm_to_gsm        = [      0.00000000       1.00000000       0.00000000 ]
                        [      0.01075092       0.00000000       0.99994221 ]

                        [      0.99999946      -0.00095315      -0.00041418 ]
```

```
Agei_to_mod       = [      0.00095315       0.99999955      -0.00000020 ]
                    [      0.00041418      -0.00000020       0.99999991 ]

                    [      0.99999946       0.00095315       0.00041418 ]
Amod_to_gei       = [     -0.00095315       0.99999955      -0.00000020 ]
                    [     -0.00041418      -0.00000020       0.99999991 ]

                    [      1.00000000       0.00005485       0.00002378 ]
Amod_to_tod       = [     -0.00005485       1.00000000      -0.00003544 ]
                    [     -0.00002378      -0.00000020       1.00000000 ]

                    [      1.00000000      -0.00005485      -0.00002378 ]
Atod_to_mod       = [      0.00005485       1.00000000       0.00003544 ]
                    [      0.00002378      -0.00003544       1.00000000 ]

                    [      0.67952777      -0.73364979       0.00000000 ]
Atod_to_pef       = [      0.73364979       0.67952777       0.00000000 ]
                    [      0.00000000       0.00000000       1.00000000 ]

                    [      0.67952777       0.73364979       0.00000000 ]
Apef_to_tod       = [     -0.73364979       0.67952777       0.00000000 ]
                    [      0.00000000       0.00000000       1.00000000 ]

                    [  6.79568000e-01  -7.33612523e-01   0.00000000e+00 ]
Ateme_to_pef      = [  7.33612523e-01   6.79568000e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [  6.79568000e-01   7.33612523e-01   0.00000000e+00 ]
Apef_to_teme      = [ -7.33612523e-01   6.79568000e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [  1.00000000e+00   0.00000000e+00   6.82045583e-07 ]
Awgs84_to_pef     = [ -1.10213630e-12   1.00000000e+00   1.61592763e-06 ]
                    [ -6.82045583e-07  -1.61592763e-06   1.00000000e+00 ]

                    [  1.00000000e+00  -1.10213630e-12  -6.82045583e-07 ]
Apef_to_wgs84     = [  0.00000000e+00   1.00000000e+00  -1.61592763e-06 ]
                    [  6.82045583e-07   1.61592763e-06   1.00000000e+00 ]


 Setting ITRF Coordinates (km).
--------------------------------
 u_itrf: -1033.479383000  7901.295275400  6380.356595800


 Transforming to PEF Coordinates (km).
---------------------------------------
  u_pef: -1033.475031306   7901.305585585   6380.344532749
  u_pef: -1033.475031300   7901.305585600   6380.344532800  (Vallado's result)
   DIFF:    -0.000000006     -0.000000015     -0.000000051  (LGM - Vallado's result)


 Transforming to TOD Coordinates (km).
---------------------------------------
  u_tod:  5094.516203638   6127.365277834   6380.344532749
  u_tod:  5094.514780400   6127.366461200   6380.344532800  (Vallado's result)
   DIFF:     0.001423238     -0.001183366     -0.000000051  (LGM - Vallado's result)
```

```
    Transforming to MOD Coordinates (km).
   --------------------------------------
     u_mod: 5094.02837421  6127.87081613  6380.24851689
     u_mod:  5094.028374500   6127.870816400   6380.248516400  (Vallado's result)
      DIFF:    -0.000000287     -0.000000269      0.000000486  (LGM - Vallado's result)


    Transforming to GCRF Coordinates (km).
   --------------------------------------
     u_gcrf:  5102.508957168   6123.011400718   6378.136928791
     u_gcrf:  5102.508953000   6123.011396000   6378.136937000  (Vallado's result)
      DIFF:     0.000004168      0.000004718     -0.000008209  (LGM - Vallado's result)
```

The differences between our values and Vallado's values are typically much less than a meter (often on the order of a cm or so). The one exception in the example shown above is the TOD vector for which there is a difference on the order of a meter in both the x and y components. It is not entirely clear why a larger discrepancy exists only for that vector, but we suspect that it may be an error in Vallado's example (we submitted an errata and we'll see what he/the publisher says). The same calculations worked with no (i.e. zeroed) ddEps and ddPsi EOP values are given below. These and the above calculations should be compared with the table 3-6 in [Vallado, 2007].

```
*******************************************************************************************
"Test of Vallado's Example 3-15. Performing IAU-76/FK5 Reduction."
  (See p. 235, Vallado, "Fundamentals of Astrodynamics and Applications", 3rd Ed., 2007.)


Given:
        r_itrf = -1033.4793830 Ihat + u.y =  7901.2952754 Jhat + 6380.3565958 Khat
Find:
        r_gcrf  on April 6, 2004, 07:51:28.386009UTC


*******************************************************************************************


    Setting Date and Time.
   -----------------------
    Date: 20040406
     UTC:  7.857885002500   (  07ʰ 51ᵐ 28ˢ.386 )


    Setting Earth Orientation Parameters.
   -------------------------------------
    DUT1:  -0.439961900000   ( -00ʰ 00ᵐ 00ˢ.440 )
      xp:  -0.000039078333   ( -00° 00′ 00″.141 )
      yp:   0.000092585833   (  00° 00′ 00″.333 )
   ddPsi:   0.000000000000   (  00° 00′ 00″.000 )
   ddEps:   0.000000000000   (  00° 00′ 00″.000 )


    Computing Coordinate Transformations.
   -------------------------------------
Time Quantitites:
    fYear              = 2004.265922
    Date               = 20040406
    UTC                = 7.85788500250000 (  07ʰ 51ᵐ 28ˢ.38600900 )
    UT1                = 7.85776279086111 (  07ʰ 51ᵐ 27ˢ.94604710 )
    TAI                = 7.86677389138889 (  07ʰ 52ᵐ 00ˢ.38600900 )
    TT = TAI+32.184s   = 7.87571389138889 (  07ʰ 52ᵐ 32ˢ.57000900 )
    TDB                = 7.87571391055673 (  07ʰ 52ᵐ 32ˢ.57007800 )
    DUT1 = UT1-UTC     = -0.4399619 seconds
    DAT  = TAI-UTC     = 32.0000000 seconds
    JD_UTC             = 2453101.827411875 days
    JD_UT1             = 2453101.827406783 days
    JD_TT              = 2453101.828154746 days
    T_UT1              =    0.0426236114 Julian Centuries of UT1
    T_TT               =    0.0426236319 Julian Centuries of TT
    year               = 2004
    month              = 4
    day                = 6
    doy                = 97
    dow                = 2
    dowstr             = Tue
    gmst (hours)       = 20.8539930 (  20ʰ 51ᵐ 14ˢ.375 )
    gmst (degrees)     = 312.8098943 (  312° 48′ 35″.619 )
    gast (hours)       = 20.8537844 (  20ʰ 51ᵐ 13ˢ.624 )
    gast (degrees)     = 312.8067654 (  312° 48′ 24″.355 )

Eccentricity and Obliquity:
    eccentricity                      = 0.01670732
    epsilon mean (obliq. of ecliptic) = 23.43873683 (  23° 26′ 19″.453 )
    epsilon true (obliq. of ecliptic) = 23.44076846 (  23° 26′ 26″.766 )

Precession Quantities:
    Zeta               = 0.0273055 (  00° 01′ 38″.300 )
    Zee                = 0.0273059 (  00° 01′ 38″.301 )
    Theta              = 0.0237306 (  00° 01′ 25″.430 )

Nutation Quantities:
    dPsi (w.o. corrections)           = -0.00341084 ( -00° 00′ 12″.279 )
    dEps (w.o. corrections)           = 0.00203163 (  00° 00′ 07″.314 )
    ddPsi (EOP correction)            = 0.00000000 (  00° 00′ 00″.000 )
    ddEps (EOP correction)            = 0.00000000 (  00° 00′ 00″.000 )
    dPsi (w. corrections)             = -0.00341084 ( -00° 00′ 12″.279 )
    dEps (w. corrections)             = 0.00203163 (  00° 00′ 07″.314 )
    epsilon true (obliq. of ecliptic) = 23.44076846 (  23° 26′ 26″.766 )
    Equation of the Equinox           = -0.00312888 ( -00° 00′ 11″.264 )

Low Accuracy Position of Sun:
    lambda_sun      =       16.860732  (  16° 51′ 38″.635 )
    earth_sun_dist  =    23476.333349 Re
    beta_sun        =             0   (  00° 00′ 00″.000 )
    RA_sun  (MOD)   =       15.539485  (  01ʰ 02ᵐ 09ˢ.476 )
    DEC_sun (MOD)   =        6.625038  (  06° 37′ 30″.138 )
    RA_sun  (TOD)   =       15.536086  (  01ʰ 02ᵐ 08ˢ.661 )
    DEC_sun (TOD)   =        6.624275  (  06° 37′ 27″.391 )
    RA_sun  (J2000) =       15.484137  (  01ʰ 01ᵐ 56ˢ.193 )
    DEC_sun (J2000) =        6.602172  (  06° 36′ 07″.819 )

High Accuracy Position of Sun:
    lambda_sun_ha   =       16.859448  (  16° 51′ 34″.011 )
    r_sun_ha        =    23473.906505 Re
    beta_sun_ha     =    4.19756e-05   (  00° 00′ 00″.151 )
    RA_sun  (MOD)   =       15.538274  (  01ʰ 02ᵐ 09ˢ.186 )
    DEC_sun (MOD)   =        6.624585  (  06° 37′ 28″.505 )
    RA_sun  (TOD)   =       15.534875  (  01ʰ 02ᵐ 08ˢ.370 )
    DEC_sun (TOD)   =        6.623822  (  06° 37′ 25″.759 )
    RA_sun  (J2000) =       15.482927  (  01ʰ 01ᵐ 55ˢ.902 )
    DEC_sun (J2000) =        6.601718  (  06° 36′ 06″.187 )

Sun vector and Ecliptic Pole in GEI2000:
    Sun              = (0.957013, 0.266113, 0.115371)
    EcPole           = (0.000000, -0.397768, 0.917486)

Geo-dipole tilt angle:
    psi                     = -0.615994  ( -00° 36′ 57″.578 )
    sin_psi                 = -0.010751
    cos_psi                 = 0.999942
    tan_psi                 = -0.010752

