My Project

2.0

Generated by Doxygen 1.13.2

Chapter 1

Bug List

File Accel.cpp

No known bugs

File Accel.h

No known bugs

File AccelHarmonic.cpp

No known bugs

File AccelHarmonic.h

No known bugs

File AccelPointMass.cpp

No known bugs

File AccelPointMass.h

No known bugs

File AzElPa.cpp

No known bugs

File AzElPa.h

No known bugs

File Cheb3D.cpp

No known bugs

File Cheb3D.h

No known bugs

File DEInteg.cpp

No known bugs

File DEInteg.h

No known bugs

File EccAnom.cpp

No known bugs

File EccAnom.h

No known bugs

File EKF_GEOS3.cpp

No known bugs

File EqnEquinox.cpp

No known bugs

File EqnEquinox.h

No known bugs

File Frac.cpp

No known bugs

File Frac.h

No known bugs

File G_AccelHarmonic.cpp

No known bugs

File G_AccelHarmonic.h

No known bugs

File gast.cpp

No known bugs

File gast.h

No known bugs

File GHAMatrix.cpp

No known bugs

File GHAMatrix.h

No known bugs

File GLOBAL.cpp

No known bugs

File GLOBAL.h

No known bugs

File gmst.cpp

No known bugs

File gmst.h

No known bugs

File IERS.cpp

No known bugs

File IERS.h

No known bugs

File JPL_Eph_DE430.cpp

Noknownbugs

File JPL_Eph_DE430.h

No known bugs

File Legendre.cpp

No known bugs

File Legendre.h

No known bugs

File LTC.cpp

No known bugs

File LTC.h

No known bugs

File matrix.cpp

No known bugs

File matrix.h

No known bugs

File MeanObliquity.cpp

No known bugs

File MeanObliquity.h

No known bugs

File MeasUpdate.cpp

No known bugs

File MeasUpdate.h

No known bugs

File Mjday.cpp

No known bugs

File Mjday.h

No known bugs

File Mjday_TDB.cpp

No known bugs

File Mjday_TDB.h

No known bugs

File NutAngles.cpp

No known bugs

File NutAngles.h

No known bugs

File NutMatrix.cpp

No known bugs

File NutMatrix.h

No known bugs

File PoleMatrix.cpp

No known bugs

File PoleMatrix.h

No known bugs

File Position.cpp

No known bugs

File Position.h

No known bugs

File PrecMatrix.cpp

No known bugs

File PrecMatrix.h

No known bugs

File R_x.cpp

No known bugs

File R_x.h

No known bugs

File R_y.cpp

No known bugs

File R_y.h

No known bugs

File R_z.cpp

No known bugs

File R_z.h

No known bugs

File SAT_Const.h

No known bugs

File sign_.cpp

No known bugs

File sign_.h

No known bugs

File timediff.cpp

No known bugs

File timediff.h

No known bugs

File TimeUpdate.cpp

No known bugs

File TimeUpdate.h

No known bugs

File VarEqn.cpp

No known bugs

File VarEqn.h

No known bugs

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Matrix Class Reference

Public Member Functions

- Matrix ()
- Matrix (const int n)
- Matrix (const int n_row, const int n_column)
- double & operator() (const int n)
- double & operator() (const int row, const int column)
- Matrix & operator+ (Matrix &m)
- Matrix & operator- (Matrix &m)
- Matrix & operator∗ (Matrix &m)
- Matrix & operator/ (Matrix &m)
- Matrix & operator= (Matrix &m)
- Matrix & operator+ (double d)
- Matrix & operator- (double d)
- Matrix & operator∗ (double d)
- Matrix & operator/ (double d)

Public Attributes

- int n_row
- int n_column
- double ∗∗ data

Friends

- ostream & operator<< (ostream &o, Matrix &m)

4.1.1 Detailed Description

Definition at line 18 of file matrix.h.

4.1.2 Constructor & Destructor Documentation

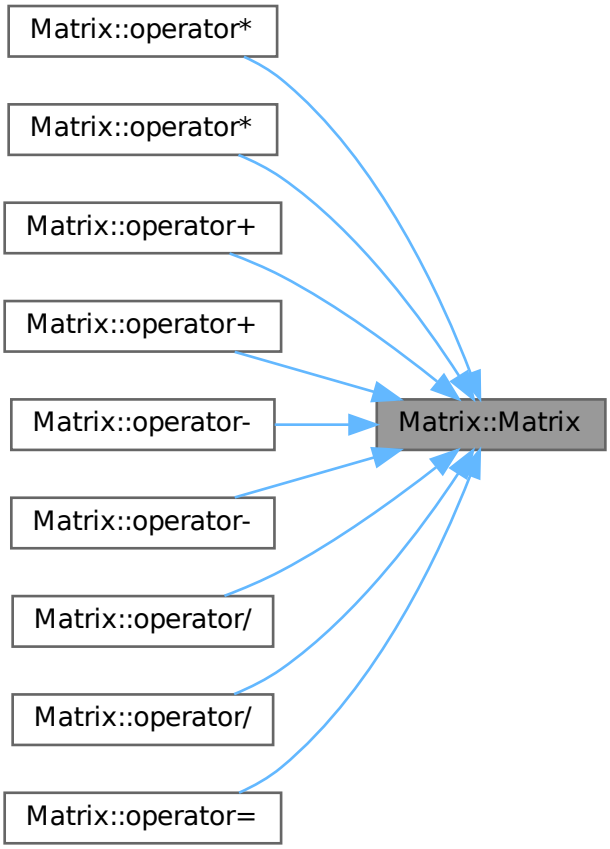
4.1.2.1 Matrix() [1/3]

```
Matrix::Matrix ()
```

Crea un objeto Matrix vacio

Definition at line 10 of file matrix.cpp.

Here is the caller graph for this function:



4.1.2.2 Matrix() [2/3]

```
Matrix::Matrix (
            const int n)
```

Crea una matriz [0][n] que simula un vector

Parameters

n	número de columas del vector

Definition at line 17 of file matrix.cpp.

4.1.2.3 Matrix() [3/3]

```
Matrix::Matrix (
            const int n_row,
            const int n_column)
```

Crea una matriz [n_row][n_column]

Parameters

n_row	número de filas de la matriz
n_column	número de columas de la matriz

Definition at line 35 of file matrix.cpp.

4.1.3 Member Function Documentation

4.1.3.1 operator()() [1/2]

```
double & Matrix::operator() (
            const int n)
```

Obtiene el elemento [(n-1)/n_column][(n-1)n_column]

Parameters

n	elemnto

Definition at line 55 of file matrix.cpp.

4.1.3.2 operator()() [2/2]

```
double & Matrix::operator() (
            const int row,
            const int column)
```

Obtiene el elemento [(row-1)][(column-1)]

Parameters

n_row	fila de la matriz
n_column	columa de la matriz

Definition at line 64 of file matrix.cpp.

4.1.3.3 operator*() [1/2]

```
Matrix & Matrix::operator* (
            double d)
```

Multiplica todas las componentes de this con d y devuelve una nueva Matrix

Parameters

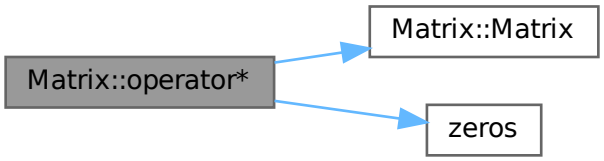
d	valor a ser operado por cada componente de la matriz

Returns

una Matrix donde todas sus componentes se le multiplica d, sin modificar las matrices

Definition at line 192 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.4 operator∗() [2/2]

```
Matrix & Matrix::operator* (
            Matrix & m)
```

Multiplación sobre matrices para 2(this∗m) Matrix y devuelve el valor

Parameters

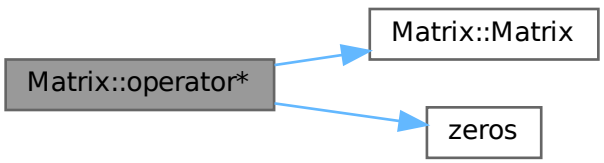
m	Matrix

Returns

devuelve un Matrix = this∗m, sin modificarlos

Definition at line 108 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.5 operator+() [1/2]

```
Matrix & Matrix::operator+ (
            double d)
```

Suma todas las componentes de this con d y devuelve una nueva Matrix

Parameters

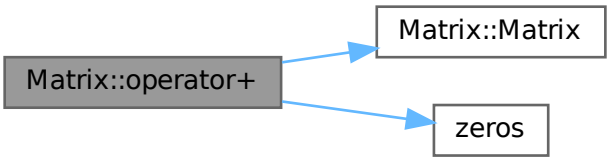
d	valor a ser operado por cada componente de la matriz

Returns

una Matrix donde todas sus componentes se le suma d, sin modificar las matrices

Definition at line 168 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.6 operator+() [2/2]

```
Matrix & Matrix::operator+ (
            Matrix & m)
```

Suma 2(this+m) Matrix y devuelve el valor

Parameters

m	Matrix

Returns

devuelve un Matrix suma de this+ m, sin modificarlos

Definition at line 74 of file matrix.cpp.

Here is the call graph for this function:

4.1.3.7 operator-() [1/2]

```
Matrix & Matrix::operator- (
            double d)
```

Resta todas las componentes de this con d y devuelve una nueva Matrix

Parameters

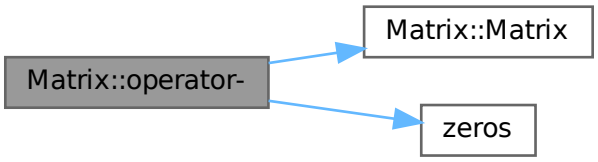
d	valor a ser operado por cada componente de la matriz

Returns

una Matrix donde todas sus componentes se le resta d, sin modificar las matrices

Definition at line 180 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.8 operator-() [2/2]

```
Matrix & Matrix::operator- (
            Matrix & m)
```

Resta 2(this-m) Matrix y devuelve el valor

Parameters

m	Matrix

Returns

devuelve un Matrix resta de this-m, sin modificarlos

Definition at line 91 of file matrix.cpp.

Here is the call graph for this function:

4.1.3.9 operator/() [1/2]

```
Matrix & Matrix::operator/ (
            double d)
```

Divide todas las componentes de this con d y devuelve una nueva Matrix

Parameters

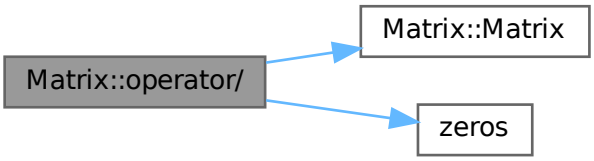
d	valor a ser operado por cada componente de la matriz

Returns

una Matrix donde todas sus componentes se le divide d, sin modificar las matrices

Definition at line 202 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.10 operator/() [2/2]

```
Matrix & Matrix::operator/ (
            Matrix & m)
```

this∗(m^(-1)) y devuelve el valor

Parameters

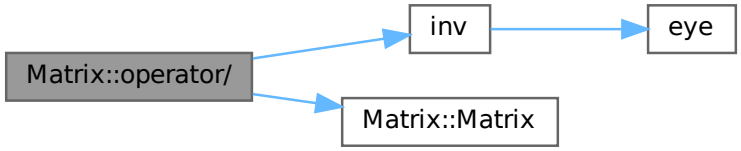
m	Matrix

Returns

devuelve un Matrix =this-∗m^(-1), sin modificarlos

Definition at line 127 of file matrix.cpp.

Here is the call graph for this function:



4.1.3.11 operator=()

```
Matrix & Matrix::operator= (
            Matrix & m)
```

Se crea una Matrix equivalente a m y se le asigna a this

Parameters

m	Matrix

Returns

una Matrix=m, sin modificar las matrices

Definition at line 138 of file matrix.cpp.

Here is the call graph for this function:

4.1.4 Friends And Related Symbol Documentation

4.1.4.1 operator<<

```
ostream & operator<< (
            ostream & o,
            Matrix & m)  [friend]
```

Definition at line 212 of file matrix.cpp.

4.1.5 Member Data Documentation

4.1.5.1 data

```
double** Matrix::data
```

Definition at line 21 of file matrix.h.

4.1.5.2 n_column

```
int Matrix::n_column
```

Definition at line 20 of file matrix.h.

4.1.5.3 n_row

```
int Matrix::n_row
```

Definition at line 20 of file matrix.h.

The documentation for this class was generated from the following files:

- matrix.h
- matrix.cpp

4.2 Param Struct Reference

Public Attributes

- double Mjd_UTC
- double Mjd_TT
- int n
- int m
- int sun
- int moon
- int planets

4.2.1 Detailed Description

Definition at line 13 of file GLOBAL.h.

4.2.2 Member Data Documentation

4.2.2.1 m

```
int Param::m
```

Definition at line 15 of file GLOBAL.h.

4.2.2.2 Mjd_TT

```
double Param::Mjd_TT
```

Definition at line 14 of file GLOBAL.h.

4.2.2.3 Mjd_UTC

```
double Param::Mjd_UTC
```

Definition at line 14 of file GLOBAL.h.

4.2.2.4 moon

```
int Param::moon
```

Definition at line 15 of file GLOBAL.h.

4.2.2.5 n

```
int Param::n
```

Definition at line 15 of file GLOBAL.h.

4.2.2.6 planets

```
int Param::planets
```

Definition at line 15 of file GLOBAL.h.

4.2.2.7 sun

```
int Param::sun
```

Definition at line 15 of file GLOBAL.h.

The documentation for this struct was generated from the following file:

- GLOBAL.h

Chapter 5

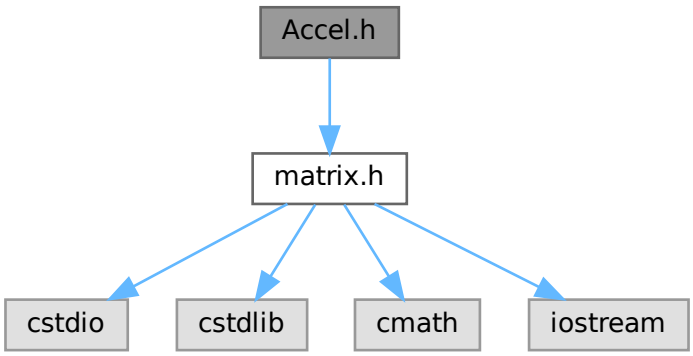
File Documentation

5.1 Accel.h File Reference

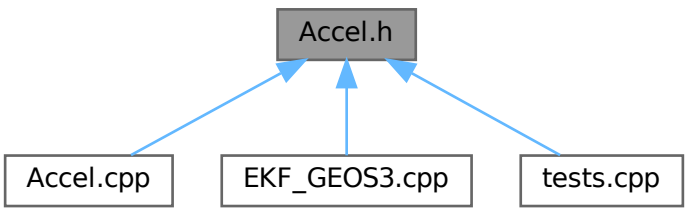
El archivo contiene la funcion Accel.

```
#include "matrix.h"
```

Include dependency graph for Accel.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & Accel (double x, Matrix Y)

5.1.1 Detailed Description

El archivo contiene la funcion Accel.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Accel.h.