Position of Moon:
   RA_moon                     = 206.871584  (  13ʰ 47ᵐ 29ˢ.180 )
   DEC_moon                    = -9.751673  ( -09° 45′ 06″.024 )
   EarthMoonDistance           = 57.990581
   MoonPhase                   = 0.989924

IGRF-derived quantities:
    M_cd             = 30046.614225
    M_cd_McIllwain   = 31165.300000
    CD_gcolat        = 10.281863 (deg.)  (  10° 16′ 54″.705 )
    CD_glon          = -71.769985 (deg.)  ( -71° 46′ 11″.947 )
    ED_x0            = -0.062927  Re  (-401.358582 km)
    ED_y0            = 0.049391  Re  (315.021780 km)
    ED_z0            = 0.032532  Re  (207.491698 km)

Transformation Matrices:
                       [      0.95701259       0.26611345       0.11537124 ]
    Amod_to_gse      = [     -0.29004636       0.87804557       0.38066925 ]
                       [      0.00000000      -0.39776828       0.91748591 ]

                       [      0.95701259       0.26611345       0.11537124 ]
    Amod_to_gsm      = [     -0.27987400       0.95167049       0.12646673 ]
                       [     -0.07614091      -0.15331966       0.98523888 ]

                       [      0.67886841      -0.73425991      -0.00023989 ]
    Agei_to_wgs84    = [      0.73425985       0.67886845      -0.00031232 ]
                       [      0.00039218       0.00003588       0.99999992 ]

                       [      0.95701259      -0.29004636       0.00000000 ]
    Agse_to_mod      = [      0.26611345       0.87804557      -0.39776828 ]
                       [      0.11537124       0.38066925       0.91748591 ]

                       [      1.00000000       0.00000000      -0.00000000 ]
    Agse_to_gsm      = [     -0.00000000       0.96492848      -0.26251289 ]
                       [     -0.00000000       0.26251289       0.96492848 ]

                       [      0.67886841       0.73425985       0.00039218 ]
    Awgs84_to_gei    = [     -0.73425991       0.67886845       0.00003588 ]
                       [     -0.00023989      -0.00031232       0.99999992 ]

                       [      0.95701259      -0.27987400      -0.07614091 ]
    Agsm_to_mod      = [      0.26611345       0.95167049      -0.15331966 ]
                       [      0.11537124       0.12646673       0.98523888 ]
```

```
                    [      0.99994221       0.00000000       0.01075092 ]
Agsm_to_sm        = [      0.00000000       1.00000000       0.00000000 ]
                    [     -0.01075092       0.00000000       0.99994221 ]

                    [      1.00000000      -0.00000000      -0.00000000 ]
Agsm_to_gse       = [      0.00000000       0.96492848       0.26251289 ]
                    [     -0.00000000      -0.26251289       0.96492848 ]

                    [      0.99994221       0.00000000      -0.01075092 ]
Asm_to_gsm        = [      0.00000000       1.00000000       0.00000000 ]
                    [      0.01075092       0.00000000       0.99994221 ]

                    [      0.99999946      -0.00095315      -0.00041418 ]
Agei_to_mod       = [      0.00095315       0.99999955      -0.00000020 ]
                    [      0.00041418      -0.00000020       0.99999991 ]

                    [      0.99999946       0.00095315       0.00041418 ]
Amod_to_gei       = [     -0.00095315       0.99999955      -0.00000020 ]
                    [     -0.00041418      -0.00000020       0.99999991 ]

                    [      1.00000000       0.00005462       0.00002368 ]
Amod_to_tod       = [     -0.00005462       1.00000000      -0.00003546 ]
                    [     -0.00002368      -0.00000020       1.00000000 ]

                    [      1.00000000      -0.00005462      -0.00002368 ]
Atod_to_mod       = [      0.00005462       1.00000000       0.00003546 ]
                    [      0.00002368      -0.00003546       1.00000000 ]

                    [      0.67952794      -0.73364963       0.00000000 ]
Atod_to_pef       = [      0.73364963       0.67952794       0.00000000 ]
                    [      0.00000000       0.00000000       1.00000000 ]

                    [      0.67952794       0.73364963       0.00000000 ]
Apef_to_tod       = [     -0.73364963       0.67952794       0.00000000 ]
                    [      0.00000000       0.00000000       1.00000000 ]

                    [  6.79568000e-01  -7.33612523e-01   0.00000000e+00 ]
Ateme_to_pef      = [  7.33612523e-01   6.79568000e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [  6.79568000e-01   7.33612523e-01   0.00000000e+00 ]
Apef_to_teme      = [ -7.33612523e-01   6.79568000e-01   0.00000000e+00 ]
                    [  0.00000000e+00   0.00000000e+00   1.00000000e+00 ]

                    [  1.00000000e+00   0.00000000e+00   6.82045583e-07 ]
Awgs84_to_pef     = [ -1.10213630e-12   1.00000000e+00   1.61592763e-06 ]
                    [ -6.82045583e-07  -1.61592763e-06   1.00000000e+00 ]

                    [  1.00000000e+00  -1.10213630e-12  -6.82045583e-07 ]
Apef_to_wgs84     = [  0.00000000e+00   1.00000000e+00  -1.61592763e-06 ]
                    [  6.82045583e-07   1.61592763e-06   1.00000000e+00 ]


Setting ITRF Coordinates (km).
-------------------------------
u_itrf: -1033.479383000  7901.295275400  6380.356595800
```

```
    Transforming to PEF Coordinates (km).
   ---------------------------------------
     u_pef: -1033.475031306   7901.305585585   6380.344532749
     u_pef: -1033.475031300   7901.305585600   6380.344532800  (Vallado's result)
      DIFF:    -0.000000006     -0.000000015     -0.000000051  (LGM - Vallado's result)


    Transforming to TOD Coordinates (km).
   ---------------------------------------
     u_tod:  5094.514781057   6127.366460619   6380.344532749
     u_tod:  5094.514780400   6127.366461200   6380.344532800  (Vallado's result)
      DIFF:     0.000000657     -0.000000581     -0.000000051  (LGM - Vallado's result)


    Transforming to MOD Coordinates (km).
   ---------------------------------------
     u_mod: 5094.02901646  6127.87093603  6380.24788896
     u_mod:  5094.029016700   6127.870936300   6380.247888500  (Vallado's result)
      DIFF:    -0.000000236     -0.000000273      0.000000456  (LGM - Vallado's result)


    Transforming to GCRF Coordinates (km).
   ---------------------------------------
     u_gcrf:  5102.509599273   6123.011520001   6378.136300595
     u_gcrf:  5102.509600000   6123.011520000   6378.136300000  (Vallado's result)
      DIFF:    -0.000000727      0.000000001      0.000000595  (LGM - Vallado's result)
```

12 WGS84 and the Shape of the Earth

The Earth is not a perfect sphere. In reality it is significantly flattened in the pole-to-pole direction. In the LanlGeoMag library, we adopt the WGS84 Earth model which approximates the shape of the Earth as a spheroid with an equatorial radius, a=6378.1370 km and a polar radius of b=6356.7523142 km. (The extra precision on the polar radius is retained to make the ratio b/a = 1-f, where f is the flattening factor.) The full set of parameters in the WGS84 model are defined in the **Lgm_WGS84.h** file (shown below).

```
#ifndef LGM_WGS84_H
#define LGM_WGS84_H

// World Geodetic System 1984 (WGS84) parameters
#define WGS84_A         6378.1370              // semi-major axis of earth
(equatorial radius) in km
#define WGS84_B         6356.7523142           // semi-minor axis of earth (polar
radius) in km ( derived from b=a(1-f) )
#define WGS84_F         0.0033528106718309896  // f Flattening factor
#define WGS84_FINV      298.2572229328697      // 1/f where f is flattening factor
#define WGS84_E2        0.00669437999014       // first eccentricity squared ( e^2
= 1-b^2/a^2 = 2f-f^2)
#define WGS84_E         0.08181919092890624    // first eccentricity ( e )
#define WGS84_EP2       0.006739496756586903   // second eccentricity squared
( ep^2 = a^2/b^2-1 = f*(2-f)/(1-f)^2 )
#define WGS84_EP        0.08209443803685366    // second eccentricity ep
#define WGS84_A2        40680631.59076899      // WGS84_A * WGS84_A (km^2)
#define WGS84_B2        40408299.98408706      // WGS84_B * WGS84_B (km^2)
```

```
#define WGS84_A2mB2     272331.6066819355       // WGS84_A2 - WGS84_B2 (km^2)
#define WGS84_E4        4.481472364144719e-05   // WGS84_E2*WGS84_E2
#define WGS84_1mE2      0.993305619995739       // 1 - WGS84_E2

#endif
```

When distances are put into units of Earth-radii, we use the value defined for WGS84_A (6378.1370 km). A number of the other parameters defined in **`Lgm_WGS84.h`** are necessary for converting between geodetic and geocentric coordinates.

12.1 Geodetic Versus Geocentric Coordinates

A point on the surface of the Earth has a specific geocentric latitude and longitude in spherical polar coordinates regardless of the shape of the Earth. If the Earth were a true sphere, the radial distance from the center of the Earth to any point on the surface would be constant. For a flattened Earth, the radial distance varies as a function of latitude. But in either case the geocentric latitude (and longitude) are the same.

However, relative to a non-spherical Earth, an important drawback to a geocentric coordinate system is that a line from the center of the Earth to a point on the surface is not normal to the surface. To overcome this problem, “geodetic coordinates” were introduced. The longitude is the same in these coordinates, but the latitude is measured as the angle from the equatorial plane to the normal passing through the point. And the height is measured along the normal direction relative to the surface (as one would expect). Figure 5 shows the relationship between geocentric and geodetic latitude. Most points on the surface of the Earth are usually referenced relative to geodetic coordinates not geocentric. GPS-based coordinates and IGRF field models are also referenced to geodetic coordinates.

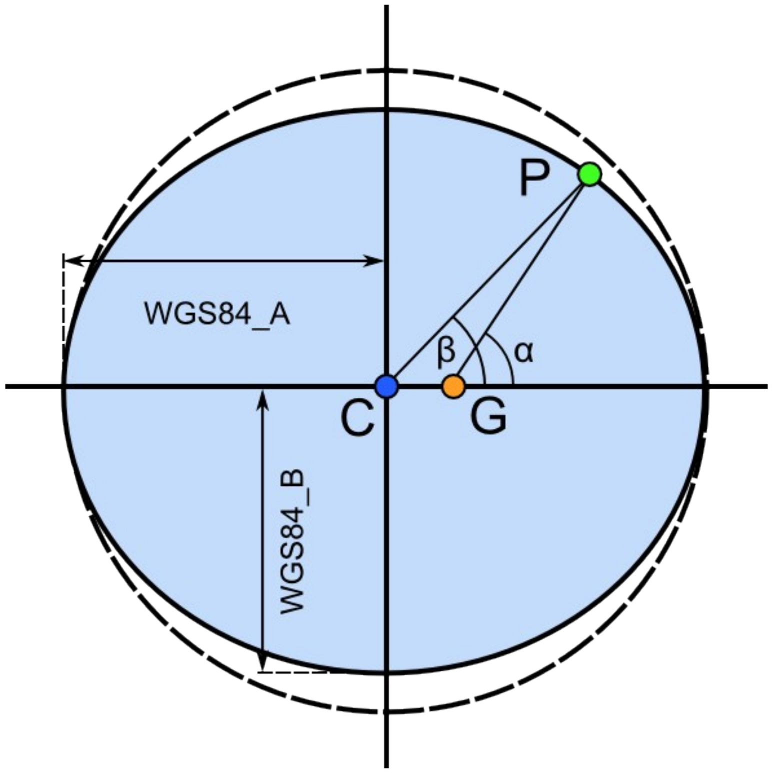


Figure 5: Geocentric versus geodetic coordinates. The geodetic latitude is measured relative to the normal (GP) whereas the geocentric latitude is measured relative to radial vector (CP). Heights in geocentric coordinates are not normal to the surface whereas they are in geodetic.

12.2 Conversion Between Geodetic and Geocentric Coordinates

Two routines are provided to convert between geodetic and geocentric coordinates;