5.1.2 Function Documentation

5.1.2.1 Accel()

```
Matrix & Accel (
            double x,
            Matrix Y)
```

Computes the acceleration of an Earth orbiting satellite due to

- the Earth's harmonic gravity field,
- the gravitational perturbations of the Sun and Moon
- the solar radiation pressure and
- the atmospheric drag

Parameters

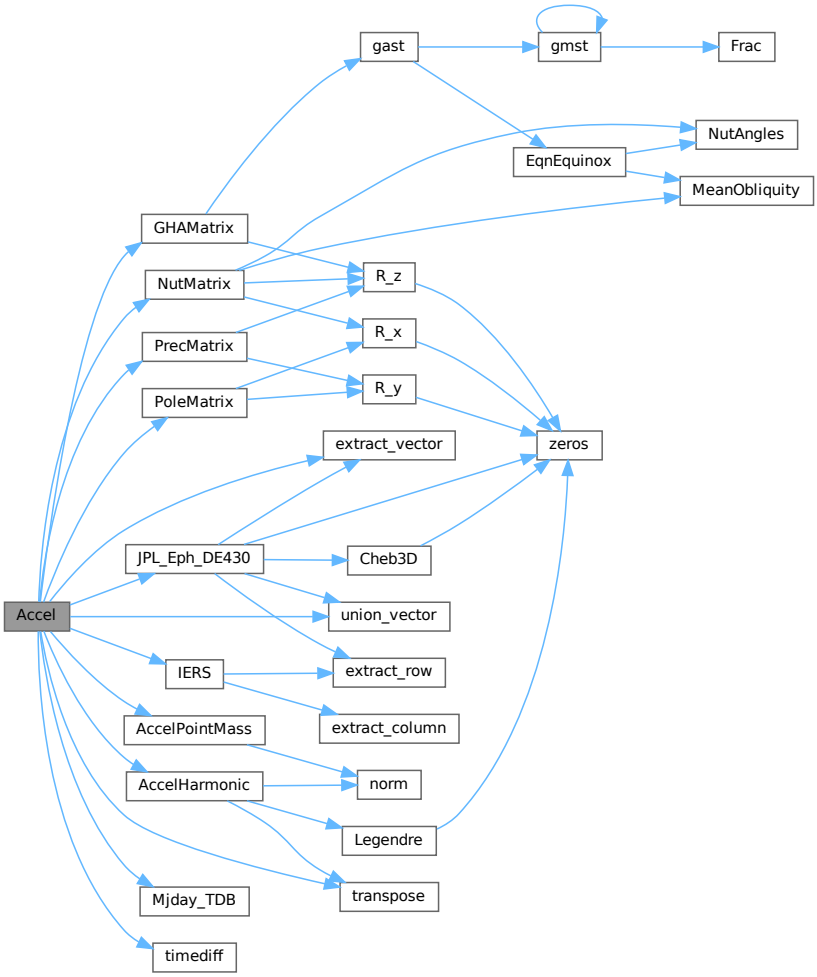
Mjd_TT	Terrestrial Time (Modified Julian Date)
Y	Satellite state vector in the ICRF/EME2000 system

Returns

dY Acceleration (a=d^2r/dt^2) in the ICRF/EME2000 system

Definition at line 20 of file Accel.cpp.

Here is the call graph for this function:



5.2 Accel.h

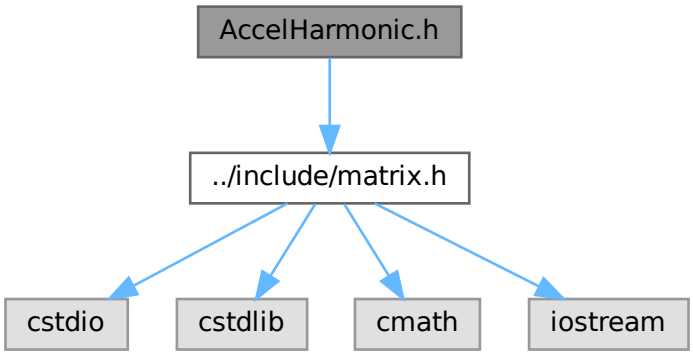
Go to the documentation of this file.

```
00001 #ifndef _Accel_
00002 #define _Accel_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00023     Matrix& Accel(double x,Matrix Y);
00024 #endif
00025
00026
```

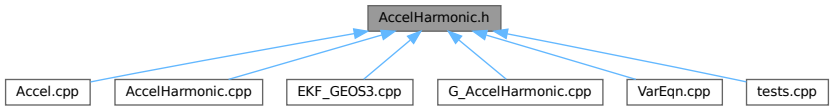
5.3 AccelHarmonic.h File Reference

El archivo contiene la funcion AccelHarmonic.

`#include "../include/matrix.h"`
Include dependency graph for AccelHarmonic.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & AccelHarmonic (Matrix r, Matrix E, int n_max, int m_max)

5.3.1 Detailed Description

El archivo contiene la funcion AccelHarmonic.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AccelHarmonic.h.

5.3.2 Function Documentation

5.3.2.1 AccelHarmonic()

```
Matrix & AccelHarmonic (
            Matrix r,
            Matrix E,
            int n_max,
            int m_max)
```

Parameters

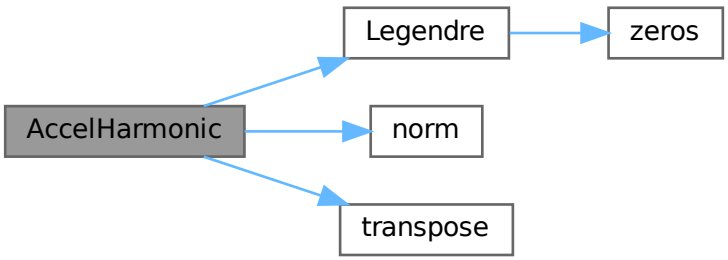
r	Satellite position vector in the inertial system
E	Transformation matrix to body-fixed system
n_max	Maximum degree
m_max	Maximum order (m_max<=n_max; m_max=0 for zonals, only)

Returns

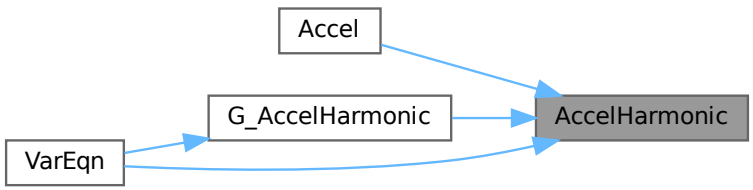
a Acceleration (a=d^2r/dt^2)

Definition at line 12 of file AccelHarmonic.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4 AccelHarmonic.h

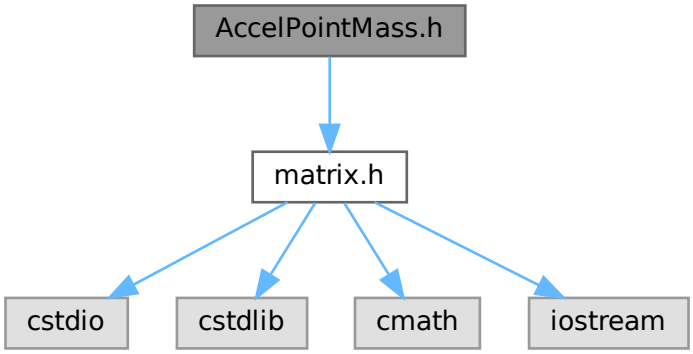
Go to the documentation of this file.

```
00001 #ifndef _AccelHarmonic_
00002 #define _AccelHarmonic_
00003 using namespace std;
00004 #include "../include/matrix.h"
00005
00012
00020     Matrix& AccelHarmonic(Matrix r,Matrix E,int n_max,int m_max);
00021 #endif
00022
00023
```

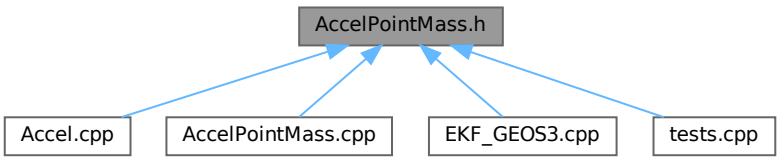
5.5 AccelPointMass.h File Reference

El archivo contiene la funcion AccelPointMass.

`#include "matrix.h"`
Include dependency graph for AccelPointMass.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & AccelPointMass (Matrix &r, Matrix &s, double GM)

5.5.1 Detailed Description

El archivo contiene la funcion AccelPointMass.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AccelPointMass.h.

5.5.2 Function Documentation

5.5.2.1 AccelPointMass()

```
Matrix & AccelPointMass (
            Matrix & r,
            Matrix & s,
            double GM)
```

Parameters

r	Satellite position vector
s	Point mass position vector
GM	Gravitational coefficient of point mass

Returns

Acceleration (a=d^2r/dt^2)

Definition at line 10 of file AccelPointMass.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

5.6 AccelPointMass.h

Go to the documentation of this file.

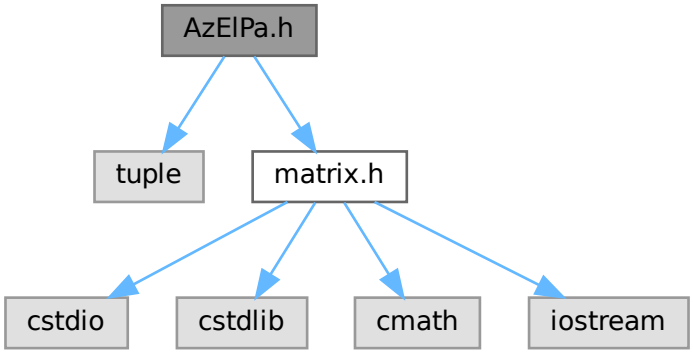
```
00001 #ifndef _AccelPointMass_
00002 #define _AccelPointMass_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00019     Matrix&    AccelPointMass(Matrix& r,Matrix& s,double GM);
00020 #endif
00021
00022
```

5.7 AzElPa.h File Reference

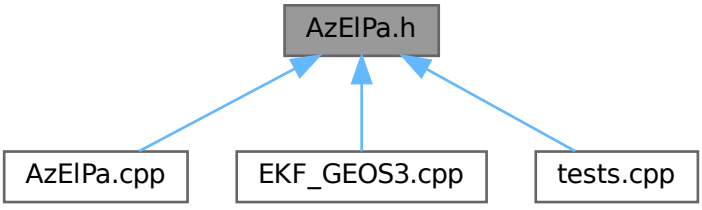
El archivo contiene la funcion AzElPa.

```
#include <tuple>
#include "matrix.h"
```

Include dependency graph for AzElPa.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< double, double, Matrix &, Matrix & > AzElPa (Matrix s)

5.7.1 Detailed Description

El archivo contiene la funcion AzElPa.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AzElPa.h.

5.7.2 Function Documentation

5.7.2.1 AzElPa()

```
tuple< double, double, Matrix &, Matrix & > AzElPa (
            Matrix s)
```

Computes azimuth, elevation and partials from local tangent coordinates s

Parameters

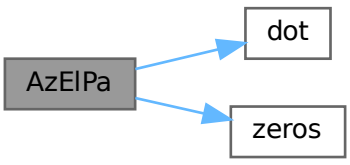
s	Matrix 1x3 Topocentric local tangent coordinates (East-North-Zenith frame)

Returns

tuple<A,E,dAds,dEds> Azimuth [rad],Elevation [rad],Partials of azimuth w.r.t. s,Partials of elevation w.r.t. s

Definition at line 9 of file AzElPa.cpp.

Here is the call graph for this function:



5.8 AzElPa.h

Go to the documentation of this file.

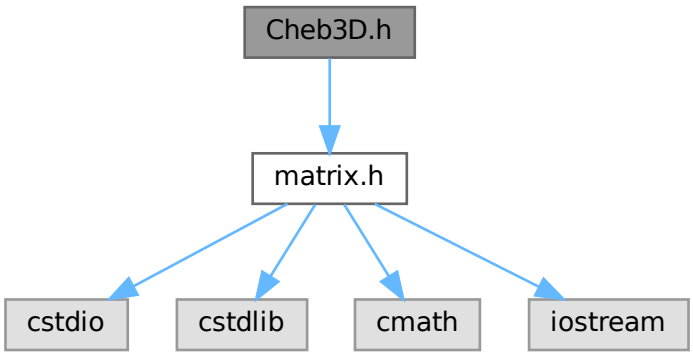
```
00001 #ifndef _AzElPa_
00002 #define _AzElPa_
00003 #include <tuple>
00004 #include "matrix.h"
00005 using namespace std;
00006
00018     tuple<double,double,Matrix&,Matrix&> AzElPa(Matrix s);
00019 #endif
00020
00021
```

5.9 Cheb3D.h File Reference

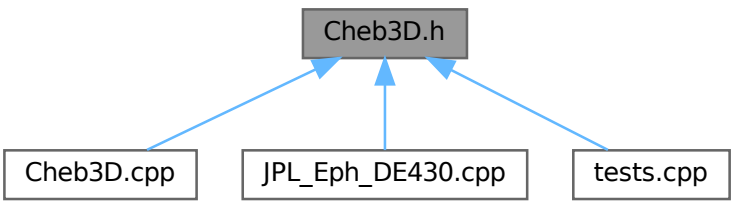
El archivo contiene la funcion Cheb3D.

```
#include "matrix.h"
```

Include dependency graph for Cheb3D.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & Cheb3D (double t, int N, double Ta, double Tb, Matrix &Cx, Matrix &Cy, Matrix &Cz)

5.9.1 Detailed Description

El archivo contiene la funcion Cheb3D.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Cheb3D.h.

5.9.2 Function Documentation

5.9.2.1 Cheb3D()

```
Matrix & Cheb3D (
            double t,
            int N,
            double Ta,
            double Tb,
            Matrix & Cx,
            Matrix & Cy,
            Matrix & Cz)
```

Parameters

t	time
N	Number of coefficients
Ta	Begin interval
Tb	End interval
Cx	Coefficients of Chebyshev polyomial (x-coordinate)
Cy	Coefficients of Chebyshev polyomial (y-coordinate)
Cz	Coefficients of Chebyshev polyomial (z-coordinate)

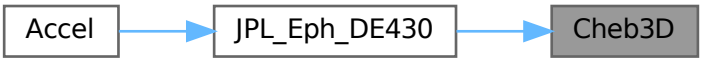
Returns

Chebyshev approximation of 3-dimensional vectors

Definition at line 9 of file Cheb3D.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



5.10 Cheb3D.h

Go to the documentation of this file.

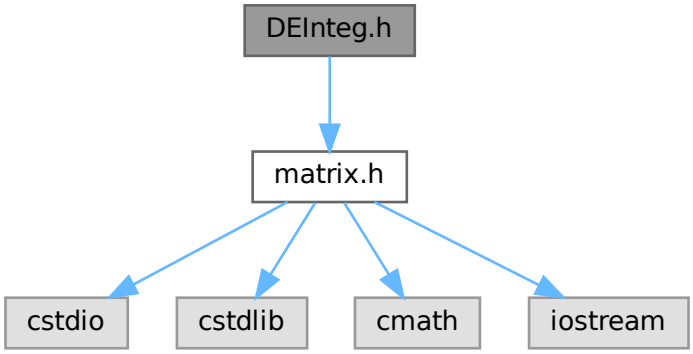
```
00001 #ifndef _Cheb3D_
00002 #define _Cheb3D_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00023     Matrix&   Cheb3D( double t, int N, double Ta, double Tb, Matrix& Cx,Matrix& Cy,Matrix& Cz);
00024 #endif
00025
00026
```

5.11 DEInteg.h File Reference

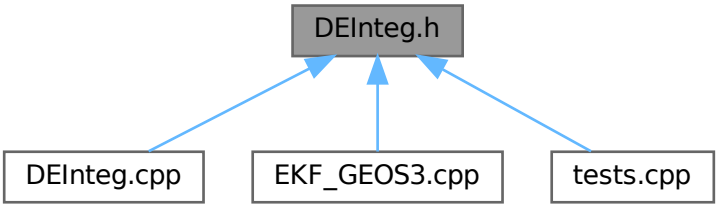
El archivo contiene la funcion DEInteg.

```
#include "matrix.h"
```

Include dependency graph for DEInteg.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & DEInteg (Matrix &f(double t, Matrix y), double t, double tout, double relerr, double abserr, int n_eqn, Matrix &y)

5.11.1 Detailed Description

El archivo contiene la funcion DEInteg.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file DEInteg.h.