```
void Lgm_GEOD_to_WGS84( double GeodLat, double GeodLong, double GeodHeight,
      Lgm_Vector *v )

void Lgm_WGS84_to_GEOD( Lgm_Vector *uin, double *GeodLat, double *GeodLong, double
      *GeodHeight )
```

An example of how to use these routines is given below. In this code, we take a geodetic position and convert it to geocentric with **`Lgm_GEOD_to_WGS84()`** then we go backwards with **`Lgm_WGS84_to_GEOD()`** to verify that we get what we started with.

```
// Geodetic.c
#include <Lgm_CTrans.h>
main(){

    double      GeodLat, GeodLong, GeodHeight, r;
    Lgm_Vector  v;
    GeodLat    = 35.0 + 53.0/60.0 + 17.0/3600.0;   // degrees
    GeodLong   = -106.0 - 18.0/60.0 - 23.0/3600.0; // degrees
    GeodHeight = 20.0; // km

    Lgm_GEOD_to_WGS84( GeodLat, GeodLong, GeodHeight, &v );

    printf( "           Geodetic Position\n");
    printf( "   ===================================\n");
    printf( "      GeodLat    : %lf degrees\n", GeodLat );
    printf( "      GeodLong   : %lf degrees\n", GeodLong );
    printf( "      GeodHeight : %lf km\n\n", GeodHeight );

    printf( "           Geocentric Position\n");
    printf( "   ===================================\n");
    printf( "      X         : %-15lf meters\n", v.x*WGS84_A*1000.0);   // meters
    printf( "      Y         : %-15lf meters\n", v.y*WGS84_A*1000.0);   // meters
    printf( "      Z         : %-15lf meters\n", v.z*WGS84_A*1000.0); // meters
    r = sqrt(v.x*v.x + v.y*v.y + v.z*v.z);
    printf( "      Radius    : %lf Re (%lf km)\n", r, r*WGS84_A );
    printf( "      Latitude  : %lf degrees\n", DegPerRad*asin( v.z/r ) );
    printf( "      Longitude : %lf degrees\n\n", DegPerRad*atan2( v.y, v.x ) );

    // Go back to verify that we get what we started with.
    Lgm_WGS84_to_GEOD( &v, &GeodLat, &GeodLong, &GeodHeight );

    printf( "           Geodetic Position\n");
    printf( "   ===================================\n");
    printf( "      GeodLat    : %lf degrees\n", GeodLat );
    printf( "      GeodLong   : %lf degrees\n", GeodLong );
    printf( "      GeodHeight : %lf km\n\n", GeodHeight );

}
```

Running this gives;

```
% gcc Geodetic.c -lm -lLanlGeoMag -lgsl -lgfortran -o Geodetic
% ./Geodetic

            Geodetic Position
   ===================================
      GeodLat    : 35.888056 degrees
      GeodLong   : -106.306389 degrees
      GeodHeight : 20.000000 km

            Geocentric Position
   ===================================
      X         : -1457073.371556 meters
      Y         : -4980740.582359 meters
      Z         : 3729859.716138  meters
      Radius    : 1.001990 Re (6390.828824 km)
      Latitude  : 35.706047 degrees
      Longitude : -106.306389 degrees
```

```
            Geodetic Position
   ===================================
      GeodLat    : 35.888056 degrees
      GeodLong   : -106.306389 degrees
      GeodHeight : 20.000000 km
```

The results obtained with an on-line tool on the NOAA website (see **http://www.ngs.noaa.gov/cgi-bin/xyz_getxyz.prl**) for this same position are;

```
 =============================================================
          Latitude            Longitude      Ellip_Ht   Ellipsoid
 INPUT =   N355317.0           W1061823.0     20000.     GRS80
 =============================================================


  X (Meters)    Y (Meters)    Z (Meters)    ELLIPSOID
  ------------- ------------- ------------- ---------

  -1457073.3716 -4980740.5824  3729859.7160 GRS80
```

These results are virtually identical to those obtained with the LanlGeoMag code. And the **Geodetic.c** code verifies that we recover what we started with. Note that the geocentric latitude differs quite significantly from the geodetic latitude. At the equator and the poles, the latitudes are the same (because even in a flattened Earth the surface is normal to the geocentric radial vector there), and the deviation is largest for middle latitudes.

13 Magnetic and Solar Based Coordinate Systems

Coordinate systems commonly used in space physics include, Geocentric Solar Magnetospheric (GSM), Solar Magnetic (SM), Geocentric Solar Ecliptic (GSE), Center Dipole (CD) Magnetic, and Eccentric Dipole (ED) Magnetic. These coordinate system are defined below;

GSM Geocentric Solar Magnetospheric . X-axis points to Sun. Earth's magnetic dipole axis lies in the X-Z plane. Y-axis completes a right-handed system.

SM Solar Magnetic. Z-axis parallel to Earth's magnetic dipole axis. Direction to Sun lies in the X-Z plane. Y-axis completes a right-handed system. Because dipole axis is along Z, the equatorial plane is the magnetic equator. GSM and SM are related by a rotation around the Y-axis which is the same in both.

GSE Geocentric Solar Ecliptic. X-axis points to Sun. Z-axis parallel to ecliptic north pole. Y-axis completes and righ-handed system.

CD Centered Dipole Magnetic. Z-axis parallel to dipole axis. Y-axis parallel to both dipole axis and rotation axis so that Y=DxS (where D is dipole vector and S is south pole).

ED Eccentric Dipole Magnetic. Same as CD, but its not geocentric. Its offset from the center of the Earth by several hundred km's toward the Pacific. The offset is important in understanding the South Atlantic Anomaly.

These coordinate systems are often implemented slightly differently due to different methods used for calculating the dipole positions and the direction to the Sun. For the dipole position, we use a fully time-dependent evaluation of the first 8 IGRF/DGRF coefficients in order to obtain the dipole position and its offset from the center of the Earth. For the Sun, we use two different methods – a low accuracy method and a high accuracy method (using multiple planetary perturbation calculations) . (Currently both are calculated, but the high accuracy method is hard-wired for the coordinate systems – need to add a switch to let user decide which to use.)

Its also important to note that algorithms for computing the position of the Sun typically provide results in a “Mean Of Date” system, not in the ICRF2000 (GEI2000) system. Transforming between GSM and GEI2000 should then involve going to the MOD system first and then transforming from MOD to GEI2000.

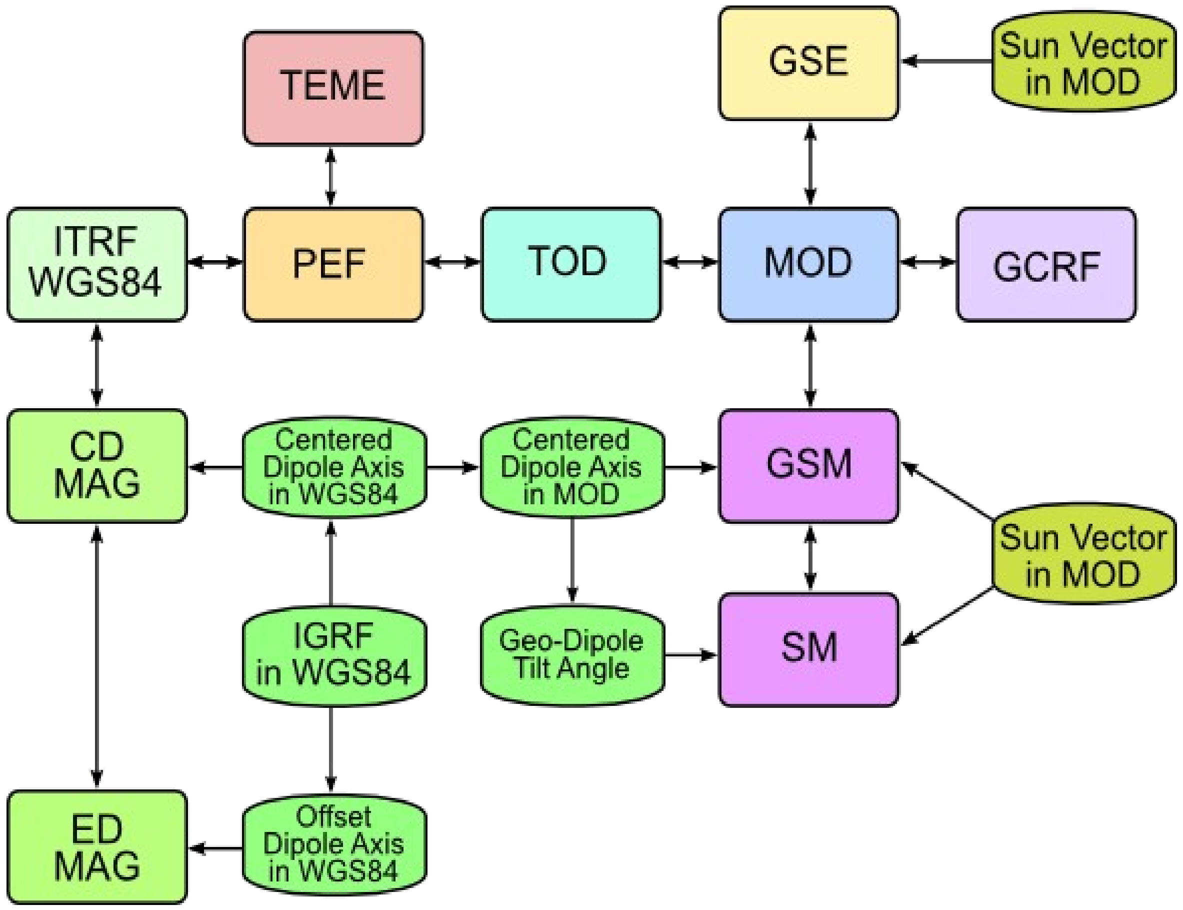


Figure 6: Relationship of other coordinate systems to terrestrial and celestial inertial systems.

14 Position of the Sun

The position of the Sun can be calculated in two different ways. There is a low-accuracy method and a high accuracy method. In both cases, the position is natively computed in MOD coordinates. Figure 2 shows how each method compares with the J2000 position of the Sun as determined by the Jet Propulsion Laboratory (JPL) for the entire year of 2007. For this comparison, the LGM positions have been converted to the J2000 system using full EOP corrections. As can be seen, the low accuracy method is accurate to about 50 arc-seconds while the high accuracy method is accurate to better than about an arc-second. Although there are still some obvious periodicities present in the residuals, the high-accuracy method appears to be very accurate indeed. In fact, the differences appear to be so small that the noise associated with the JPL numerical computations can be seen clearly. By default the high accuracy method is used in the LanlGeoMag library.

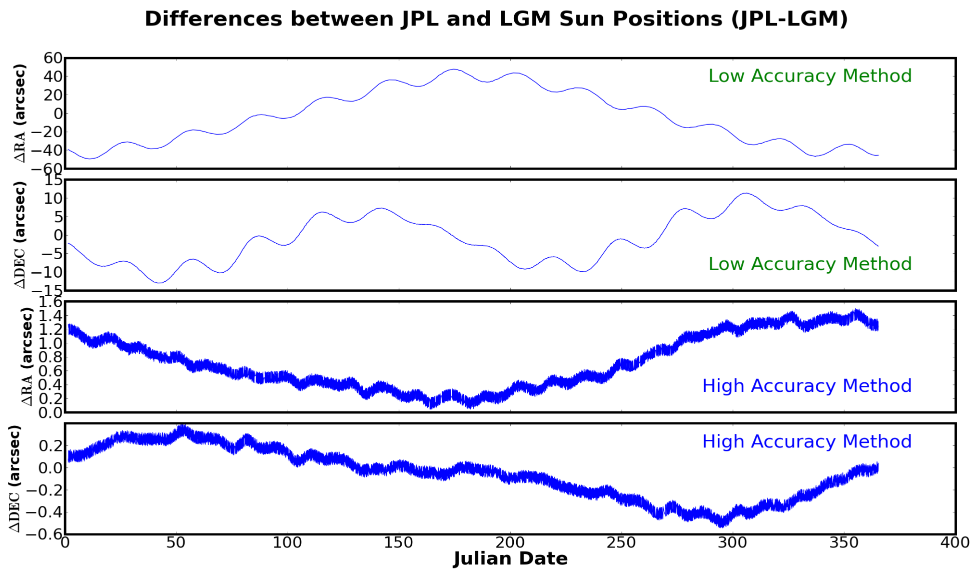


Figure 7: Differences between J2000 Sun Positions. The upper two panels show the difference between JPL's definitive positions and those obtained with the low accuracy method in LanlGeoMag. The bottom two panels show the difference between JPL and the high accuracy method in the LanlGeoMag library.

15 Time Dependence of IGRF/DGRF Coefficients

IGRF models are released every five years on years ending with a 0 or a 5 (e.g. 2000, 2005, 2010, etc.), and the set of coefficients are valid for the given epoch. For example, the coefficients in the 2000 model are valid for the time 2000.0. It is important to note that the coefficients change with time, and they must be updated for a given time of interest. Since the library contains all IGRF/DGRF models from 1900 to present, we are able to interpolate the coefficients to any time between 1900 and the time of the latest model. Extrapolation is necessary for times beyond the epoch of the latest model. Currently, we use linear interpolation for times between 1900 and the epoch of the latest IGRF, and for times beyond the epoch of the latest IGRF, we use linear extrapolation based on the previous two IGRF models. In future releases, we plan to include options to utilize more of the models by making use of polynomial or rational function interpolation/extrapolation.

16 Location of the Centered Dipole and Offset Dipole

The position and orientation of the Earth's magnetic dipole axis is determined in a fully time-dependent manner from the first 8 coefficients in the IGRF/DGRF spherical harmonic expansions. For a given date and time, the time-dependent IGRF coefficients are first determined via interpolation or extrapolation. Then the location of the centered dipole and its offset from the Earth's center are computed as described below.

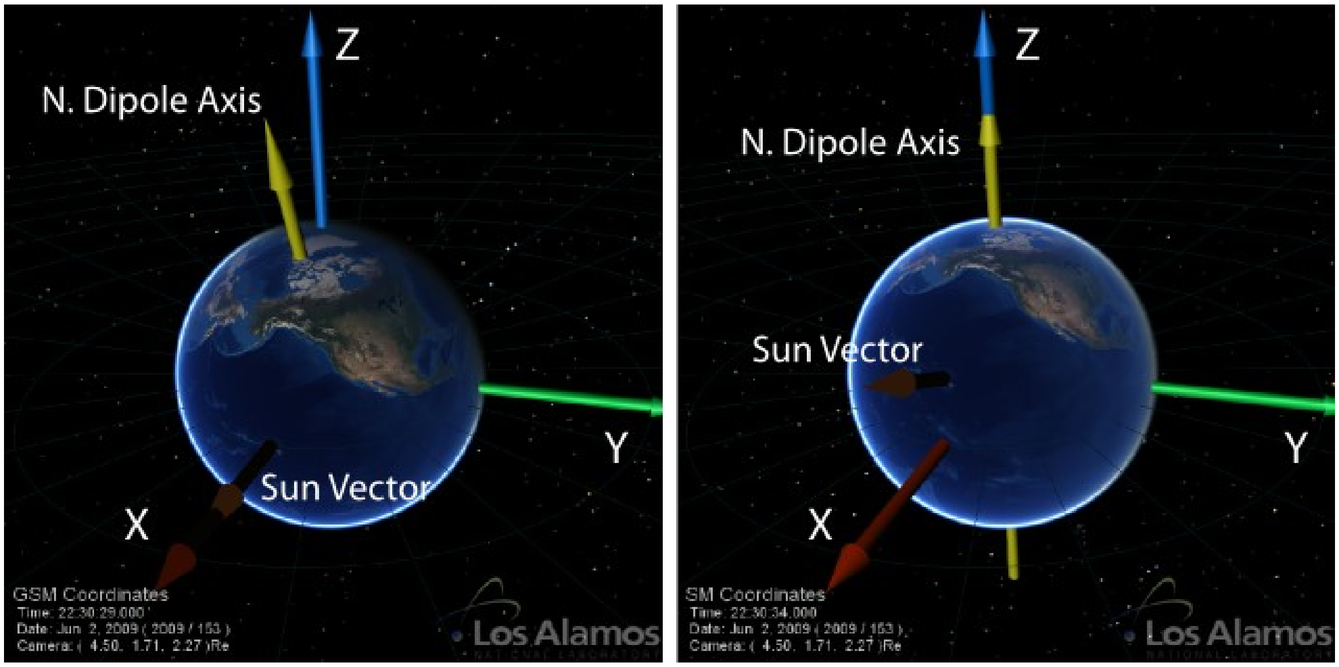


Figure 8: Three-dimensional view illustrating the difference between the GSM (left) and SM (right) coordinate systems. In both systems, the dipole axis and the sun vector are contrained to lie in the X-Z plane. The difference is that GSM has X along the Sun vector whereas SM has Z along the north dipole direction. The angle between the dipole axis and the GSM Z-axis (or the angle between the SM X-axis and the sun vector) is called the geo-dipole tilt angle, Ψ. The two systems are related by a simple rotation about the Y-axis which is common to both systems.

16.1 Centered Dipole

From *Chapman and Bartels [1962]*, the northern geocentric longitude and co-latitude, $(\phi_{0,} \theta_0)$ of the dipole axis is given in terms of the first 3 time-dependent IGRF coefficients as;

$$\begin{aligned} \phi_0 &= \tan^{-1}(h_1^1 / g_1^1) \\ \theta_0 &= 180° - \cos^{-1}(g_1^0 \left[(g_1^0)^2 + (g_1^1)^2 + (h_1^1)^2 \right]^{-1/2}) \end{aligned} \qquad (12)$$

Note that for the Earth the dipole axis actually points southward – i.e., it sticks out of the southern polar region.. So $(\phi_{0,} \theta_0)$ actually represents the "anti-dipole" direction, or where the tail of the dipole vector sticks out of the northern polar region. In Figure 8, the yellow vector labeled "N. Dipole Axis" is actually the anti-dipole direction.

16.2 Dipole Offset

From *Akasofu and Chapman [1972]* (and *Chapman and Bartels [1962]* – but note that this reference has typos in equations 22-24*)*, The center of the offset dipole is determined from the first 8 IGRF coefficients and is located at $(x_{0,} y_{0,} z_0)$ where;

$$\begin{aligned} x_0 &= a(L_x - g_1^0 E)/(3\mathrm{H}_0^2) \\ y_0 &= a(L_y - g_1^1 E)/(3\mathrm{H}_0^2) \\ z_0 &= a(L_z - h_1^1 E)/(3\mathrm{H}_0^2) \end{aligned} \tag{13}$$

and;

$$\begin{aligned} H_0^2 &= (g_1^0)^2 + (g_1^1)^2 + (h_1^1)^2 \\ L_x &= -g_1^1 g_2^0 + \sqrt{3}(g_1^0 g_2^1 + g_1^1 g_2^2 + h_1^1 h_2^2) \\ L_y &= -h_1^1 g_2^0 + \sqrt{3}(g_1^0 h_2^1 - h_1^1 g_2^2 + g_1^1 h_2^2) \\ L_z &= 2\mathrm{g}_1^0 g_2^0 + \sqrt{3}(g_1^1 g_2^1 + h_1^1 h_2^1) \\ E &= (L_0 g_1^0 + L_1 g_1^1 + L_2 h_1^1)/(4\mathrm{H}_0^2) \end{aligned} \tag{14}$$

and a is the radius of the Earth (6378.1370 km in the WGS84 model). The position $(x_{0,} y_{0,} z_0)$ is in geocentric geographic coordinates and is found to be displaced by several hundred kilometers from the center of the Earth towards the western Pacific Ocean (about 22 degrees) north of the equator. The offset of the dipole from the center of the Earth is therefore not negligible. The displacement of the dipole field toward the western Pacific causes a depression of the low altitude field on the other side of the Earth which gives rise to the South Atlantic Anomaly (SSA).

17 Two Line Elements and SGP4

We also include utility routines to read Two Line Elements (TLEs) and evaluate object locations and velocities using the SGP4 orbit propagator. As we discussed earlier, the SGP4 propagator outputs in a relatively obscure coordinate system called TEME (True Equator, Mean Equinox), so you need to convert the TEME output to a more standard coordinate system. See *Vallado* [2007] and *Vallado et al.* [2006] for details on the TEME coordinate system.

`int LgmSgp_SGP4_Init( _SgpInfo *s, _SgpTLE *t )`

Initialization routine for SGP4. The second argument is a pointer to a TLE structure.

`int LgmSgp_SGP4( double TSINCE, _SgpInfo *s );`

First argument is time (in minutes) since the TLE epoch.

`int LgmSgp_TleChecksum( char *Line )`

Verifies TLE checksum.