5.11.2 Function Documentation

5.11.2.1 DEInteg()

```
Matrix & DEInteg (
            Matrix & fdouble t, Matrix y,
            double t,
            double tout,
            double relerr,
            double abserr,
            int n_eqn,
            Matrix & y)
```

Numerical integration methods for ordinaray differential equations This module provides implemenation of the variable order variable stepsize multistep method of Shampine & Gordon.

Parameters

f	funcion pasas double y Matrix devuelve Matrix
t	double
tout	double
relerr	double
abserr	double
n_eqn	int
y	Matrix

Returns

Matrix resultado

5.12 DEInteg.h

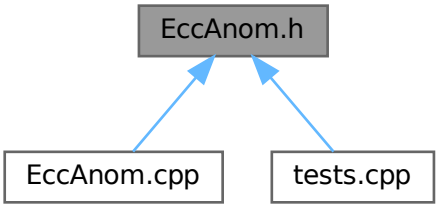
Go to the documentation of this file.

```
00001 #ifndef _DEInteg_
00002 #define _DEInteg_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00025      Matrix& DEInteg(Matrix& f(double t,Matrix y),double t, double tout,double relerr,double
      abserr,int n_eqn,Matrix &y);
00026 #endif
00027
00028
```

5.13 EccAnom.h File Reference

El archivo contiene la funcion EccAnom.

This graph shows which files directly or indirectly include this file:



Functions

- double EccAnom (double M, double e)

5.13.1 Detailed Description

El archivo contiene la funcion EccAnom.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file EccAnom.h.

5.13.2 Function Documentation

5.13.2.1 EccAnom()

```
double EccAnom (
            double M,
            double e)
```

Computes the eccentric anomaly for elliptic orbits

Parameters

M	Mean anomaly in [rad]
e	Eccentricity of the orbit [0,1]

Returns

double resultado

Definition at line 12 of file EccAnom.cpp.

5.14 EccAnom.h

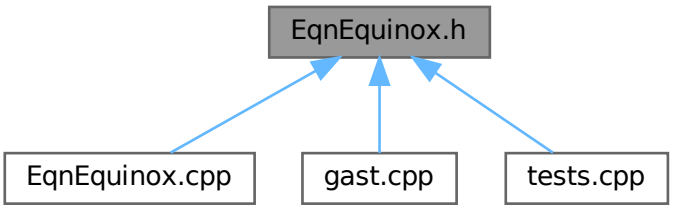
Go to the documentation of this file.

```
00001 #ifndef _EccAnom_
00002 #define _EccAnom_
00003
00004 using namespace std;
00005
00018     double EccAnom (double M,double e);
00019 #endif
00020
00021
```

5.15 EqnEquinox.h File Reference

El archivo contiene la funcion EqnEquinox.

This graph shows which files directly or indirectly include this file:



Functions

- double EqnEquinox (double Mjd_TT)

5.15.1 Detailed Description

El archivo contiene la funcion EqnEquinox.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file EqnEquinox.h.

5.15.2 Function Documentation

5.15.2.1 EqnEquinox()

```
double EqnEquinox (
            double Mjd_TT)
```

Computation of the equation of the equinoxes

Parameters

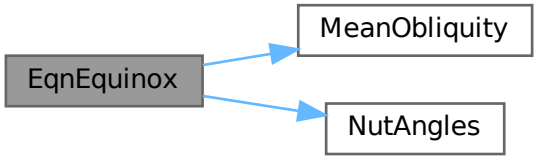
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

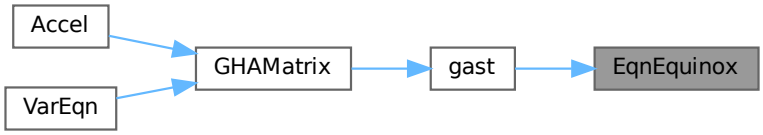
double Equation of the equinoxes

Definition at line 13 of file EqnEquinox.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.16 EqnEquinox.h

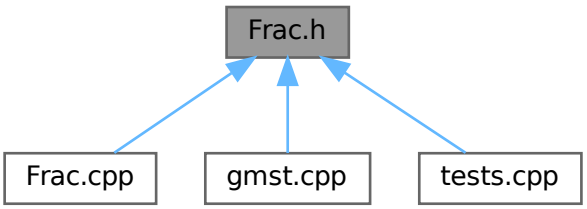
Go to the documentation of this file.

```
00001 #ifndef _EqnEquinox_
00002 #define _EqnEquinox_
00003 using namespace std;
00004
00016     double EqnEquinox (double Mjd_TT);
00017 #endif
00018
00019
```

5.17 Frac.h File Reference

El archivo contiene la funcion Frac.

This graph shows which files directly or indirectly include this file:



Functions

- double Frac (double x)

5.17.1 Detailed Description

El archivo contiene la funcion Frac.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Frac.h.

5.17.2 Function Documentation

5.17.2.1 Frac()

```
double Frac (
            double x)
```

Parameters

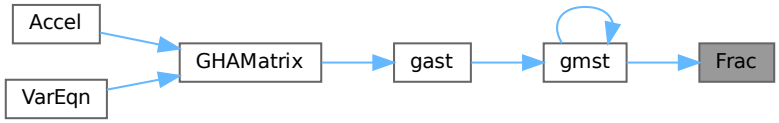
x	double

Returns

double parte fraccion de x

Definition at line 10 of file Frac.cpp.

Here is the caller graph for this function:



5.18 Frac.h

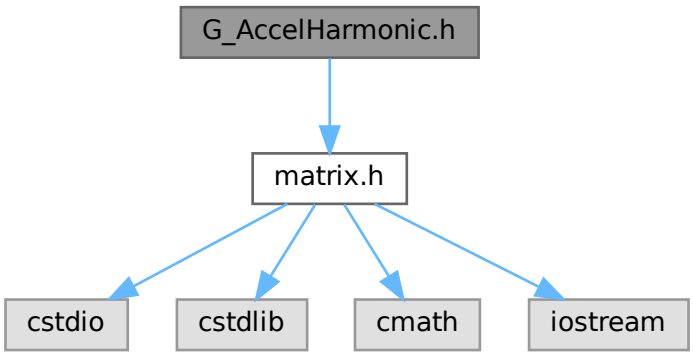
Go to the documentation of this file.

```
00001 #ifndef _Frac_
00002 #define _Frac_
00003
00004 using namespace std;
00005
00016     double Frac(double x);
00017 #endif
00018
00019
```

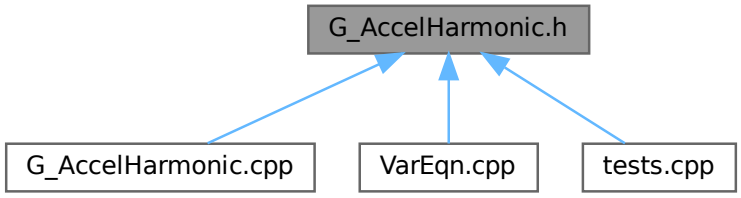
5.19 G_AccelHarmonic.h File Reference

El archivo contiene la funcion G_AccelHarmonic.

`#include "matrix.h"`
Include dependency graph for G_AccelHarmonic.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & G_AccelHarmonic (Matrix r, Matrix U, int n_max, int m_max)

5.19.1 Detailed Description

El archivo contiene la funcion G_AccelHarmonic.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file G_AccelHarmonic.h.

5.19.2 Function Documentation

5.19.2.1 G_AccelHarmonic()

```
Matrix & G_AccelHarmonic (
            Matrix r,
            Matrix U,
            int n_max,
            int m_max)
```

Parameters

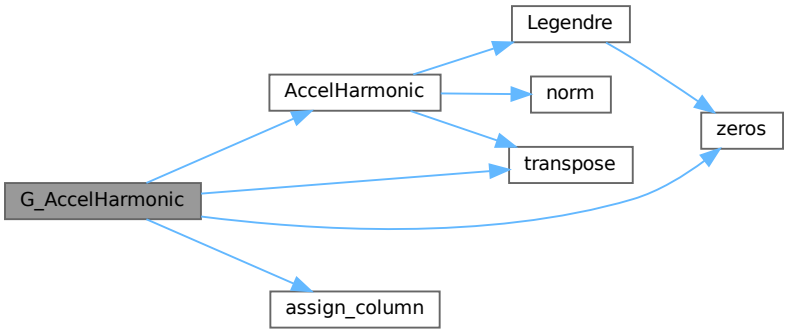
r	Satellite position vector in the true-of-date system
U	Transformation matrix to body-fixed syste
n	Gravity model degree
m	Gravity model order

Returns

G Gradient (G=da/dr) in the true-of-date system

Definition at line 10 of file G_AccelHarmonic.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.20 G_AccelHarmonic.h

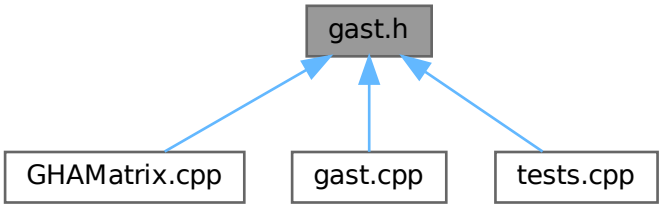
Go to the documentation of this file.

```
00001 #ifndef _G_AccelHarmonic_
00002 #define _G_AccelHarmonic_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00020     Matrix& G_AccelHarmonic( Matrix r,Matrix U, int n_max, int m_max );
00021 #endif
00022
00023
```

5.21 gast.h File Reference

El archivo contiene la funcion gast.

This graph shows which files directly or indirectly include this file:



Functions

- double gast (double Mjd_UT1)

5.21.1 Detailed Description

El archivo contiene la funcion gast.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file gast.h.

5.21.2 Function Documentation

5.21.2.1 gast()

```
double gast (
            double Mjd_UT1)
```

Greenwich Apparent Sidereal Time

Parameters

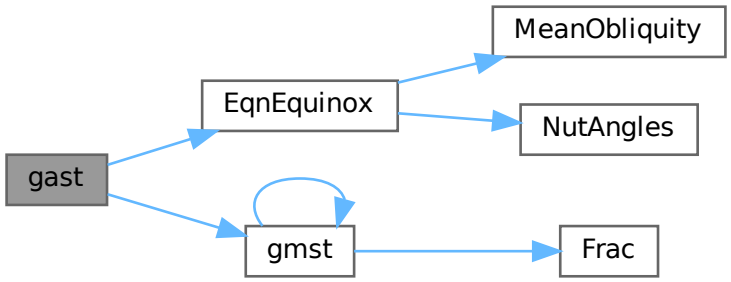
Mjd_UT1	Modified Julian Date UT1

Returns

gstime GAST in [rad]

Definition at line 14 of file gast.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.22 gast.h

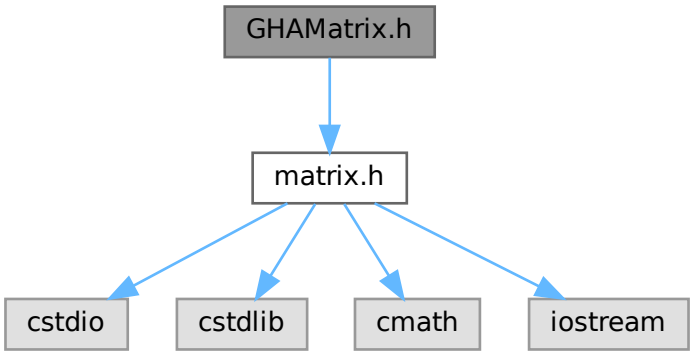
Go to the documentation of this file.

```
00001 #ifndef _gast_
00002 #define _gast_
00003
00004 using namespace std;
00005
00017     double gast(double Mjd_UT1);
00018 #endif
00019
00020
```

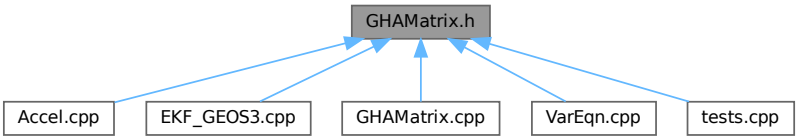
5.23 GHAMatrix.h File Reference

El archivo contiene la funcion GHAMatrix.

`#include "matrix.h"`
Include dependency graph for GHAMatrix.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & GHAMatrix (double Mjd_UT1)

5.23.1 Detailed Description

El archivo contiene la funcion GHAMatrix.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file GHAMatrix.h.

5.23.2 Function Documentation

5.23.2.1 GHAMatrix()

```
Matrix & GHAMatrix (
            double Mjd_UT1)
```

Transformation from true equator and equinox to Earth equator and Greenwich meridian system

Parameters

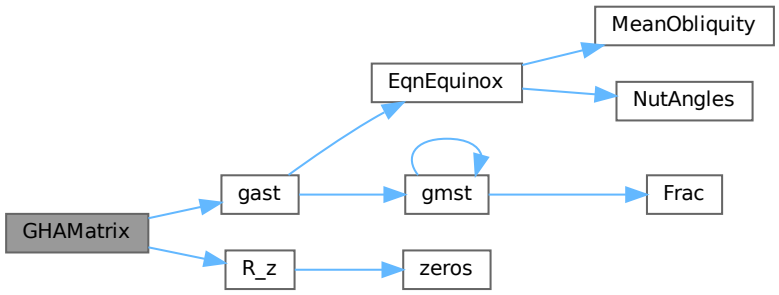
Mjd_UT1	Modified Julian Date UT1

Returns

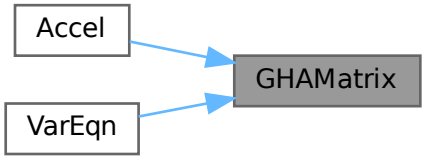
GHAmat Greenwich Hour Angle matrix

Definition at line 11 of file GHAMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.24 GHAMatrix.h

Go to the documentation of this file.

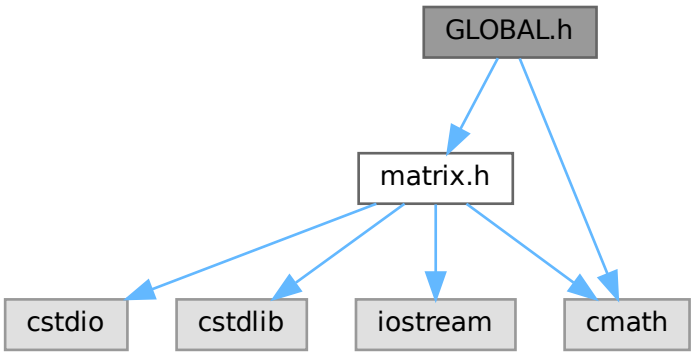
```
00001 #ifndef _GHAMatrix_
00002 #define _GHAMatrix_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00018     Matrix& GHAMatrix (double Mjd_UT1);
00019 #endif
00020
00021
```

5.25 GLOBAL.h File Reference

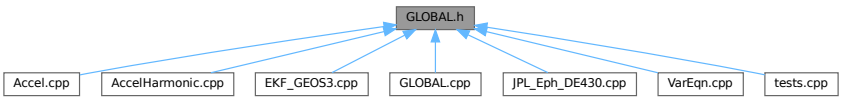
El archivo contiene la funcion GLOBAL.

```
#include "matrix.h"
#include <cmath>
```

Include dependency graph for GLOBAL.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Param

Functions

- void eop19620101 (int c=21413)
- void GGM03S (int n=181)
- void DE430Coeff (int row=2285, int column=1020)
- void AuxParamLoad ()
- void GEOS3 (int nobs=46)

Variables

- Param AuxParam
- Matrix eopdata
- Matrix Cnm
- Matrix Snm
- Matrix PC
- Matrix obs

5.25.1 Detailed Description

El archivo contiene la funcion GLOBAL.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file GLOBAL.h.