```
int LgmSgp_ReadTlesFromFile( char *Filename, int *nTLEs,
                             _SgpTLE *TLEs, int Verbosity )
```

Reads a list of TLEs from a file. User gives the filename that the TLEs are stored in and an array of TLE structures . As valid TLEs are read they are decoded and dumped into the TLEs array. The number of valid TLEs found is return in nTLEs. The user must initialize the nTLEs variable to zero before this routine is called for the first time. Subsequently, it can be called any number of numbers (so long as you don't overflow the TLEs array) which is useful for accumulating TLEs that are stored in separate files.

int LgmSgp_ReadTlesFromStrings(char *Line0, char *Line1, char *Line2, int *nTLEs, _SgpTLE *TLEs, int Verbosity)

Reads a single TLE from the three user-provided strings Line0, Line1, Line2.. User also gives an array of TLE structures . If the strings constitute a valid TLE they are decoded and placed in the array of TLE structures and nTLEs variable will be incremented. The user must initialize the nTLEs variable to zero before this routine is called for the first time. It can be called any number of times after that (so long as you don't overflow the TLEs array). This is useful for accumulating TLEs that you somehow have in the form of strings instead of stored in files (e.g. as the result of already having read them in with some other piece of code).

An example showing how to use the **LgmSgp_ReadTlesFromStrings( )** routine is given below;

```
#include <stdlib.h>
#include <stdio.h>
#include <Lgm_CTrans.h>
#include <Lgm_Sgp.h>

/*
 * This example shows how to compute positions of objects from TLEs
 * (Two Line Elements).
 */

int main( int argc, char *argv[] ){

    int         Year, Month, Day;
    int         StartYear, StartMonth, StartDay, StartDoy;
    int         EndYear, EndMonth, EndDay, EndDoy;
    long int    Date, StartDate, EndDate;
    double      UT, tsince, JD, StartUT, EndUT, StartJD, EndJD, JDinc;
    Lgm_CTrans  *c = Lgm_init_ctrans( 0 );
    Lgm_Vector  Ugse, Uteme;
    int         nTLEs;
    char        Line0[100], Line1[100], Line2[100], *ptr;
    char        *InputFile  = "ssa_swx_tle_input.txt";
    char        *OutputFile = "ssa_swx_tle_output.txt";
    FILE        *fp;

    /*
     * Open input file and extract:
     *   1) the 3 TLE lines
     *   2) Start Date and Time
     *   3) End Date and Time
     */
    if ( (fp = fopen( InputFile, "r" )) != NULL ) {
        fgets( Line0, 99, fp );
        fgets( Line1, 99, fp );
        fgets( Line2, 99, fp );
        fscanf( fp, "%ld %lf", &StartDate, &StartUT );
        fscanf( fp, "%ld %lf", &EndDate, &EndUT );
    } else {
        printf( "Couldnt open file %s for reading\n", InputFile );
        exit( 1 );
    }

    /*
     * Remove any extraneous newline and/or linefeeds at the end of the strings.
     */
    if ( (ptr = strstr(Line0, "\n")) != NULL ) *ptr = '\0';
    if ( (ptr = strstr(Line0, "\r")) != NULL ) *ptr = '\0';
    if ( (ptr = strstr(Line1, "\n")) != NULL ) *ptr = '\0';
    if ( (ptr = strstr(Line1, "\r")) != NULL ) *ptr = '\0';
    if ( (ptr = strstr(Line2, "\n")) != NULL ) *ptr = '\0';
    if ( (ptr = strstr(Line2, "\r")) != NULL ) *ptr = '\0';

    /*
     * Alloc some memory for the SgpInfo structure and the TLEs array (here we
     * only have a single element in the array)
     */
    _SgpInfo *s    = (_SgpInfo *)calloc( 1, sizeof(_SgpInfo) );
    _SgpTLE *TLEs = (_SgpTLE *)calloc( 1, sizeof(_SgpTLE) );


    /*
     * Read in TLEs from the Line0, Line1 and Line2 strings. nTLEs must be
     * initialized to zero by the user.
     */
    nTLEs = 0;
    LgmSgp_ReadTlesFromStrings( Line0, Line1, Line2, &nTLEs, TLEs, 1 );

    /*
     * All the TLEs have their own epoch times in them. And the propagator
     * (sgp4) uses the "time since (in minutes)". So for a given time of
     * interest, we need to compute the tsince needed. Convert Start/End Dates
     * to Julian dates -- they are easier to loop over contiguously.
     */
    Lgm_Doy( StartDate, &StartYear, &StartMonth, &StartDay, &StartDoy );
    StartJD = Lgm_JD( StartYear, StartMonth, StartDay, StartUT, c );

    Lgm_Doy( EndDate, &EndYear, &EndMonth, &EndDay, &EndDoy );
    EndJD = Lgm_JD( EndYear, EndMonth, EndDay, EndUT, c );

    JDinc = 5.0/1440.0; // 5-min increment

    if ( (fp = fopen( OutputFile, "w" )) == NULL ) {
        printf( "Couldnt open file %s for writing\n", OutputFile );
        exit( 1 );
    }

    // init SGP4
    LgmSgp_SGP4_Init( s, &TLEs[0] );
    printf("%%%s\n", TLEs[0].Line0 );
    printf("%%%s\n", TLEs[0].Line1 );
    printf("%%%s\n", TLEs[0].Line2 );
    fprintf(fp, "%%%s\n", TLEs[0].Line0 );
    fprintf(fp, "%%%s\n", TLEs[0].Line1 );
    fprintf(fp, "%%%s\n", TLEs[0].Line2 );

    // loop over specified time range
    for ( JD = StartJD; JD <= EndJD; JD += JDinc ) {

        // Convert the current JD back to Date/UT etc..
        Lgm_jd_to_ymdh ( JD, &Date, &Year, &Month, &Day, &UT );

        // Set up the trans matrices
        Lgm_Set_Coord_Transforms( Date, UT, c );

        // "time since" in minutes (thats what SGP4 wants)
        tsince = (JD - TLEs[0].JD)*1440.0;
        // Call SGP4. Coords are in TEME.
        LgmSgp_SGP4( tsince, s );
        Uteme.x = s->X/WGS84_A; Uteme.y = s->Y/WGS84_A; Uteme.z = s->Z/WGS84_A;

        // Example of converting TEME->GSE coords.
        Lgm_Convert_Coords( &Uteme, &Ugse, TEME_TO_GSE, c );

        // Dump result to file
        printf("%8ld %.10lf %10.6lf %10.6lf %10.6lf\n", Date, UT, Ugse.x, Ugse.y, Ugse.z  );
        fprintf(fp, "%8ld %.10lf %10.6lf %10.6lf %10.6lf\n", Date, UT, Ugse.x, Ugse.y, Ugse.z  );

    }

    fclose(fp);
    return(0);
}
```

With the following input file;

```
OPS 5111 (NAVSTAR 1)
1 10684U 78020A   09054.00588353  .00000074  00000-0  10000-3 0  9676
2 10684 063.0801 245.2342 0065866 177.7286 182.3286 01.98072046211191
20090501 0.0
20090501 0.5
```

we get this output;

```
%OPS 5111 (NAVSTAR 1)
%1 10684U 78020A   09054.00588353  .00000074  00000-0  10000-3 0  9676
%2 10684 063.0801 245.2342 0065866 177.7286 182.3286 01.98072046211191
20090501 0.0000000000  -0.506399   0.972731  -4.046189
20090501 0.0833333284  -0.680838   0.923741  -4.033272
20090501 0.1666666567  -0.854007   0.873040  -4.012795
20090501 0.2499999851  -1.025582   0.820724  -3.984801
20090501 0.3333333135  -1.195244   0.766893  -3.949350
20090501 0.4166666418  -1.362678   0.711649  -3.906515
20090501 0.4999999702  -1.527573   0.655095  -3.856382
```

Here is a slightly different example showing how to use **`LgmSgp_ReadTlesFromFile( )`**.