5.25.2 Function Documentation

5.25.2.1 AuxParamLoad()

```
void AuxParamLoad ()
```

Carga AuxParam

Definition at line 17 of file GLOBAL.cpp.

5.25.2.2 DE430Coeff()

```
void DE430Coeff (
            int row = 2285,
            int column = 1020)
```

Lee el archivo DE430Coeff.txt y recoge cada fila y lo asigna a PC

Parameters

row	número de filas a recoger
column	número de columnas a recoger

Definition at line 68 of file GLOBAL.cpp.

Here is the call graph for this function:

5.25.2.3 eop19620101()

```
void eop19620101 (
             int c = 21413)
```

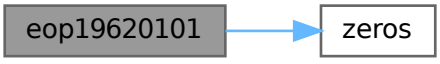
Lee el archivo eop19620101.txt y recoge cada fila y lo asigna a eopdata

Parameters

c	número de filas a recoger

Definition at line 26 of file GLOBAL.cpp.

Here is the call graph for this function:



5.25.2.4 GEOS3()

```
void GEOS3 (
             int nobs = 46)
```

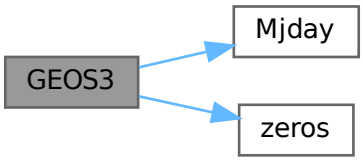
Lee el archivo GEOS3.txt y recoge cada fila y lo asigna a obs

Parameters

nobs	número de filas a recoger

Definition at line 87 of file GLOBAL.cpp.

Here is the call graph for this function:



5.25.2.5 GGM03S()

```
void GGM03S (
             int n = 181)
```

Lee el archivo GGM03S.txt y recoge cada fila y lo asigna a Cnm y Snm

Parameters

c	dimension de la matriz

Definition at line 47 of file GLOBAL.cpp.

Here is the call graph for this function:

5.25.3 Variable Documentation

5.25.3.1 AuxParam

```
Param AuxParam  [extern]
```

Definition at line 11 of file GLOBAL.cpp.

5.25.3.2 Cnm

```
Matrix Cnm  [extern]
```

Definition at line 13 of file GLOBAL.cpp.

5.25.3.3 eopdata

```
Matrix eopdata  [extern]
```

Definition at line 12 of file GLOBAL.cpp.

5.25.3.4 obs

```
Matrix obs  [extern]
```

Definition at line 16 of file GLOBAL.cpp.

5.25.3.5 PC

```
Matrix PC  [extern]
```

Definition at line 15 of file GLOBAL.cpp.

5.25.3.6 Snm

```
Matrix Snm  [extern]
```

Definition at line 14 of file GLOBAL.cpp.

5.26 GLOBAL.h

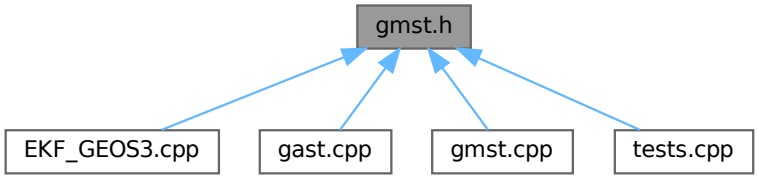
Go to the documentation of this file.

```
00001
00007 #ifndef _GLOBAL_
00008 #define _GLOBAL_
00009 using namespace std;
00010 #include "matrix.h"
00011 #include <cmath>
00012
00013 typedef struct{
00014     double Mjd_UTC,Mjd_TT;
00015     int n,m,sun,moon,planets;
00016 } Param;
00017
00018 extern Param AuxParam;
00019 extern Matrix eopdata;
00020 extern Matrix Cnm;
00021 extern Matrix Snm;
00022 extern Matrix PC;
00023 extern Matrix obs;
00024
00029 void eop19620101(int c=21413);
00030
00035 void GGM03S(int n=181);
00036
00042 void DE430Coeff(int row=2285,int column=1020);
00043
00047 void AuxParamLoad();
00048
00053 void GEOS3(int nobs=46);
00054
00055 #endif
00056
00057
```

5.27 gmst.h File Reference

El archivo contiene la funcion gmst.

This graph shows which files directly or indirectly include this file:



Functions

- double gmst (double Mjd_UT1)

5.27.1 Detailed Description

El archivo contiene la funcion gmst.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file gmst.h.

5.27.2 Function Documentation

5.27.2.1 gmst()

```
double gmst (
            double Mjd_UT1)
```

Greenwich Mean Sidereal Time

Parameters

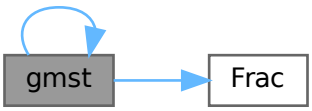
Mjd_UT1	Modified Julian Date UT1

Returns

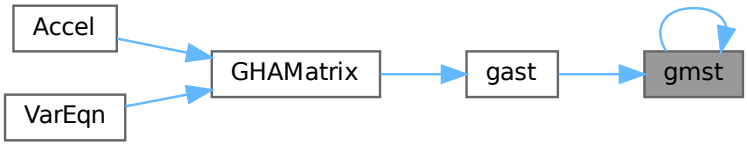
gmstime GMST in [rad]

Definition at line 12 of file gmst.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.28 gmst.h

Go to the documentation of this file.

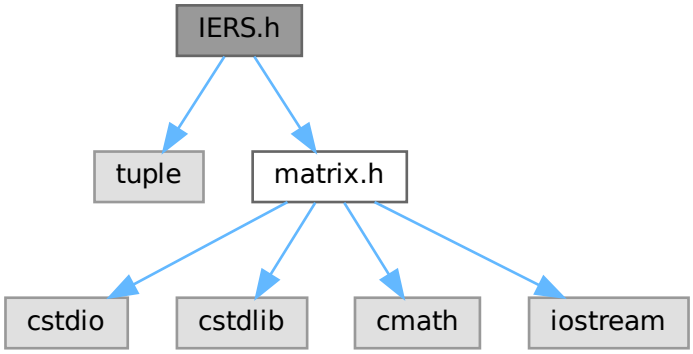
```
00001 #ifndef _gmst_
00002 #define _gmst_
00003
00004 using namespace std;
00005
00017     double gmst(double Mjd_UT1);
00018 #endif
00019
00020
```

5.29 IERS.h File Reference

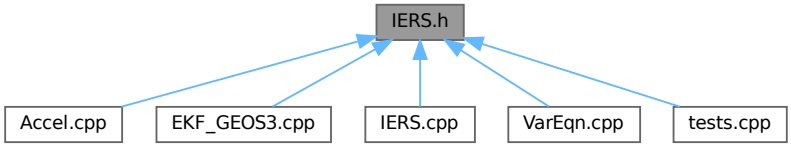
El archivo contiene la funcion IERS.

```
#include <tuple>
#include "matrix.h"
```

Include dependency graph for IERS.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< double, double, double, double, double, double, double, double, double > IERS (Matrix eop, double Mjd_UTC, char interp='n')

5.29.1 Detailed Description

El archivo contiene la funcion IERS.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file IERS.h.

5.29.2 Function Documentation

5.29.2.1 IERS()

```
tuple< double, double, double, double, double, double, double, double, double > IERS (
            Matrix eop,
            double Mjd_UTC,
            char interp = 'n')
```

IERS: Management of IERS time and polar motion data

Parameters

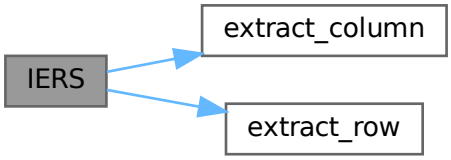
eop	matrix 4x13
Mjd_UTC	double
interp	char

Returns

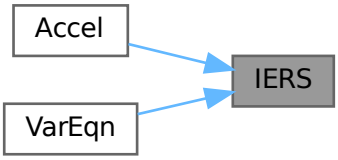
tupla <x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC>

Definition at line 10 of file IERS.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.30 IERS.h

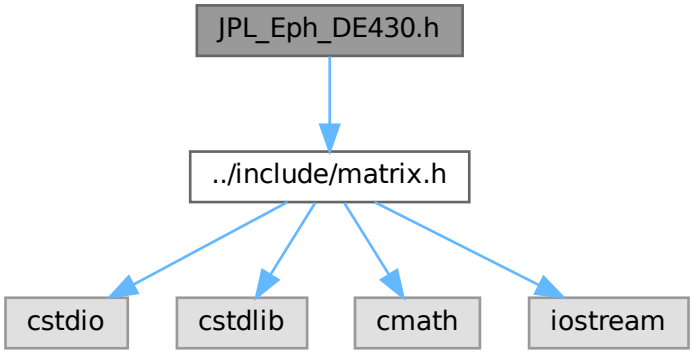
Go to the documentation of this file.

```
00001 #ifndef _IERS_
00002 #define _IERS_
00003 #include <tuple>
00004 #include "matrix.h"
00005 using namespace std;
00006
00020     tuple<double,double,double,double,double,double,double,double,double> IERS(Matrix eop,double
      Mjd_UTC,char interp='n');
00021 #endif
00022
00023
```

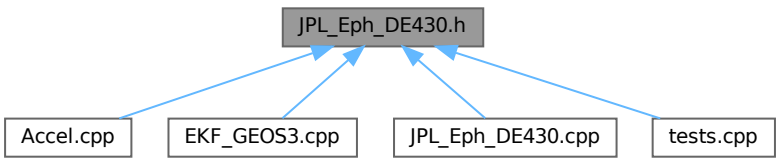
5.31 JPL_Eph_DE430.h File Reference

El archivo contiene la funcion JPL_Eph_DE430.

`#include "../include/matrix.h"`
Include dependency graph for JPL_Eph_DE430.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix & > JPL_Eph_DE430 (double Mjd_TDB)

5.31.1 Detailed Description

El archivo contiene la funcion JPL_Eph_DE430.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file JPL_Eph_DE430.h.

5.31.2 Function Documentation

5.31.2.1 JPL_Eph_DE430()

```
tuple< Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix
&, Matrix &, Matrix & > JPL_Eph_DE430 (
            double Mjd_TDB)
```

Parameters

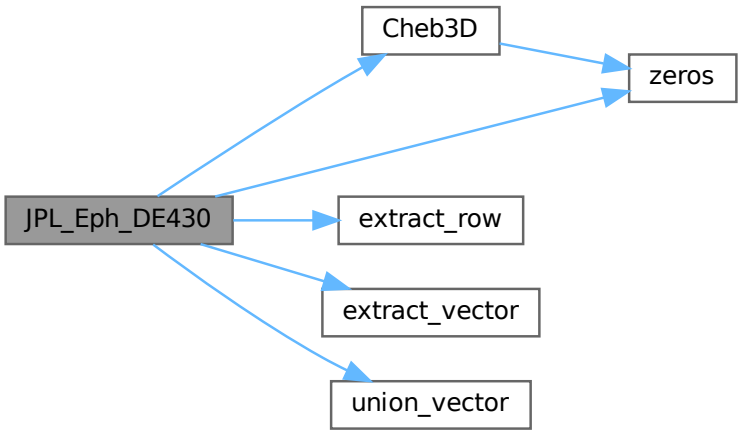
P	Matrix
Phi	Matrix
Qdt	Matrix

Returns

Matrix

Definition at line 12 of file JPL_Eph_DE430.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.32 JPL_Eph_DE430.h

Go to the documentation of this file.

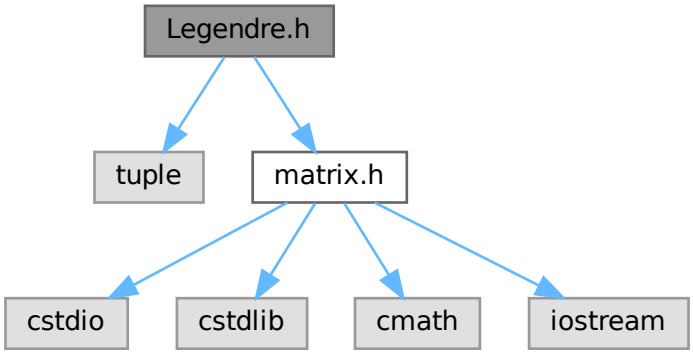
```
00001 #ifndef _JPL_Eph_DE430_
00002 #define _JPL_Eph_DE430_
00003 using namespace std;
00004 #include"../include/matrix.h"
00005
00018     tuple<Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&>
      JPL_Eph_DE430(double Mjd_TDB);
00019
00020 #endif
00021
00022
```

5.33 Legendre.h File Reference

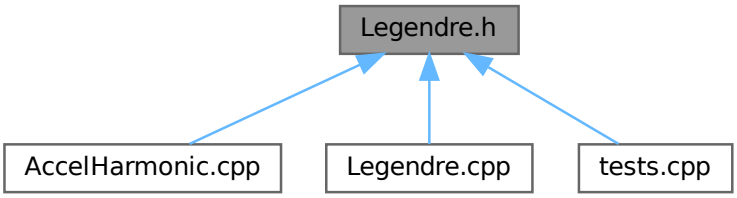
El archivo contiene la funcion Legendre.

```
#include <tuple>
#include "matrix.h"
```

Include dependency graph for Legendre.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< Matrix &, Matrix & > Legendre (int n, int m, double fi)

5.33.1 Detailed Description

El archivo contiene la funcion Legendre.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Legendre.h.

5.33.2 Function Documentation

5.33.2.1 Legendre()

```
tuple< Matrix &, Matrix & > Legendre (
            int n,
            int m,
            double fi)
```

Time differences [s]

Parameters

n	int
m	int
fi	double [rad]

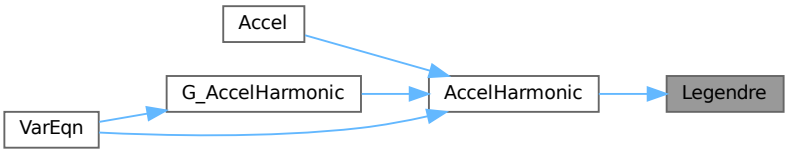
Returns

tupla <pnm,dpnm>

Definition at line 8 of file Legendre.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



5.34 Legendre.h

Go to the documentation of this file.

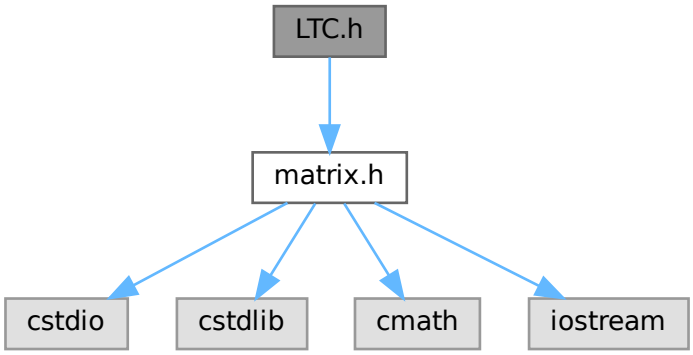
```
00001 #ifndef _Legendre_
00002 #define _Legendre_
00003 #include <tuple>
00004 #include "matrix.h"
00005 using namespace std;
00006
00020     tuple<Matrix&,Matrix&> Legendre(int n,int m,double fi);
00021 #endif
00022
00023
```

5.35 LTC.h File Reference

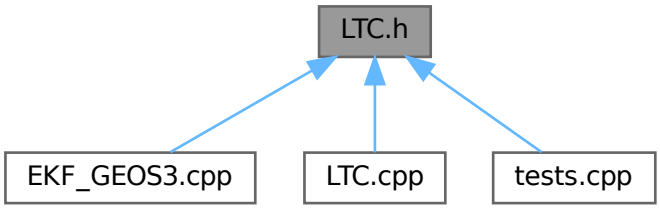
El archivo contiene la funcion LTC.

```
#include "matrix.h"
```

Include dependency graph for LTC.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & LTC (double lon, double lat)

5.35.1 Detailed Description

El archivo contiene la funcion LTC.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file LTC.h.

5.35.2 Function Documentation

5.35.2.1 LTC()

```
Matrix & LTC (
            double lon,
            double lat)
```

Transformation from Greenwich meridian system to local tangent coordinates

Parameters

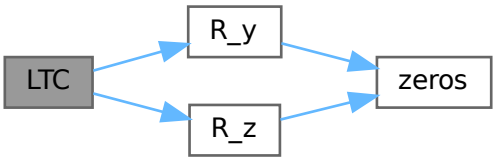
lon	-Geodetic East longitude [rad]
lat	-Geodetic latitude [rad]

Returns

M -Rotation matrix from the Earth equator and Greenwich meridian to the local tangent (East-North-Zenith) coordinate system

Definition at line 10 of file LTC.cpp.

Here is the call graph for this function:



5.36 LTC.h

Go to the documentation of this file.

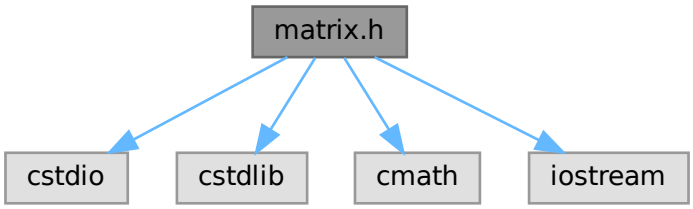
```
00001 #ifndef _LTC_
00002 #define _LTC_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00019     Matrix& LTC(double lon,double lat);
00020 #endif
00021
00022
```

5.37 matrix.h File Reference

El archivo contiene las funciones para operaciones con matrices y vectores junto con su definicion.

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <iostream>
```

Include dependency graph for matrix.h:



This graph shows which files directly or indirectly include this file:

Classes

- class Matrix

Functions

- ostream & operator<< (ostream &o, Matrix &m)
- Matrix & zeros (const int n_row, const int n_column)
- Matrix & eye (const int size)
- Matrix & transpose (Matrix &m)
- Matrix & inv (Matrix &m)
- Matrix & zeros (const int n)
- double norm (Matrix &m)
- double dot (Matrix &v, Matrix &w)
- Matrix & cross (Matrix &v, Matrix &w)
- Matrix & extract_vector (Matrix &v, int start, int end)
- Matrix & union_vector (Matrix &v, Matrix &w)
- Matrix & extract_row (Matrix &v, int i)
- Matrix & extract_column (Matrix &v, int i)
- Matrix & assign_row (Matrix &v, Matrix &w, int i)
- Matrix & assign_column (Matrix &v, Matrix &w, int i)

5.37.1 Detailed Description

El archivo contiene las funciones para operaciones con matrices y vectores junto con su definicion.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file matrix.h.

5.37.2 Function Documentation

5.37.2.1 assign_column()

```
Matrix & assign_column (
            Matrix & v,
            Matrix & w,
            int i)
```

Asigna la columna i-1 con w y lo devuelve

Parameters

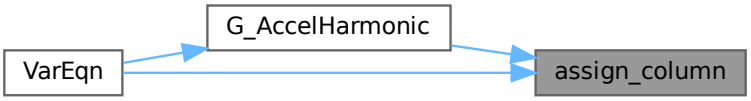
v	Matrix con tamaño x x y ,x pertenece a N y pertenece a N
w	Matrix con tamaño 1 x y ,y pertenece a N
i	es la columna tiene que ser >=1 && <=v.n_column

Returns

Matrix v con la columna i-1 cambiada por los elementos de w

Definition at line 424 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.2 assign_row()

```
Matrix & assign_row (
            Matrix & v,
            Matrix & w,
            int i)
```

Asigna la fila i-1 con w y lo devuelve

Parameters

v	Matrix con tamaño x x y ,x pertenece a N y pertenece a N
w	Matrix con tamaño 1 x y ,y pertenece a N
i	es la fila tiene que ser >=1 && <=v.n_row

Returns

Matrix v con la fila i-1 cambiada por los elementos de w

Definition at line 412 of file matrix.cpp.

5.37.2.3 cross()

```
Matrix & cross (
            Matrix & v,
            Matrix & w)
```

Devulve el producto vectorial

Parameters

v	Matrix con tamaño 1 x 3
w	Matrix con tamaño 1 x 3

Returns

producto escalar de v x w

Definition at line 353 of file matrix.cpp.

5.37.2.4 dot()

```
double dot (
            Matrix & v,
            Matrix & w)
```

Devulve el producto escalar

Parameters

v	Matrix con tamaño 1 x 3
w	Matrix con tamaño 1 x 3

Returns

producto escalar de v·w

Definition at line 342 of file matrix.cpp.

Here is the caller graph for this function:

5.37.2.5 extract_column()

```
Matrix & extract_column (
            Matrix & v,
            int i)
```

Extrae la columna i-1 de v y lo devuelve

Parameters

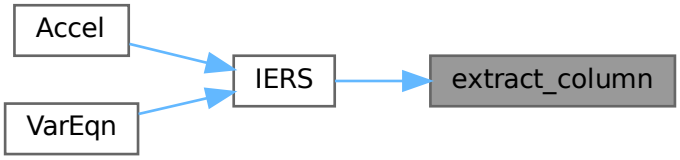
v	Matrix
i	es la columna tiene que ser >=1 && <=v.n_column

Returns

Matrix con la columna i-1 de v

Definition at line 401 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.6 extract_row()

```
Matrix & extract_row (
            Matrix & v,
            int i)
```

Extrae la fila i-1 de v y lo devuelve

Parameters

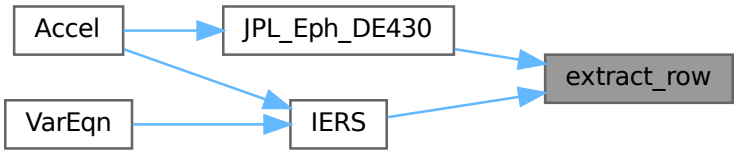
v	Matrix
i	es la fila tiene que ser >=1 && <=v.n_row

Returns

Matrix con la fila i-1 de v

Definition at line 389 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.7 extract_vector()

```
Matrix & extract_vector (
            Matrix & v,
            int start,
            int end)
```

Extrae del vector v desde la posicion start hasta end, incluidos

Parameters

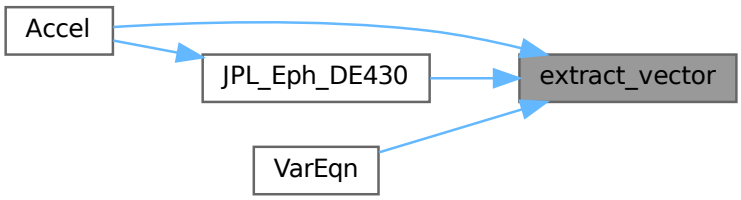
v	Matrix 1 x n
start	inicio del vector resultado
end	fin del vector resultado

Returns

Matrix 1 x (end - start + 1)

Definition at line 364 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.8 eye()

```
Matrix & eye (
            const int size)
```

Crea una Matrix identidad con tamaño size x size

Parameters

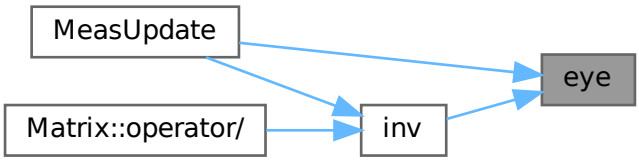
size	dimension de la matriz

Returns

una Matrix tamaño size x size

Definition at line 233 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.9 inv()

```
Matrix & inv (
            Matrix & m)
```

Crea una Matrix inversa de m, sin modificar m

Parameters

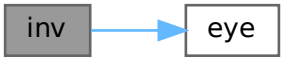
m	Matrix que tiene que ser cuadrada, es decir, con el mismo numero de columnas que filas

Returns

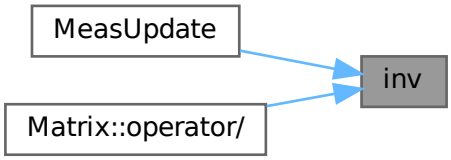
una Matrix inversa de m

Definition at line 267 of file matrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.2.10 norm()

```
double norm (
            Matrix & m)
```

Devulve la norma 2 de una Matrix que simula un vector

Parameters

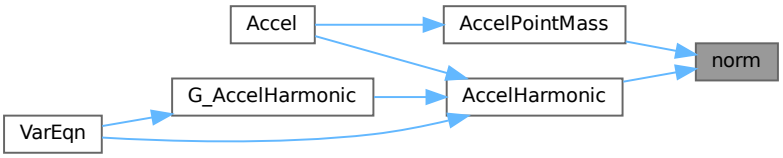
m	Matrix 1 x n_column

Returns

la norma 2 de m

Definition at line 333 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.11 operator<<()

```
ostream & operator<< (
            ostream & o,
            Matrix & m)
```

Definition at line 212 of file matrix.cpp.

5.37.2.12 transpose()

```
Matrix & transpose (
            Matrix & m)
```

Crea una Matrix traspuesta de m,sin modificar m

Parameters

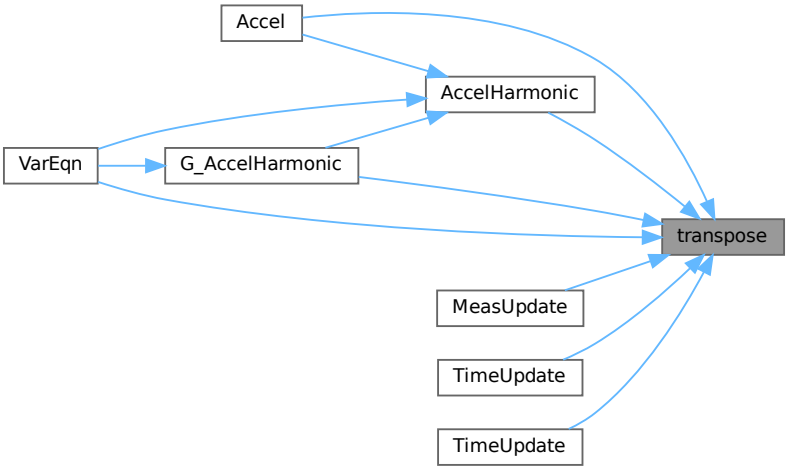
m	Matrix

Returns

una Matrix traspuesta de m

Definition at line 246 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.13 union_vector()

```
Matrix & union_vector (
            Matrix & v,
            Matrix & w)
```

Devuelve la matriz resultado de realizar la union entre v y w

Parameters

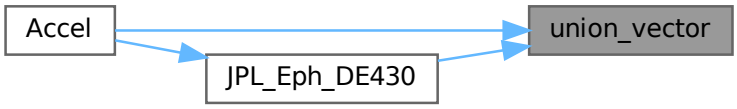
v	Matrix con tamaño 1 x n ,n pertenece a N
w	Matrix con tamaño 1 x n ,n pertenece a N

Returns

producto escalar de v x w

Definition at line 375 of file matrix.cpp.

Here is the caller graph for this function:



5.37.2.14 zeros() [1/2]

```
Matrix & zeros (
            const int n)
```

Crea una Matrix con todas sus componentes a 0

Parameters

n	numero de columnas que tiene la matriz

Returns

una Matrix tamaño 1 x n

Definition at line 323 of file matrix.cpp.

5.37.2.15 zeros() [2/2]

```
Matrix & zeros (
            const int n_row,
            const int n_column)
```

Crea una Matrix con todas sus componentes a 0

Parameters

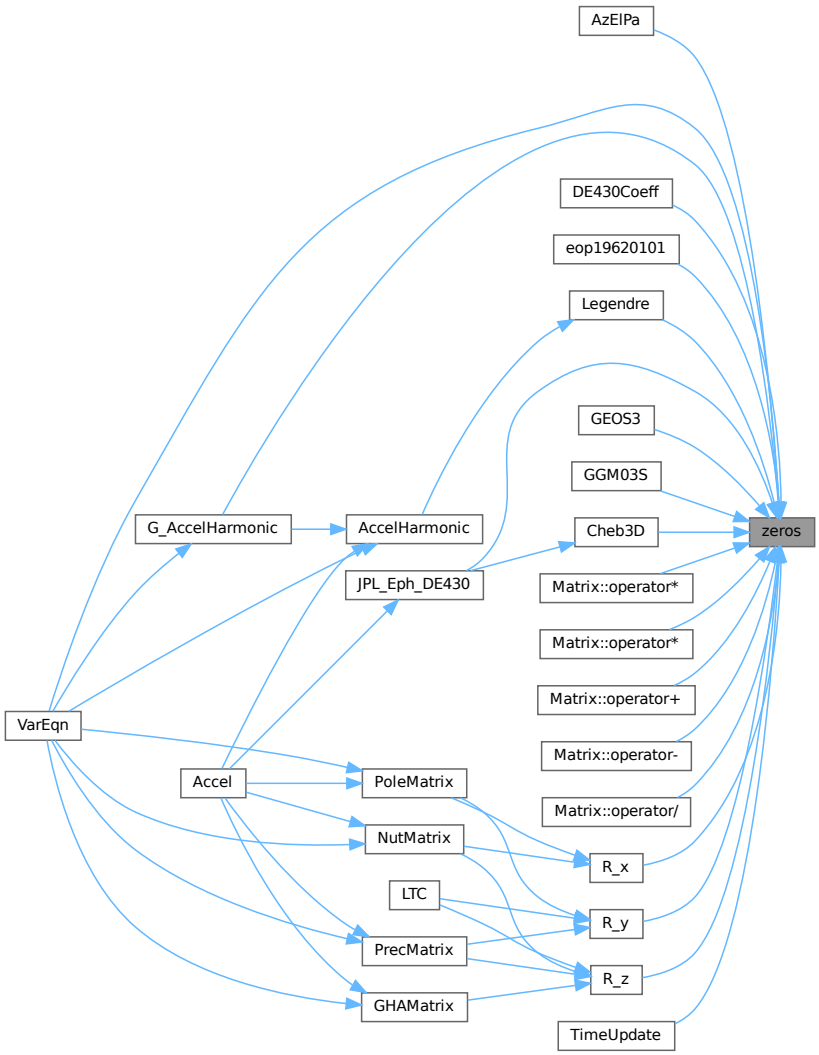
n_row	numero de filas que tiene la matriz
n_column	numero de columnas que tiene la matriz

Returns

una Matrix tamaño n_row x n_column

Definition at line 222 of file matrix.cpp.

Here is the caller graph for this function:



5.38 matrix.h

Go to the documentation of this file.

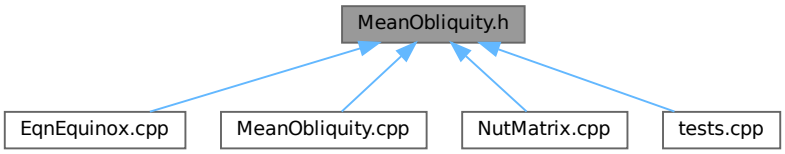
```
00001 #ifndef _MATRIX_
00002 #define _MATRIX_
00003
00004 #include <cstdio>
00005 #include <cstdlib>
00006 #include <cmath>
00007 #include <iostream>
00008
00009 using namespace std;
00010
00018 class Matrix {
00019 public:
00020     int n_row, n_column;
00021     double **data;
00022
00023     // Parameterized constructor
00027     Matrix();
00032     Matrix(const int n);
```