```
#include <stdlib.h>
#include <stdio.h>
#include <Lgm_CTrans.h>
#include <Lgm_Sgp.h>

/*
 * This example shows how to compute positions of objects from TLEs (Two Line
 * Elements). In this example, we read multiple TLEs from a file
 */
int main( int argc, char *argv[] ){

    int         Year, Month, Day;
    int         StartYear, StartMonth, StartDay, StartDoy;
    int         EndYear, EndMonth, EndDay, EndDoy;
    long int    Date, StartDate, EndDate;
    double      UT, tsince, JD, StartUT, EndUT, StartJD, EndJD, JDinc;
    Lgm_CTrans  *c = Lgm_init_ctrans( 0 );
    Lgm_Vector  Ugse, Uteme;
    int         nTLEs, i;
    char        TleFile[512];
    char        *InputFile  = "input.txt";
    char        *OutputFile = "output.txt";
    FILE        *fp;

    /*
     * Open input file and extract:
     *   1) name of a file containing TLEs
     *   2) Start Date and Time
     *   3) End Date and Time
     */
    if ( (fp = fopen( InputFile, "r" )) != NULL ) {
        fscanf( fp, "%s", TleFile );
        fscanf( fp, "%ld %lf", &StartDate, &StartUT );
        fscanf( fp, "%ld %lf", &EndDate, &EndUT );
    } else {
        printf( "Couldnt open file %s for reading\n", InputFile );
        exit( 1 );
    }

    /*
     * Alloc some memory for the SgpInfo structure and the TLEs array (here we
     * alloc 20000 elements -- in more robust applications you'd want to
     * dynamically alloc and realloc as you go so you dont waste memory. )
     */
    _SgpInfo *s   = (_SgpInfo *)calloc( 1, sizeof(_SgpInfo) );
    _SgpTLE *TLEs = (_SgpTLE *)calloc( 20000, sizeof(_SgpTLE) );


    /*
     * Read in TLEs.
     */
    nTLEs = 0;
    LgmSgp_ReadTlesFromFile( TleFile, &nTLEs, TLEs, 1 );
    printf("Found %d Valid TLEs in file %s\n", nTLEs, TleFile );

    /*
     * All the TLEs have their own epoch times in them. And the propagator
     * (sgp4) uses the "time since (in minutes)". So for a given time of
     * interest, we need to compute the tsince needed. Convert Start/End Dates
     * to Julian dates -- they are easier to loop over contiguously.
     */
    Lgm_Doy( StartDate, &StartYear, &StartMonth, &StartDay, &StartDoy );
    StartJD = Lgm_JD( StartYear, StartMonth, StartDay, StartUT, c );

    Lgm_Doy( EndDate, &EndYear, &EndMonth, &EndDay, &EndDoy );
    EndJD = Lgm_JD( EndYear, EndMonth, EndDay, EndUT, c );

    JDinc = 5.0/1440.0; // 5-min increment

    if ( (fp = fopen( OutputFile, "w" )) == NULL ) {
        printf( "Couldnt open file %s for writing\n", OutputFile );
        exit( 1 );
    }

    // loop through each TLE found
    for (i=0; i<nTLEs; i++){

        // init SGP4
        LgmSgp_SGP4_Init( s, &TLEs[i] );
        fprintf(fp, "%%%s\n", TLEs[i].Line0 );
        fprintf(fp, "%%%s\n", TLEs[i].Line1 );
        fprintf(fp, "%%%s\n", TLEs[i].Line2 );

        // loop over specified time range
        for ( JD = StartJD; JD <= EndJD; JD += JDinc ) {

            // Convert the current JD back to Date/UT etc..
            Lgm_jd_to_ymdh ( JD, &Date, &Year, &Month, &Day, &UT );
            // Set up the trans matrices
            Lgm_Set_Coord_Transforms( Date, UT, c );

            // "time since" in minutes (thats what SGP4 wants)
            tsince = (JD - TLEs[i].JD)*1440.0;

            // Call SGP4. Coords are in TEME.
            LgmSgp_SGP4( tsince, s );
            Uteme.x = s->X/WGS84_A; Uteme.y = s->Y/WGS84_A; Uteme.z = s->Z/WGS84_A;

            // Example of converting TEME->GSM coords.
            Lgm_Convert_Coords( &Uteme, &Ugse, TEME_TO_GSM, c );

            // Dump result to file
            fprintf(fp, "%8ld %.10lf %10.6lf %10.6lf %10.6lf\n", Date, UT, Ugse.x, Ugse.y, Ugse.z  );

        }
    }

    fclose(fp);
    return(0);
}
```

With the **input.txt** input file;

```
TLEs.txt
20080101 0.0
20080101 0.5
```

and the **TLE.txt** file containing;

```
SAMPEX
1 22012U 92038A   09132.11471116  .00002346  00000-0  57742-4 0  8988
2 22012 081.6385 344.6301 0048130 039.5837 320.8901 15.38845473928866
TOPEX/POSEIDON
1 22076U 92052A   09131.94192993 -.00000038 +00000-0 +10000-3 0 03535
2 22076 066.0320 145.9983 0007930 265.3380 094.6730 12.80947256783608
TDRS 6
1 22314U 93003B   09131.51819318 +.00000095 +00000-0 +10000-3 0 04709
2 22314 009.1252 059.1227 0006591 337.0883 208.5930 01.00279826059817
FAST
1 24285U 96049A   09133.18444126  .00000315  00000-0  28335-4 0  7739
2 24285 082.9720 076.2454 1997379 233.6616 106.3248 11.26193295514598
CXO
1 25867U 99040B   09132.12500000 +.00000298 +00000-0 +10000-3 0 03226
2 25867 065.2923 070.4297 7459513 276.0695 359.6709 00.37806577004876
TELSTAR 12 (ORION 2)
1 25949U 99059A   09131.92834822 -.00000117  00000-0  10000-3 0  6703
2 25949 000.0657 085.7680 0003031 320.0986 143.1654 01.00271979 35017
NOAA 16
1 26536U 00055A   09132.30843460 -.00000223  00000-0 -95270-4 0  3984
2 26536 099.1751 135.1081 0010892 027.3979 332.7779 14.12516453445230
IBEX
1 33401U 08051A   08322.28291206 -.00000938 +00000-0 +10000-3 0 00053
2 33401 014.0406 072.5742 8843496 056.3560 009.4607 00.13253372000071
HST
1 20580U 90037B   09132.93383239  .00000388  00000-0  17126-4 0  3915
2 20580  28.4692 332.3792 0003661 283.8491  76.1691 15.00483233844553
POLAR
1 23802U 96013A   09053.22528995 +.00000077 +00000-0 +10000-3 0 06583
2 23802 085.1933 323.6503 6021382 081.4626 338.5274 01.29847079062745
```

we get the following in **output.txt;**

```
%SAMPEX
%1 22012U 92038A   09132.11471116  .00002346  00000-0  57742-4 0  8988
```

```
%2 22012 081.6385 344.6301 0048130 039.5837 320.8901 15.38845473928866
20080101 0.0000000000   0.143999   1.014648  -0.328265
20080101 0.0833333284   0.211788   0.841993  -0.633529
20080101 0.1666666567   0.256131   0.576017  -0.868461
20080101 0.2499999851   0.272048   0.245950  -1.006662
20080101 0.3333333135   0.257654  -0.111674  -1.032238
20080101 0.4166666418   0.214411  -0.456886  -0.941688
20080101 0.4999999702   0.147034  -0.750644  -0.744568
%TOPEX/POSEIDON
%1 22076U 92052A   09131.94192993 -.00000038 +00000-0 +10000-3 0 03535
%2 22076 066.0320 145.9983 0007930 265.3380 094.6730 12.80947256783608
20080101 0.0000000000  -1.133527  -0.336627   0.252584
20080101 0.0833333284  -0.980081  -0.487465   0.512315
20080101 0.1666666567  -0.750484  -0.599918   0.732428
20080101 0.2499999851  -0.462544  -0.665296   0.895743
20080101 0.3333333135  -0.138602  -0.678612   0.989452
20080101 0.4166666418   0.196183  -0.638969   1.006116
20080101 0.4999999702   0.515780  -0.549623   0.944264
%TDRS 6
%1 22314U 93003B   09131.51819318 +.00000095 +00000-0 +10000-3 0 04709
%2 22314 009.1252 059.1227 0006591 337.0883 208.5930 01.00279826059817
20080101 0.0000000000  -3.738627   5.380923  -0.867880
20080101 0.0833333284  -3.847816   5.295524  -0.914899
20080101 0.1666666567  -3.955173   5.207428  -0.961791
20080101 0.2499999851  -4.060649   5.116665  -1.008504
20080101 0.3333333135  -4.164193   5.023266  -1.054985
20080101 0.4166666418  -4.265755   4.927266  -1.101180
20080101 0.4999999702  -4.365287   4.828699  -1.147031
%FAST
%1 24285U 96049A   09133.18444126  .00000315  00000-0  28335-4 0  7739
%2 24285 082.9720 076.2454 1997379 233.6616 106.3248 11.26193295514598
20080101 0.0000000000   1.079558   0.521648  -1.017890
20080101 0.0833333284   1.156784   0.655898  -0.800741
20080101 0.1666666567   1.191367   0.766321  -0.554402
20080101 0.2499999851   1.179266   0.847183  -0.286681
20080101 0.3333333135   1.116990   0.892572  -0.007106
20080101 0.4166666418   1.002018   0.896504   0.272462
20080101 0.4999999702   0.833594   0.853288   0.537143
%CXO
%1 25867U 99040B   09132.12500000 +.00000298 +00000-0 +10000-3 0 03226
%2 25867 065.2923 070.4297 7459513 276.0695 359.6709 00.37806577004876
20080101 0.0000000000   2.580285   2.450691  -2.860113
20080101 0.0833333284   2.381872   2.511926  -2.941909
20080101 0.1666666567   2.179950   2.569049  -3.019715
20080101 0.2499999851   1.974778   2.621917  -3.093370
20080101 0.3333333135   1.766635   2.670406  -3.162726
20080101 0.4166666418   1.555814   2.714408  -3.227650
20080101 0.4999999702   1.342629   2.753838  -3.288027
%TELSTAR 12 (ORION 2)
%1 25949U 99059A   09131.92834822 -.00000117  00000-0  10000-3 0  6703
%2 25949 000.0657 085.7680 0003031 320.0986 143.1654 01.00271979 35017
20080101 0.0000000000  -1.315109   6.451446   0.517066
20080101 0.0833333284  -1.444521   6.428652   0.451310
20080101 0.1666666567  -1.573246   6.402703   0.384796
20080101 0.2499999851  -1.701220   6.373591   0.317571
20080101 0.3333333135  -1.828384   6.341313   0.249681
20080101 0.4166666418  -1.954676   6.305863   0.181177
20080101 0.4999999702  -2.080036   6.267240   0.112110
%NOAA 16
%1 26536U 00055A   09132.30843460 -.00000223  00000-0 -95270-4 0  3984
%2 26536 099.1751 135.1081 0010892 027.3979 332.7779 14.12516453445230
20080101 0.0000000000   0.512716   0.889736  -0.479260
20080101 0.0833333284   0.519693   0.995659  -0.148909
20080101 0.1666666567   0.477626   1.008327   0.195375
20080101 0.2499999851   0.390424   0.926489   0.521223
20080101 0.3333333135   0.266321   0.757849   0.797982
20080101 0.4166666418   0.117078   0.518319   0.999701
20080101 0.4999999702  -0.043171   0.230439   1.107615
```