```
00038     Matrix(const int n_row, const int n_column);
00039
00040     // Member operators
00045     double& operator () (const int n);
00051     double& operator () (const int row, const int column);
00057     Matrix& operator + (Matrix &m);
00063     Matrix& operator - (Matrix &m);
00069     Matrix& operator * (Matrix &m);
00075     Matrix& operator / (Matrix &m);
00081     Matrix& operator = (Matrix &m);
00087     Matrix& operator + (double d);
00093     Matrix& operator - (double d);
00099     Matrix& operator * (double d);
00105     Matrix& operator / (double d);
00106
00107     // Non-member operators
00108     friend ostream& operator « (ostream &o, Matrix &m);
00109 };
00110
00111 // Operator overloading
00112 ostream& operator « (ostream &o, Matrix &m);
00113
00114
00115 // Methods
00122     Matrix& zeros(const int n_row, const int n_column);
00128     Matrix& eye(const int size);
00134     Matrix& transpose(Matrix &m);
00140     Matrix& inv(Matrix &m) ;
00146     Matrix& zeros(const int n);
00152     double norm(Matrix &m) ;
00159     double dot(Matrix &v,Matrix &w);
00166     Matrix& cross(Matrix &v,Matrix &w);
00174     Matrix& extract_vector(Matrix &v,int start,int end);
00175
00182     Matrix& union_vector(Matrix &v,Matrix &w);
00183
00184
00191     Matrix& extract_row(Matrix &v,int i);
00192
00193
00200     Matrix& extract_column(Matrix &v,int i);
00201
00202
00210     Matrix& assign_row(Matrix &v,Matrix &w,int i);
00211
00212
00220     Matrix& assign_column(Matrix &v,Matrix &w,int i);
00221
00222
00223 #endif
```

5.39 MeanObliquity.h File Reference

El archivo contiene la funcion MeanObliquity.

This graph shows which files directly or indirectly include this file:



Functions

- double MeanObliquity (double Mjd_TT)

5.39.1 Detailed Description

El archivo contiene la funcion MeanObliquity.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file MeanObliquity.h.

5.39.2 Function Documentation

5.39.2.1 MeanObliquity()

```
double MeanObliquity (
            double Mjd_TT)
```

Computes the mean obliquity of the ecliptic

Parameters

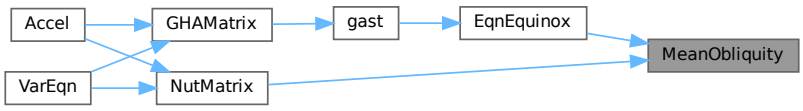
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

obliquity of the ecliptic [rad]

Definition at line 10 of file MeanObliquity.cpp.

Here is the caller graph for this function:



5.40 MeanObliquity.h

Go to the documentation of this file.

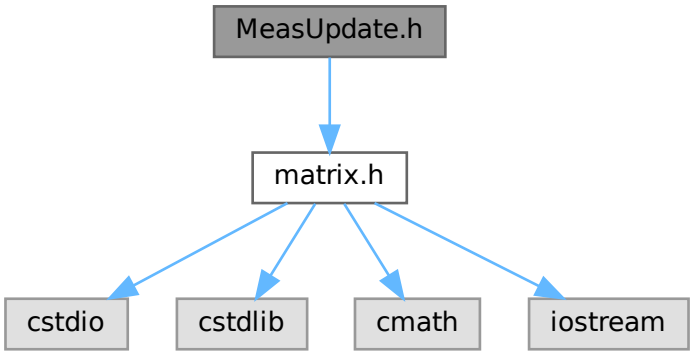
```
00001 #ifndef _MeanObliquity_
00002 #define _MeanObliquity_
00003
00004 using namespace std;
00005
00017      double MeanObliquity (double Mjd_TT);
00018 #endif
00019
00020
```

5.41 MeasUpdate.h File Reference

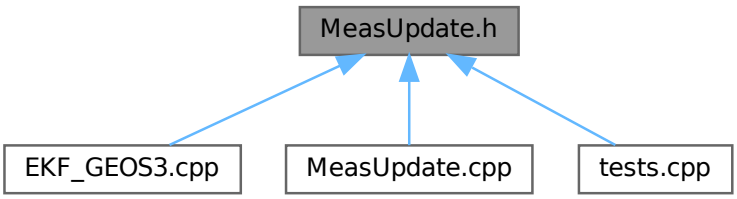
El archivo contiene la funcion MeasUpdate.

```
#include "matrix.h"
```

Include dependency graph for MeasUpdate.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< Matrix &, Matrix &, Matrix & > MeasUpdate (Matrix x, double z, double g, double s, Matrix G, Matrix P, int n)

5.41.1 Detailed Description

El archivo contiene la funcion MeasUpdate.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file MeasUpdate.h.

5.41.2 Function Documentation

5.41.2.1 MeasUpdate()

```
tuple< Matrix &, Matrix &, Matrix & > MeasUpdate (
            Matrix x,
            double z,
            double g,
            double s,
            Matrix G,
            Matrix P,
            int n)
```

Parameters

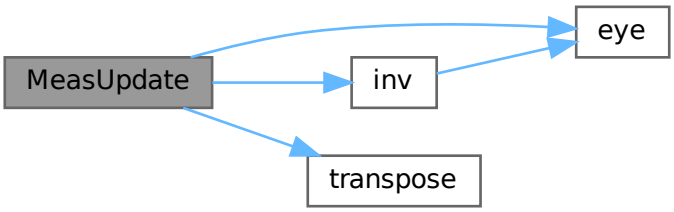
x	Matrix n∗1
z	double
g	double
s	double
G	Matrix 1∗n
P	Matrix n∗n
n	int

Returns

tupla de 3 Matrix

Definition at line 8 of file MeasUpdate.cpp.

Here is the call graph for this function:



5.42 MeasUpdate.h

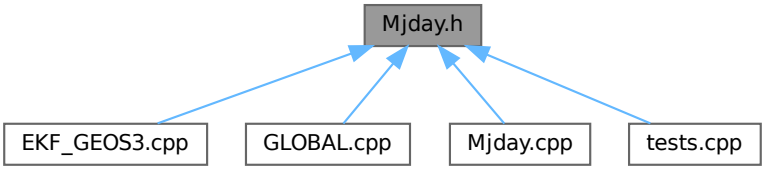
Go to the documentation of this file.

```
00001 #ifndef _MeasUpdate_
00002 #define _MeasUpdate_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00023     tuple<Matrix&,Matrix&,Matrix&> MeasUpdate(Matrix x, double z,double g,double s,Matrix G,Matrix P,
      int n);
00024 #endif
00025
00026
```

5.43 Mjday.h File Reference

El archivo contiene la funcion Mjday.

This graph shows which files directly or indirectly include this file:



Functions

- double Mjday (int yr, int mon, int day, int hr=0, int min=0, int sec=0)

5.43.1 Detailed Description

El archivo contiene la funcion Mjday.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Mjday.h.

5.43.2 Function Documentation

5.43.2.1 Mjday()

```
double Mjday (
            int yr,
            int mon,
            int day,
            int hr = 0,
            int min = 0,
            int sec = 0)
```

Parameters

year	- year
mon	- month
day	- day
hr	- universal time hour
min	- universal time min
sec	- universal time sec

Returns

Modified julian date

Definition at line 9 of file Mjday.cpp.

Here is the caller graph for this function:

5.44 Mjday.h

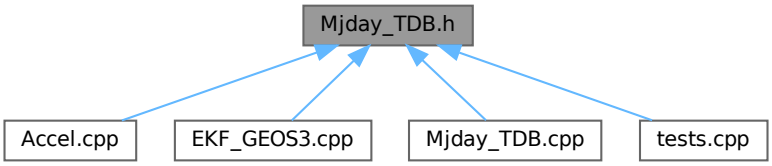
Go to the documentation of this file.

```
00001 #ifndef _Mjday_
00002 #define _Mjday_
00003
00004 using namespace std;
00005
00021     double Mjday(int yr, int mon,int day,int hr=0,int min=0,int sec=0);
00022 #endif
00023
00024
```

5.45 Mjday_TDB.h File Reference

El archivo contiene la funcion Mjday_TDB.

This graph shows which files directly or indirectly include this file:



Functions

- double Mjday_TDB (double Mjd_TT)

5.45.1 Detailed Description

El archivo contiene la funcion Mjday_TDB.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Mjday_TDB.h.

5.45.2 Function Documentation

5.45.2.1 Mjday_TDB()

```
double Mjday_TDB (
            double Mjd_TT)
```

Parameters

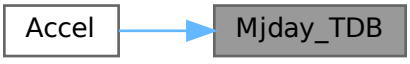
Mjd_TT	- Modified julian date (TT)

Returns

Modified julian date (TDB)

Definition at line 10 of file Mjday_TDB.cpp.

Here is the caller graph for this function:



5.46 Mjday_TDB.h

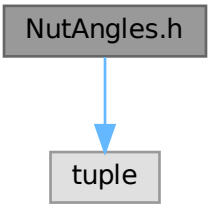
Go to the documentation of this file.

```
00001 #ifndef _Mjday_TDB_
00002 #define _Mjday_TDB_
00003
00004 using namespace std;
00005
00016     double Mjday_TDB(double Mjd_TT);
00017 #endif
00018
00019
```

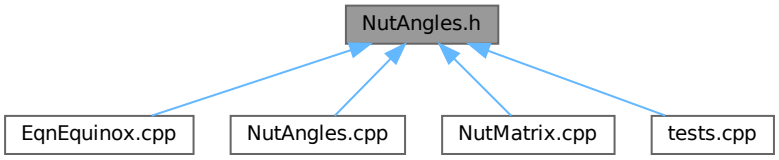
5.47 NutAngles.h File Reference

El archivo contiene la funcion NutAngles.

`#include <tuple>`
Include dependency graph for NutAngles.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< double, double > NutAngles (double Mjd_TT)

5.47.1 Detailed Description

El archivo contiene la funcion NutAngles.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file NutAngles.h.

5.47.2 Function Documentation

5.47.2.1 NutAngles()

```
tuple< double, double > NutAngles (
            double Mjd_TT)
```

Nutation in longitude and obliquity

Parameters

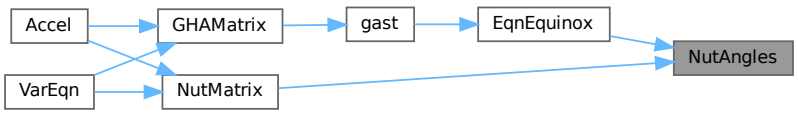
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

tupla < dpsi,deps> Nutation Angles

Definition at line 11 of file NutAngles.cpp.

Here is the caller graph for this function:



5.48 NutAngles.h

Go to the documentation of this file.

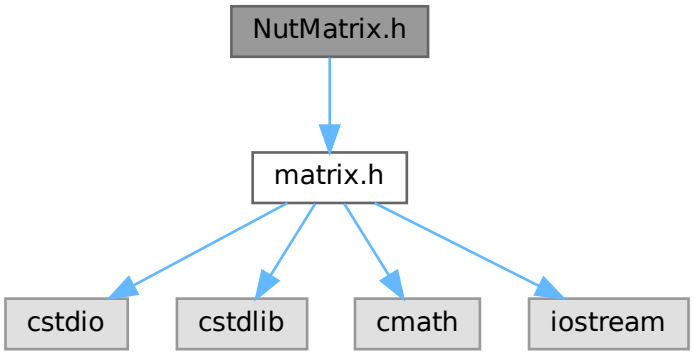
```
00001 #ifndef _NutAngles_
00002 #define _NutAngles_
00003 #include <tuple>
00004 using namespace std;
00005
00017     tuple<double,double> NutAngles (double Mjd_TT);
00018 #endif
00019
00020
```

5.49 NutMatrix.h File Reference

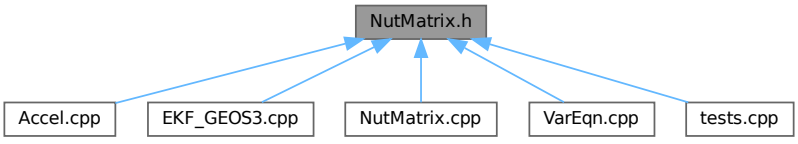
El archivo contiene la funcion NutMatrix.

```
#include "matrix.h"
```

Include dependency graph for NutMatrix.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & NutMatrix (double Mjd_TT)

5.49.1 Detailed Description

El archivo contiene la funcion NutMatrix.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file NutMatrix.h.

5.49.2 Function Documentation

5.49.2.1 NutMatrix()

```
Matrix & NutMatrix (
            double Mjd_TT)
```

Transformation from mean to true equator and equinox

Parameters

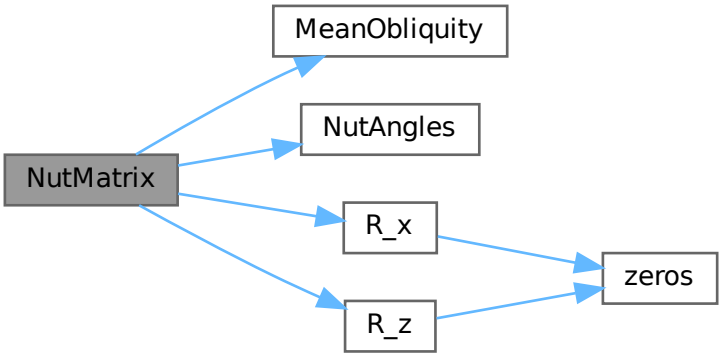
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

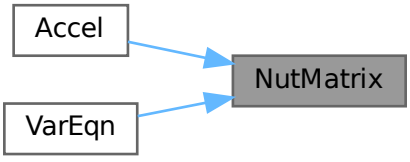
NutMat Nutation matrix

Definition at line 13 of file NutMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.50 NutMatrix.h

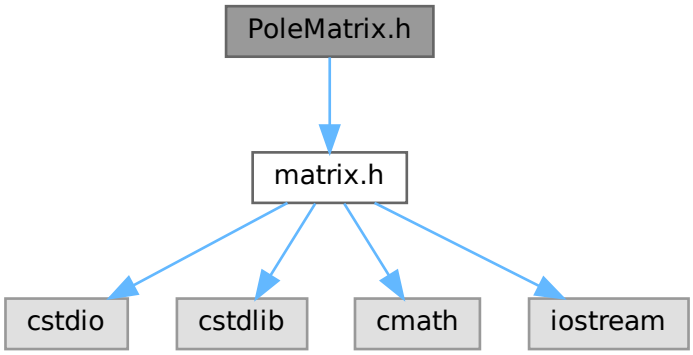
Go to the documentation of this file.

```
00001 #ifndef _NutMatrix_
00002 #define _NutMatrix_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00018     Matrix& NutMatrix (double Mjd_TT);
00019 #endif
00020
00021
```

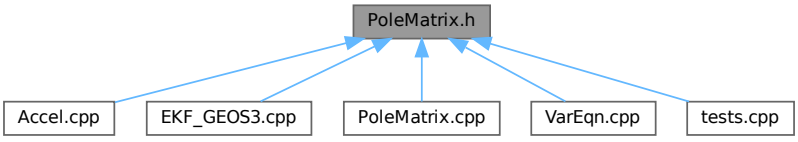
5.51 PoleMatrix.h File Reference

El archivo contiene la funcion PoleMatrix.

`#include "matrix.h"`
Include dependency graph for PoleMatrix.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & PoleMatrix (double xp, double yp)

5.51.1 Detailed Description

El archivo contiene la funcion PoleMatrix.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file PoleMatrix.h.

5.51.2 Function Documentation

5.51.2.1 PoleMatrix()

```
Matrix & PoleMatrix (
            double xp,
            double yp)
```

Transformation from pseudo Earth-fixed to Earth-fixed coordinates for a given date

Parameters

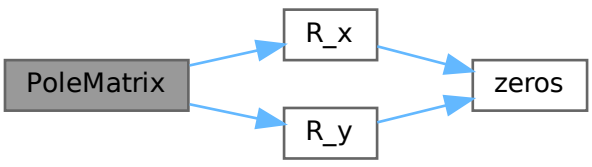
Pole	coordinte(xp,yp)

Returns

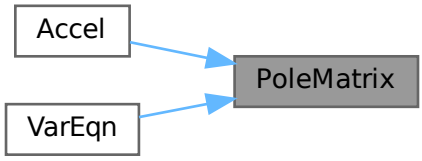
PoleMat Pole matrix

Definition at line 11 of file PoleMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.52 PoleMatrix.h

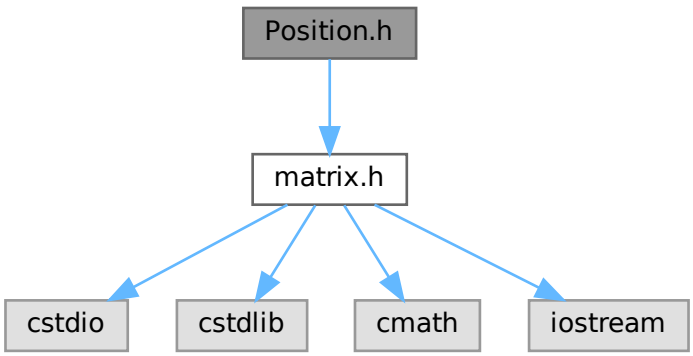
Go to the documentation of this file.

```
00001 #ifndef _PoleMatrix_
00002 #define _PoleMatrix_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00018     Matrix& PoleMatrix (double xp,double yp);
00019 #endif
00020
00021
```

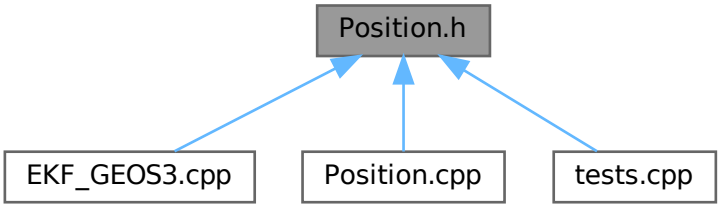
5.53 Position.h File Reference

El archivo contiene la funcion Position.

`#include "matrix.h"`
Include dependency graph for Position.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & Position (double lon, double lat, double h)

5.53.1 Detailed Description

El archivo contiene la funcion Position.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Position.h.

5.53.2 Function Documentation

5.53.2.1 Position()

```
Matrix & Position (
            double lon,
            double lat,
            double h)
```

Parameters

lon	longitude [rad]
lat	latitude [rad]
h	altitude [m]

Returns

Position vector (r [m]) from geodetic coordinates (Longitude [rad],latitude [rad], altitude [m])

Definition at line 10 of file Position.cpp.

5.54 Position.h

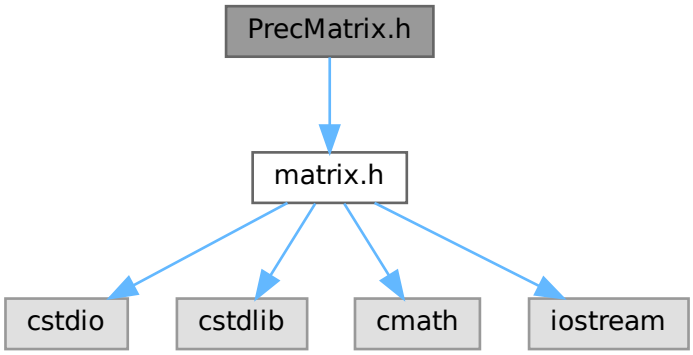
Go to the documentation of this file.

```
00001 #ifndef _Position_
00002 #define _Position_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00019     Matrix& Position(double lon,double lat,double h);
00020 #endif
00021
00022
```

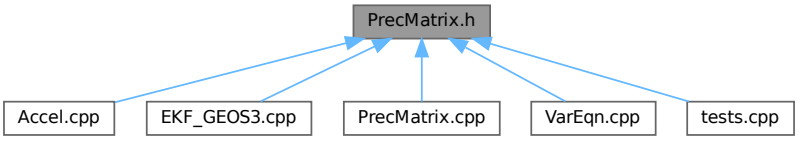
5.55 PrecMatrix.h File Reference

El archivo contiene la funcion PrecMatrix.

`#include "matrix.h"`
Include dependency graph for PrecMatrix.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & PrecMatrix (double Mjd_1, double Mjd_2)

5.55.1 Detailed Description

El archivo contiene la funcion PrecMatrix.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file PrecMatrix.h.

5.55.2 Function Documentation

5.55.2.1 PrecMatrix()

```
Matrix & PrecMatrix (
            double Mjd_1,
            double Mjd_2)
```

Precession transformation of equatorial coordinates

Parameters

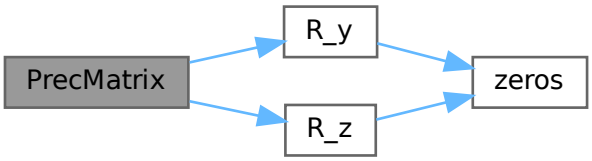
Mjd_1	Epoch given (Modified Julian Date TT)
MjD_2	Epoch to precess to (Modified Julian Date TT)

Returns

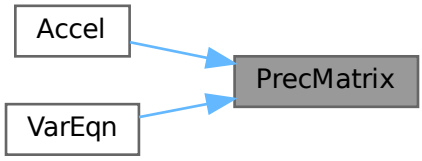
PrecMat Precession transformation matrix

Definition at line 13 of file PrecMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.56 PrecMatrix.h

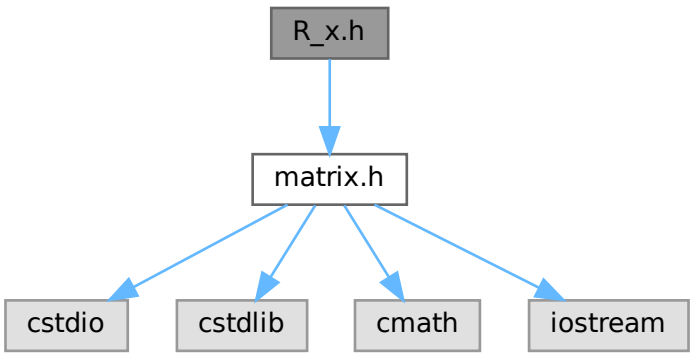
Go to the documentation of this file.

```
00001 #ifndef _PrecMatrix_
00002 #define _PrecMatrix_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00019     Matrix& PrecMatrix (double Mjd_1, double Mjd_2);
00020 #endif
00021
00022
```

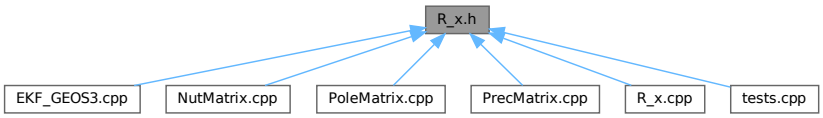
5.57 R_x.h File Reference

El archivo contiene la funcion R_x.

`#include "matrix.h"`
Include dependency graph for R_x.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & R_x (double angle)

5.57.1 Detailed Description

El archivo contiene la funcion R_x.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_x.h.

5.57.2 Function Documentation

5.57.2.1 R_x()

```
Matrix & R_x (
            double angle)
```

Parameters

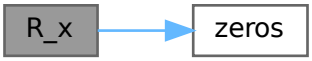
angle	angulo de rotacion

Returns

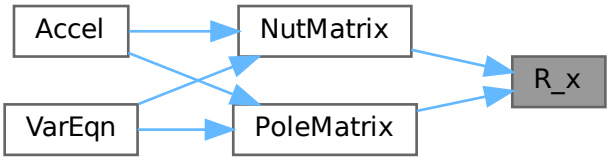
Matrix resultado

Definition at line 9 of file R_x.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.58 R_x.h

Go to the documentation of this file.

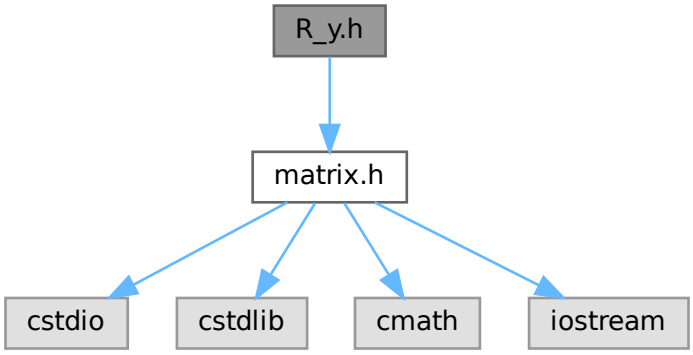
```
00001 #ifndef _R_x_
00002 #define _R_x_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00017     Matrix& R_x(double angle);
00018 #endif
00019
00020
```

5.59 R_y.h File Reference

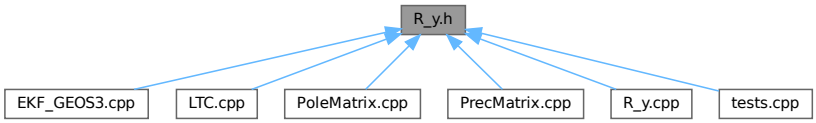
El archivo contiene la funcion R_y.

```
#include "matrix.h"
```

Include dependency graph for R_y.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & R_y (double angle)

5.59.1 Detailed Description

El archivo contiene la funcion R_y.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_y.h.

5.59.2 Function Documentation

5.59.2.1 R_y()

```
Matrix & R_y (
            double angle)
```

Parameters

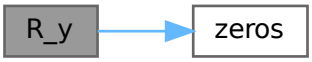
angle	angulo de rotacion

Returns

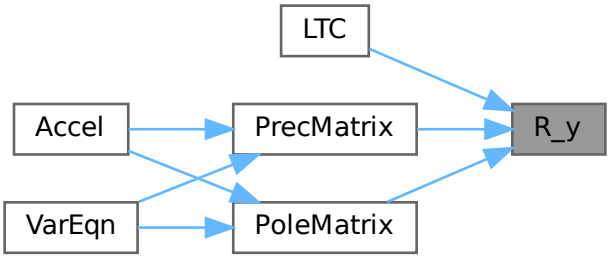
Matrix resultado

Definition at line 9 of file R_y.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.60 R_y.h

Go to the documentation of this file.

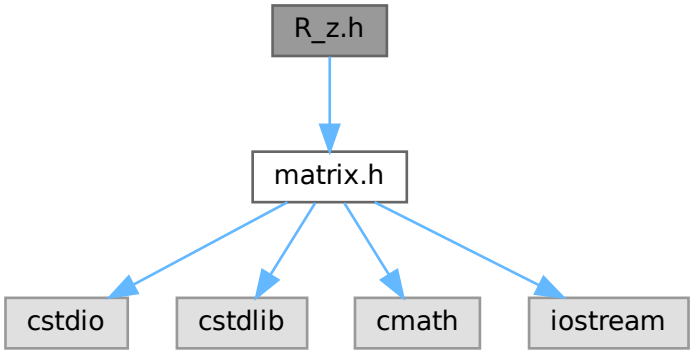
```
00001 #ifndef _R_y_
00002 #define _R_y_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00017     Matrix& R_y(double angle);
00018 #endif
00019
00020
```

5.61 R_z.h File Reference

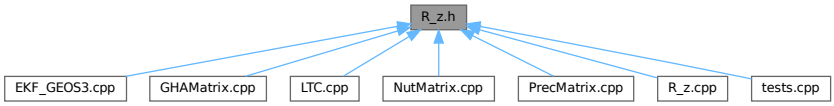
El archivo contiene la funcion R_z.

```
#include "matrix.h"
```

Include dependency graph for R_z.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & R_z (double angle)

5.61.1 Detailed Description

El archivo contiene la funcion R_z.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_z.h.

5.61.2 Function Documentation

5.61.2.1 R_z()

```
Matrix & R_z (
            double angle)
```

Parameters

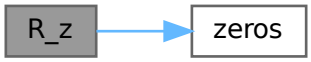
angle	angulo de rotacion

Returns

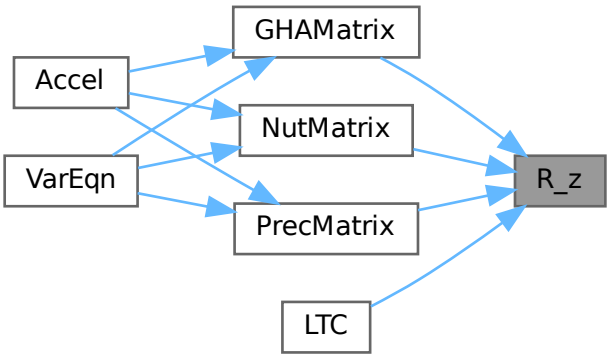
Matrix resultado

Definition at line 9 of file R_z.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.62 R_z.h

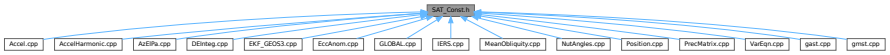
Go to the documentation of this file.

```
00001 #ifndef _R_z_
00002 #define _R_z_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00017     Matrix& R_z(double angle);
00018 #endif
00019
00020
```

5.63 SAT_Const.h File Reference

El archivo contiene SAT_Const.

This graph shows which files directly or indirectly include this file:



Variables

- const double pi =3.14159265358979323846264338327950288419716939937510582097494
- const double eps =2.22044604925031e-16
- const double pi2 = 2∗pi
- const double Rad = pi/180
- const double Deg = 180/pi
- const double Arcs = 3600∗180/pi
- const double MJD_J2000 = 51544.5
- const double T_B1950 = -0.500002108
- const double c_light = 299792458.000000000
- const double AU = 149597870700.000000
- const double R_Earth = 6378.1363e3
- const double f_Earth = 1/298.257223563
- const double R_Sun = 696000e3
- const double R_Moon = 1738e3
- const double omega_Earth = 15.04106717866910/3600∗Rad
- const double GM_Earth = 398600.435436e9
- const double GM_Sun = 132712440041.939400e9
- const double GM_Moon = GM_Earth/81.30056907419062
- const double GM_Mercury = 22031.780000e9
- const double GM_Venus = 324858.592000e9
- const double GM_Mars = 42828.375214e9
- const double GM_Jupiter = 126712764.800000e9
- const double GM_Saturn = 37940585.200000e9
- const double GM_Uranus = 5794548.600000e9
- const double GM_Neptune = 6836527.100580e9
- const double GM_Pluto = 977.0000000000009e9
- const double P_Sol = 1367/c_light

5.63.1 Detailed Description

El archivo contiene SAT_Const.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file SAT_Const.h.

5.63.2 Variable Documentation

5.63.2.1 Arcs

```
const double Arcs = 3600*180/pi
```

Definition at line 17 of file SAT_Const.h.

5.63.2.2 AU

```
const double AU = 149597870700.000000
```

Definition at line 23 of file SAT_Const.h.

5.63.2.3 c_light

```
const double c_light = 299792458.000000000
```

Definition at line 22 of file SAT_Const.h.

5.63.2.4 Deg

```
const double Deg = 180/pi
```

Definition at line 16 of file SAT_Const.h.

5.63.2.5 eps

```
const double eps =2.22044604925031e-16
```

Definition at line 12 of file SAT_Const.h.

5.63.2.6 f_Earth

```
const double f_Earth = 1/298.257223563
```

Definition at line 29 of file SAT_Const.h.

5.63.2.7 GM_Earth

```
const double GM_Earth = 398600.435436e9
```

Definition at line 37 of file SAT_Const.h.

5.63.2.8 GM_Jupiter

```
const double GM_Jupiter = 126712764.800000e9
```

Definition at line 43 of file SAT_Const.h.

5.63.2.9 GM_Mars

```
const double GM_Mars = 42828.375214e9
```

Definition at line 42 of file SAT_Const.h.

5.63.2.10 GM_Mercury