```
%IBEX
%1 33401U 08051A   08322.28291206 -.00000938 +00000-0 +10000-3 0 00053
%2 33401 014.0406 072.5742 8843496 056.3560 009.4607 00.13253372000071
20080101 0.0000000000  42.593621  13.439428  17.909539
20080101 0.0833333284  42.581696  13.465956  17.892803
20080101 0.1666666567  42.569720  13.494103  17.874760
20080101 0.2499999851  42.557693  13.523865  17.855403
20080101 0.3333333135  42.545615  13.555236  17.834724
20080101 0.4166666418  42.533485  13.588207  17.812717
20080101 0.4999999702  42.521303  13.622771  17.789374
%HST
%1 20580U 90037B   09132.93383239  .00000388  00000-0  17126-4 0  3915
%2 20580  28.4692 332.3792 0003661 283.8491  76.1691 15.00483233844553
20080101 0.0000000000  -0.821100  -0.655486  -0.284594
20080101 0.0833333284  -0.547993  -0.857265  -0.387666
20080101 0.1666666567  -0.216646  -0.968149  -0.449149
20080101 0.2499999851   0.137693  -0.976364  -0.462674
20080101 0.3333333135   0.477392  -0.881016  -0.427041
20080101 0.4166666418   0.766428  -0.692163  -0.346299
20080101 0.4999999702   0.974168  -0.429727  -0.229279
%POLAR
%1 23802U 96013A   09053.22528995 +.00000077 +00000-0 +10000-3 0 06583
%2 23802 085.1933 323.6503 6021382 081.4626 338.5274 01.29847079062745
20080101 0.0000000000   3.799619   3.366695   0.035709
20080101 0.0833333284   3.698618   3.318238   0.165130
20080101 0.1666666567   3.593452   3.266339   0.294161
20080101 0.2499999851   3.483969   3.210846   0.422659
20080101 0.3333333135   3.370006   3.151592   0.550457
20080101 0.4166666418   3.251395   3.088392   0.677366
20080101 0.4999999702   3.127959   3.021045   0.803169
```

Note that in the first example the coordinates were in units of Earth radii in the GSE system. In the last example, they were in GSM. Also note that for these examples, I am are not concerned with whether or not the TLEs are actually up-to-date – I just grabbed some old ones out of a file that was lying around. In real applications, you will need to use TLEs that are suitable for the time period you are interested in and these can be obtained from sites like http://www.space-track.org .

18 Internal Magnetic Field Models

Describe IGRF/DGRF models and time-dependent determination of CD, ED poles. The cenetered dipole and the offset dipole parameters are determined from time dependent IGRF coefficients. Etc...

```
/*
 *  B_internal
 *
 *  Function Prototypes for internal models
 */
int Lgm_B_igrf(Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo *);
int Lgm_B_cdip(Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo *);
int Lgm_B_edip(Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo *);
```

19 External Magnetic Field Models

```
void Lgm_FreeMagInfo( Lgm_MagModelInfo  *Info );
void Lgm_CopyMagInfo( Lgm_MagModelInfo *s, Lgm_MagModelInfo *t );


// T87
int Lgm_B1_T87( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_B2_T87( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_B3_T87( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_B_T87( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );


//  T89
int Lgm_BM_T89( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_BT_T89( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_BRC_T89( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_BC_T89( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int Lgm_B_T89( Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );

//  T96MOD
int Lgm_B_T96MOD_MGH( Lgm_Vector *v, Lgm_Vector *B, Lgm_MagModelInfo *Info );
void lgm_field_t96mod_mgh_(double *, double *, int *IYEAR,int *IDAY,int *IH,int
*IM,double *SEC,double *X,double *Y,double *Z,double *BX,double *BY,double *BZ);
void lgm_field_t96mod_mgh__(double *PARMOD, double *AMDF, int *IYEAR,int *IDAY,int
*IH,int *IM,double *SEC,double *X,double *Y,double *Z,double *BX,double *BY,double
*BZ);
void lgm_field_t96mod_( int *, int *, int *, int *, double *, double *, double *,
double *, double *, double *, double * );


//  TS04
int Lgm_B_TS04( Lgm_Vector *v, Lgm_Vector *B, Lgm_MagModelInfo *Info );
void Lgm_ComputeW( double W[], int i, double Nk[], double Vk[],
                                                     double Bsk[], int nk );
void Lgm_T04_s( int IOPT, double *PARMOD, double PS, double SINPS,double COSPS,
             double X, double Y, double Z, double *BX, double *BY, double *BZ);


//  Computing B from scattered data -- (e.g. an irregular mesh)
int Lgm_B_FromScatteredData( Lgm_Vector *v, Lgm_Vector *B, Lgm_MagModelInfo
*Info );


// Simplified Mead field.
int Lgm_SimplifiedMead(Lgm_Vector *v, Lgm_Vector *B, Lgm_MagModelInfo *Info);

/*
 * Getter/Setter functions. Add more? (You can always change
 * anything in the    structures themselves.)
 */
void Lgm_MagModelInfo_Set_Psw( double Psw, Lgm_MagModelInfo *m );
void Lgm_MagModelInfo_Set_Kp( double Kp, Lgm_MagModelInfo *m );
void Lgm_Set_Octree_kNN_InterpMethod( Lgm_MagModelInfo *m, int Method );
void Lgm_Set_Octree_kNN_k( Lgm_MagModelInfo *m, int k );
void Lgm_Set_Octree_kNN_MaxDist( Lgm_MagModelInfo *m, double MaxDist );
void Lgm_Set_Open_Limits( Lgm_MagModelInfo *m, double xmin, double xmax, double
ymin, double ymax, double zmin, double zmax );
```

20 Magnetic Field Tracing Routines

The LanlGeoMag library has a number of magnetic field tracing utilities that allow the user to determine a field line's: topology, footpoints, minimum-B point(s), particle mirror points, etc. Here we describe some of these rotuines and give some simple examples of their use.

```
int Lgm_Trace( Lgm_Vector *u, Lgm_Vector *v1, Lgm_Vector *v2, Lgm_Vector *v3,
                      double Height, double TOL1, double TOL2, Lgm_MagModelInfo *Info )
```

For a given position (u) this routine attempts to find the northern (v1) and southern (v2) foot-points of the magnetic field line that passes through u. In addition, it also tries to find the location of the minimum $|\vec{B}|$ point (v3) along the field line. All position vectors are in GSM in units of Earth radii. The altitude above the surface of the Earth (in km) used to define the "foot-point" must be given as Height. A typical value to use for Height is 120.0 km. Note that Height is the altitude above the surface of the WGS84 spheroid.

The return value can be used to determine what type of magnetic field line was found and therefore which of the returned positions (v1, v2, v3) are valid. The return codes and their corresponding meaning are listed below:

Return Code	Meaning
LGM_OPEN_IMF	field line is open at both ends (i.e. an IMF FL) (no points valid).
LGM_CLOSED	field line is closed (v1, v2, v3 all valid).
LGM_OPEN_N_LOBE	field line is northern lobe FL (v1 valid).
LGM_OPEN_S_LOBE	field line is southern lobe FL (v2 valid).
LGM_INSIDE_EARTH	initial point is inside Earth (no points valid).
LGM_TARGET_HEIGHT_UNREACHABLE	Field line never gets up to target height (i.e. it always remains below it). (no points valid).

The user should examine the value of the return code before using the v1, v2, and v3 values.

Lgm_Trace() also sets some other values in the Info structure that is passed to it, as listed below:

Variable set in MagModelInfo Structure	Variable Type	Comment
Info->Pmin	**Lgm_Vector**	Pmin holds the position of Bmin (i.e. it is the same as the value of v3 that is returned through the argument list).

`Info->smin`	**double**	Distance along field line in units of Re from the southern footpoint to Pmin
`Info->Bvecmin`	**Lgm_Vector**	Magnetic field vector at Pmin
`Info->Bmin`	**double**	Magnitude of Bvecmin
`Info->d2B_ds2`	**double**	Curvature of field at Pmin, $\partial^2 B / \partial s^2$
`Info->Sb0`	**double**	Sb integral for nearly equatorially mirroring particles, Sb0 ($S_{b0} = \pi \sqrt{(2 B_{min} / (\partial^2 B / \partial s^2))}$)

```
int Lgm_TraceToEarth( Lgm_Vector *u, Lgm_Vector *v, double TargetHeight,
                             double sgn, double tol, Lgm_MagModelInfo *Info )
```

Starting at position u, this routine attempts to trace to the Earth along the field line threading u. If successful, the footpoint location is returned in v. User must specify a direction to trace (sgn must be 1.0 or -1.0) and the height above the Earth's surface to stop – i.e. the goal is to find the foot-point of the field line for the given direction. If both foot-points are desired, user must call this routine twice (once for each direction). Lgm_Trace() is a convenience routine that does that in a single call and also finds some other quantities as well.

```
int Lgm_TraceToMinBSurf( Lgm_Vector *u, Lgm_Vector *v, double Htry,
                                     double tol, Lgm_MagModelInfo *Info );
```

Starting at position u, this routine attempts to trace to the minimum-B point along the field line threading u. If successful, the minimum-B location is returned in v. The user specifies an initial stepsize to use in Htry and the tolerance to use for convergence.

```
int Lgm_TraceToSMEquat(  Lgm_Vector *u, Lgm_Vector *v, double tol, Lgm_MagModelInfo *Info );
```

Starting at position u, this routine attempts to trace to the point along the field line threading u that intersects the Solar-Magnetic equatorial plane. If successful, the the intersection point is returned in v. The user specifies the tolerance to use for convergence.

```
int Lgm_TraceToMinRdotB( Lgm_Vector *u, Lgm_Vector *v, double tol, Lgm_MagModelInfo *Info );
```

Starting at position u, this routine attempts to trace to the point along the field line threading u where $\vec{r} \cdot \vec{B}$ reverses sign. If successful, the point is returned in v. The user specifies the tolerance to use for convergence.

```
int Lgm_TraceToMirrorPoint( Lgm_Vector *u, Lgm_Vector *v, double *Sm, double H0,
                             double Bm, double sgn, double tol, Lgm_MagModelInfo *Info );
```

Starting at position u, this routine attempts to trace to the point along the field line (threading u) where the magnetic field strength is equal to a given value in nT (Bm). If successful, the point is returned in v. The user specifies the tolerance (tol) to use for convergence and the direction to trace (sgn). The value of Sm will be returned as the length along the field line from the initial point to the mirror point found (these values are useful for setting up limits of integration when computing the integral invariant, I or the Sb integral, etc.) The user must also input the height above the Earth's surface that is used to define the loss cone. If particles mirror below this altitude, they will be considered to be inside the losscone.

```
int Lgm_TraceLine(  Lgm_Vector *, Lgm_Vector *, double, double, double, Lgm_MagModelInfo * );
int Lgm_TraceLine2(  Lgm_Vector *, Lgm_Vector *, double, double, double, Lgm_MagModelInfo * );
void AddNewPoint( double s, double B, Lgm_Vector P, Lgm_MagModelInfo *Info );
void InitSpline( Lgm_MagModelInfo *Info );
```

These routines are used to trace field lines out and store them in a 1-D array. The idea is to be able to use these arrays in interpolation functions to avoid having to re-computed the field all the time. This can save a lot of computation when using complicated/expensive B-field models. Since the resulting arrays likely will not have the various critical points explicitly represented in them, there are routines that allow for the addition of extra points. Etc..

```
void Lgm_ModMid( Lgm_Vector *, Lgm_Vector *, double, int, double, int (*Mag)(Lgm_Vector *, Lgm_Vector
*, Lgm_MagModelInfo *), Lgm_MagModelInfo * );
void Lgm_RatFunExt( int, double, Lgm_Vector *, Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo * );
int  Lgm_MagStep( Lgm_Vector *, Lgm_Vector *, double, double *, double *, double, double, double *, int
*, int (*Mag)(Lgm_Vector *, Lgm_Vector *, Lgm_MagModelInfo *), Lgm_MagModelInfo * );
```

21 Additional Magnetic Coordinates

CGM, APEX, etc... (These may be in stand-alone apps that were built ontop of an earlier version of the library). Probably should pull them into the library...

22 Adiabatic Invariants and Other Things

I, K, L*, Hilton L, McIlwain L, AlphaOfK, etc... There are probably more of these tucked away somewhere else too...

```
/*
 * routines/functions for field integrals, invariants, etc.
 */
double  Iinv( Lgm_MagModelInfo *fInfo );

double  I_integrand( double s, _qpInfo *qpInfo );
double  Iinv_interped( Lgm_MagModelInfo *fInfo );

double  I_integrand_interped( double s, _qpInfo *qpInfo );

double  SbIntegral( Lgm_MagModelInfo *fInfo );

double  Sb_integrand( double s, _qpInfo *qpInfo );

double  SbIntegral_interped( Lgm_MagModelInfo *fInfo );

double  Sb_integrand_interped( double s, _qpInfo *qpInfo );

void    ratint( double *xa, double *ya, int n, double x, double *y, double *dy );

void    polint(double *xa, double *ya, int n, double x, double *y, double *dy);

void    Interp( double xa[],  double ya[], long int n, double x, double *y);

void    Interp2( double xa[],  double ya[], long int n, double x, double *y);

double  LFromIBmM_Hilton( double I, double Bm, double M );

double  IFromLBmM_Hilton( double L, double Bm, double M );

double  LFromIBmM_McIlwain( double I, double Bm, double M );

double  IFromLBmM_McIlwain( double L, double Bm, double M );

double  BofS( double s, Lgm_MagModelInfo *Info );

int     SofBm( double Bm, double *ss, double *sn, Lgm_MagModelInfo *Info );
```

23 Drift Shell Routines

There are a bunch of these that have already been ingested into the library. Other routines should be as well (e.g. Drift velocity calcs, etc.)

24 Vector Utilities

Following vector routines are defined in <LgmVec.h>

```
void    Lgm_CrossProduct(Lgm_Vector *, Lgm_Vector *, Lgm_Vector *);

double  Lgm_DotProduct(Lgm_Vector *, Lgm_Vector *);

double  Lgm_NormalizeVector(Lgm_Vector *);

void    Lgm_ScaleVector(Lgm_Vector *, double);

double  Lgm_Magnitude( Lgm_Vector *);
```

```
void    Lgm_ForceMagnitude(Lgm_Vector *, double);

void    Lgm_MatTimesVec(double A[3][3], Lgm_Vector *, Lgm_Vector *);

void    Lgm_MatTimesMat( double A[3][3], double B[3][3], double R[3][3] );

void    Lgm_VecSub(Lgm_Vector *c, Lgm_Vector *a, Lgm_Vector *b );

void    Lgm_VecAdd(Lgm_Vector *c, Lgm_Vector *a, Lgm_Vector *b );

double  Lgm_VecDiffMag(Lgm_Vector *a, Lgm_Vector *b );

void    Lgm_Transpose( double A[3][3], double B[3][3] );
```

25 Quaternion Utilities

Following Quaternion routines are defined in <LgmQuat.h>

```
double  Lgm_NormalizeQuat( double *Q );

double  Lgm_MatrixTrace( double A[3][3] );

void    Lgm_MatrixToQuat( double A[3][3], double *Q );

void    Lgm_Quat_To_Matrix( double Q[4], double A[3][3] );

void    Lgm_QuatToAxisAngle( double *Q, double *Angle, Lgm_Vector *u );

void    Lgm_AxisAngleToQuat( Lgm_Vector *u, double Angle, double *Q );

void    Lgm_QuatRotateVector( double *Q, Lgm_Vector *v, Lgm_Vector *vp );

double  Lgm_QuatMagnitude( double *Q );

double  Lgm_QuatVecLength( double *v );

double  Lgm_QuatVecDot( double *v1, double *v2 );

void    Lgm_QuatVecZero( double *v );

void    Lgm_QuatVecSet( double *v, double x, double y, double z );

void    Lgm_QuatVecAdd( double *a, double *b, double *c );

void    Lgm_QuatVecSub( double *a, double *b, double *c);

void    Lgm_QuatVecCopy( double *v1, double *v2 );

void    Lgm_QuatVecScale( double *v, double f );

void    Lgm_QuatVecNormalize( double *v );

void    Lgm_QuatVecCross( double *a, double *b, double *result );
void    Lgm_QuatCombineQuats( double Q1[4], double Q2[4], double Q[4] );
```

26 Octree Utilities

To allow efficient interpolation of arbitrary mesh-based B-field models. Routines for creating and manipulating octrees and an implementation of the kNN (k-nearest neighbor) algorithm which allows you to find k nearest neighbors in a point cloud efficiently.

```
void            Binary( unsigned int n, char *Str );

void            Lgm_OctreeFreeBranch( Lgm_OctreeCell *Cell );

void            Lgm_FreeOctree( Lgm_OctreeCell *ot );

Lgm_OctreeCell  *Lgm_CreateOctreeRoot( );

Lgm_OctreeCell  *Lgm_OctreeTraverseToLocCode( Lgm_OctreeCell *Cell, unsigned int
ChildLevel, unsigned int xLocationCode, unsigned int yLocationCode, unsigned int
zLocationCode );

Lgm_OctreeCell  *Lgm_LocateNearestCell( Lgm_OctreeCell *Root, Lgm_Vector *q );

double          MinDist( Lgm_OctreeCell *Cell, Lgm_Vector *q );

double          InsertCell( Lgm_OctreeCell *Cell, Lgm_Vector *q, pQueue **PQ,
double MaxDist2 );

void            InsertPoint( Lgm_OctreeCell *Cell, int j, Lgm_Vector *q, pQueue
**PQ );

Lgm_OctreeCell  *DescendTowardClosestLeaf( Lgm_OctreeCell *Node, pQueue **PQ,
Lgm_Vector *q, double MaxDist2 );

pQueue          *PopObj( pQueue **PQ );

int             Lgm_Octree_kNN( Lgm_Vector *q, Lgm_OctreeCell *Root, int K, int
*Kgot, double MaxDist2, Lgm_OctreeData *kNN );

Lgm_OctreeCell  *CreateNewOctants( Lgm_OctreeCell *Parent );

void            SubDivideVolume( Lgm_OctreeCell *Vol );

Lgm_OctreeCell  *Lgm_InitOctree( Lgm_Vector *ObjectPoints, Lgm_Vector *ObjectData,
unsigned long int N, double *Min, double *Max, double *Diff );
```

27 List all the other .h files too....

28 References

Akasofu S.--I. and S. Chapman, Solar Terrestrial Physics, Oxford ?University Press, 1972.

Astronomical Almanac, a joint publication of the U. S. Nautical Almanac Office, United States Naval Observatory (USNO), in the United States and Her Majesty's Nautical Almanac Office (HMNAO), United Kingdom Hydrographic Office (UKHO), in the United Kingdom, Published yearly.

Chapman S. and J. Bartels, Geomagnetism, Vol 2, Oxford University Press, 1962.

Duffet-Smith, Practical Astronomy with your Calculator, Cambridge University Press; 3 edition, 1988. (Good general (but brief) introduction to concepts and reductions).

Montenbruck, O., and T. Pfleger, Astronomy on the Personal Computer, Springer, 3rd edition, 1988. (Some good algorithms in here. There is a new edition of this book out now, but I dont have it).

Seidelmann, K. (Ed.), Explanatory Supplement to the Astronomical Almanac, University Science Books; Revised edition, 2005. (Definitive explanations and theory.)

Vallado, D. A., P. Crawford, R. Hujsak, and T. S. Kelso, Revisiting Spacetrack Report #3, American Institute of Aeronautics and Astronautics, 2006-6753, 2006. (Excellent discussion of TLEs, SGP4, and corrections to the older buggy SGP4.)

Vallado, D. A., Fundamentals of Astrodynamics and Applications, 3rd Edition, Space Technology Library, Springer, 2007. (Becoming the “standard” text for astrodynamical coordinate system reductions and more.)