```
const double GM_Mercury = 22031.780000e9
```

Definition at line 40 of file SAT_Const.h.

5.63.2.11 GM_Moon

```
const double GM_Moon = GM_Earth/81.30056907419062
```

Definition at line 39 of file SAT_Const.h.

5.63.2.12 GM_Neptune

```
const double GM_Neptune = 6836527.100580e9
```

Definition at line 46 of file SAT_Const.h.

5.63.2.13 GM_Pluto

```
const double GM_Pluto = 977.0000000000009e9
```

Definition at line 47 of file SAT_Const.h.

5.63.2.14 GM_Saturn

```
const double GM_Saturn = 37940585.200000e9
```

Definition at line 44 of file SAT_Const.h.

5.63.2.15 GM_Sun

```
const double GM_Sun = 132712440041.939400e9
```

Definition at line 38 of file SAT_Const.h.

5.63.2.16 GM_Uranus

```
const double GM_Uranus = 5794548.600000e9
```

Definition at line 45 of file SAT_Const.h.

5.63.2.17 GM_Venus

```
const double GM_Venus = 324858.592000e9
```

Definition at line 41 of file SAT_Const.h.

5.63.2.18 MJD_J2000

```
const double MJD_J2000 = 51544.5
```

Definition at line 20 of file SAT_Const.h.

5.63.2.19 omega_Earth

```
const double omega_Earth = 15.04106717866910/3600*Rad
```

Definition at line 34 of file SAT_Const.h.

5.63.2.20 P_Sol

```
const double P_Sol = 1367/c_light
```

Definition at line 50 of file SAT_Const.h.

5.63.2.21 pi

```
const double pi =3.14159265358979323846264338327950288419716939937510582097494
```

Definition at line 11 of file SAT_Const.h.

5.63.2.22 pi2

```
const double pi2 = 2*pi
```

Definition at line 14 of file SAT_Const.h.

5.63.2.23 R_Earth

```
const double R_Earth = 6378.1363e3
```

Definition at line 28 of file SAT_Const.h.

5.63.2.24 R_Moon

```
const double R_Moon = 1738e3
```

Definition at line 31 of file SAT_Const.h.

5.63.2.25 R_Sun

```
const double R_Sun = 696000e3
```

Definition at line 30 of file SAT_Const.h.

5.63.2.26 Rad

```
const double Rad = pi/180
```

Definition at line 15 of file SAT_Const.h.

5.63.2.27 T_B1950

```
const double T_B1950 = -0.500002108
```

Definition at line 21 of file SAT_Const.h.

5.64 SAT_Const.h

Go to the documentation of this file.

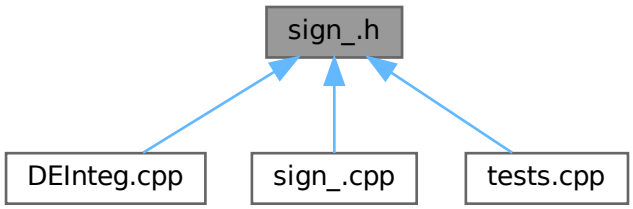
```
00001 #ifndef _SAT_Const_
00002 #define _SAT_Const_
00003
00004
00011      const double pi=3.14159265358979323846264338327950288419716939937510582097494;
00012      const double eps=2.22044604925031e-16;
00013     // Mathematical constants
00014     const double pi2       = 2*pi;               // 2pi
00015     const double Rad       = pi/180;             // Radians per degree
00016     const double Deg       = 180/pi;             // Degrees per radian
00017     const double Arcs      = 3600*180/pi;        // Arcseconds per radian
00018
00019     // General
00020     const double MJD_J2000 = 51544.5;            // Modified Julian Date of J2000
00021     const double T_B1950   = -0.500002108;       // Epoch B1950
00022     const double c_light   = 299792458.000000000; // Speed of light  [m/s]; DE430
00023     const double AU        = 149597870700.000000; // Astronomical unit [m]; DE430
00024
00025     // Physical parameters of the Earth, Sun and Moon
00026
00027     // Equatorial radius and flattening
00028     const double R_Earth   = 6378.1363e3;      // Earth's radius [m]; DE430
00029     const double f_Earth   = 1/298.257223563;  // Flattening; WGS-84
00030     const double R_Sun     = 696000e3;         // Sun's radius [m]; DE430
00031     const double R_Moon    = 1738e3;           // Moon's radius [m]; DE430
00032
00033     // Earth rotation (derivative of GMST at J2000; differs from inertial period by precession)
00034     const double omega_Earth = 15.04106717866910/3600*Rad;   // [rad/s]; WGS-84
00035
00036     // Gravitational coefficients
00037     const double GM_Earth    = 398600.435436e9;                 // [m^3/s^2]; DE430
00038     const double GM_Sun      = 132712440041.939400e9;           // [m^3/s^2]; DE430
00039     const double GM_Moon     = GM_Earth/81.30056907419062;      // [m^3/s^2]; DE430
00040     const double GM_Mercury  = 22031.780000e9;                  // [m^3/s^2]; DE430
```

```
00041     const double GM_Venus    = 324858.592000e9;                 // [m^3/s^2]; DE430
00042     const double GM_Mars     = 42828.375214e9;                  // [m^3/s^2]; DE430
00043     const double GM_Jupiter  = 126712764.800000e9;              // [m^3/s^2]; DE430
00044     const double GM_Saturn   = 37940585.200000e9;               // [m^3/s^2]; DE430
00045     const double GM_Uranus   = 5794548.600000e9;                // [m^3/s^2]; DE430
00046     const double GM_Neptune  = 6836527.100580e9;                // [m^3/s^2]; DE430
00047     const double GM_Pluto    = 977.0000000000009e9;             // [m^3/s^2]; DE430
00048
00049     // Solar radiation pressure at 1 AU
00050     const double P_Sol       = 1367/c_light; // [N/m^2] (~1367 W/m^2); IERS 96
00051
00052 #endif
00053
00054
```

5.65 sign_.h File Reference

El archivo contiene la funcion sign_.

This graph shows which files directly or indirectly include this file:



Functions

- double sign_ (double a, double b)

5.65.1 Detailed Description

El archivo contiene la funcion sign_.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file sign_.h.

5.65.2 Function Documentation

5.65.2.1 sign_()

```
double sign_ (
            double a,
            double b)
```

Parameters

a	double
b	double

Returns

absolute value of a with sign of b

Definition at line 10 of file sign_.cpp.

5.66 sign_.h

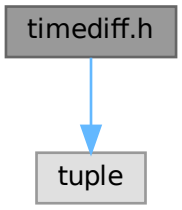
Go to the documentation of this file.

```
00001 #ifndef _sign__
00002 #define _sign__
00003
00004 using namespace std;
00005
00017     double sign_(double a, double b);
00018 #endif
00019
00020
```

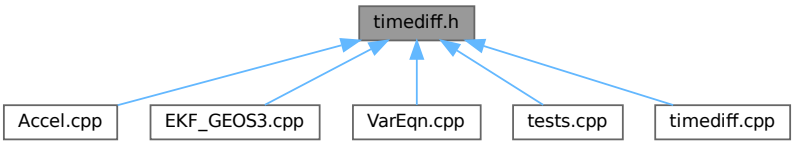
5.67 timediff.h File Reference

El archivo contiene la funcion timediff.

`#include <tuple>`
Include dependency graph for timediff.h:



This graph shows which files directly or indirectly include this file:



Functions

- tuple< double, double, double, double, double > timediff (double UT1_UTC, double TAI_UTC)

5.67.1 Detailed Description

El archivo contiene la funcion timediff.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file timediff.h.

5.67.2 Function Documentation

5.67.2.1 timediff()

```
tuple< double, double, double, double, double > timediff (
            double UT1_UTC,
            double TAI_UTC)
```

Time differences [s]

Parameters

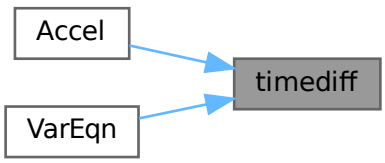
UT1_UTC	double
TAI_UTC	double

Returns

tupla <UT1_TAI, UTC_GPS, UT1_GPS, TT_UTC, GPS_UTC>

Definition at line 9 of file timediff.cpp.

Here is the caller graph for this function:



5.68 timediff.h

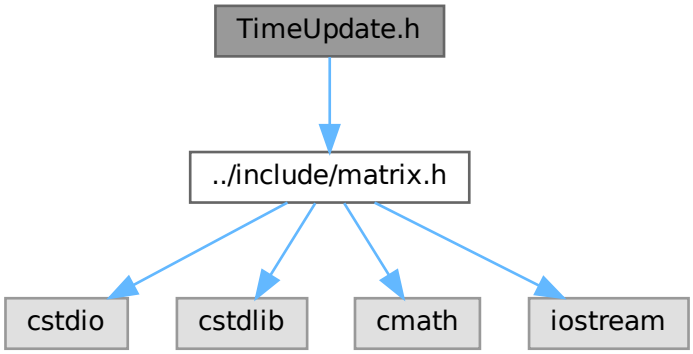
Go to the documentation of this file.

```
00001 #ifndef _timediff_
00002 #define _timediff_
00003 #include <tuple>
00004 using namespace std;
00005
00018     tuple<double,double,double,double,double> timediff(double UT1_UTC,double TAI_UTC);
00019 #endif
00020
00021
```

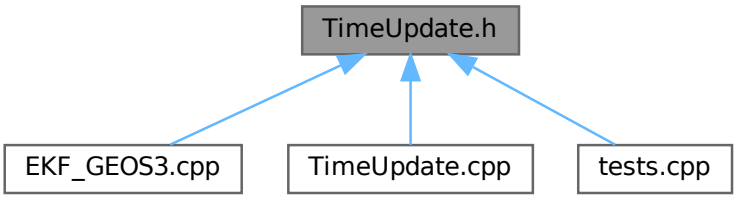
5.69 TimeUpdate.h File Reference

El archivo contiene la funcion TimeUpdate.

`#include "../include/matrix.h"`
Include dependency graph for TimeUpdate.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & TimeUpdate (Matrix P, Matrix Phi, Matrix Qdt)
- Matrix & TimeUpdate (Matrix P, Matrix Phi)

5.69.1 Detailed Description

El archivo contiene la funcion TimeUpdate.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file TimeUpdate.h.

5.69.2 Function Documentation

5.69.2.1 TimeUpdate() [1/2]

```
Matrix & TimeUpdate (
            Matrix P,
            Matrix Phi)
```

Parameters

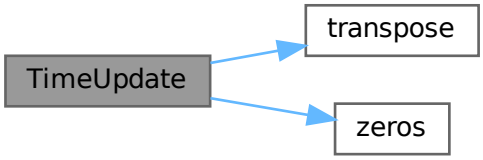
P	Matrix
Phi	Matrix

Returns

Matrix

Definition at line 15 of file TimeUpdate.cpp.

Here is the call graph for this function:



5.69.2.2 TimeUpdate() [2/2]

```
Matrix & TimeUpdate (
            Matrix P,
            Matrix Phi,
            Matrix Qdt)
```

Parameters

P	Matrix
Phi	Matrix
Qdt	Matrix

Returns

Matrix

Definition at line 9 of file TimeUpdate.cpp.

Here is the call graph for this function:

5.70 TimeUpdate.h

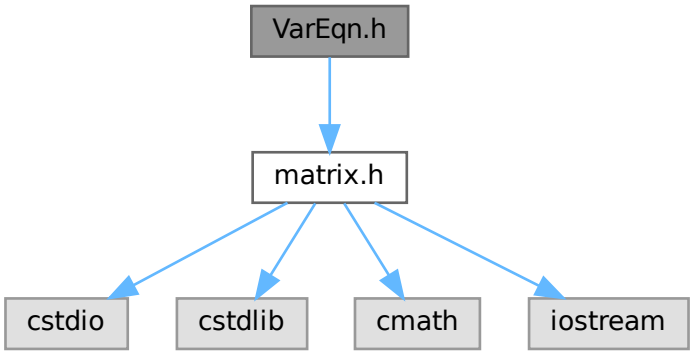
Go to the documentation of this file.

```
00001 #ifndef _TimeUpdate_
00002 #define _TimeUpdate_
00003 using namespace std;
00004 #include "../include/matrix.h"
00005
00018     Matrix& TimeUpdate(Matrix P,Matrix Phi,Matrix Qdt);
00019
00025     Matrix& TimeUpdate(Matrix P,Matrix Phi);
00026 #endif
00027
00028
```

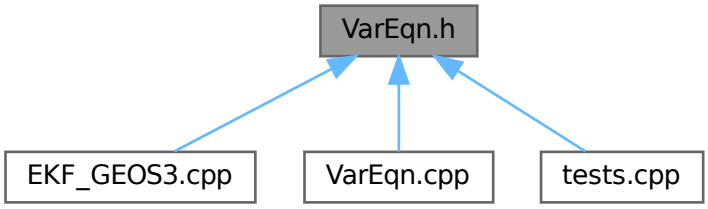
5.71 VarEqn.h File Reference

El archivo contiene la funcion VarEqn.

```
#include "matrix.h"
```
Include dependency graph for VarEqn.h:



This graph shows which files directly or indirectly include this file:



Functions

- Matrix & VarEqn (double x, Matrix yPhi)

5.71.1 Detailed Description

El archivo contiene la funcion VarEqn.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file VarEqn.h.

5.71.2 Function Documentation

5.71.2.1 VarEqn()

```
Matrix & VarEqn (
            double x,
            Matrix yPhi)
```

Computes the variational equations, i.e. the derivative of the state vector and the state transition matrix

Parameters

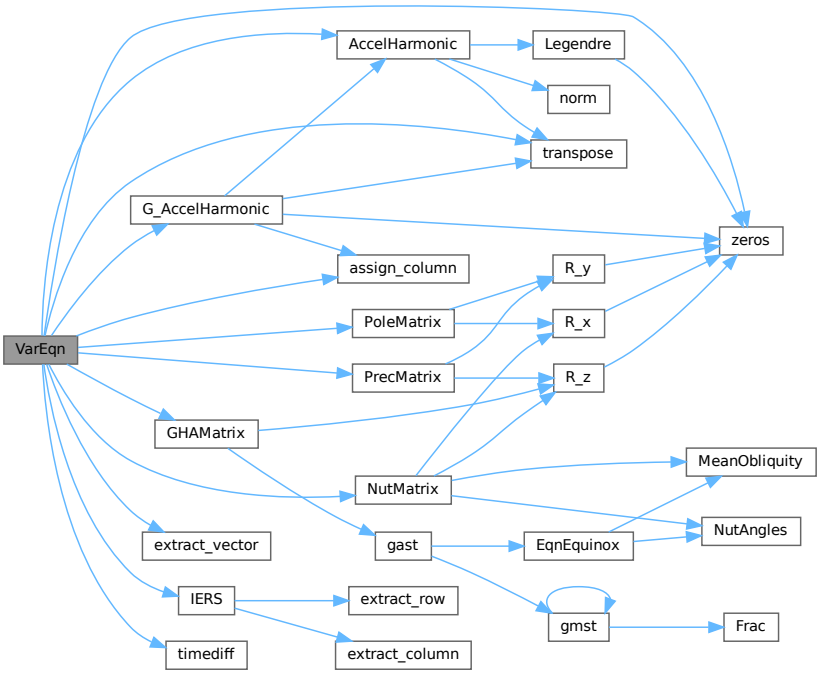
x	Time since epoch in [s]
yPhi	(6+36)-dim vector comprising the state vector (y) and the state transition matrix (Phi) in column wise storage order

Returns

yPhip Derivative of yPhi

Definition at line 18 of file VarEqn.cpp.

Here is the call graph for this function:



5.72 VarEqn.h

Go to the documentation of this file.

```
00001 #ifndef _VarEqn_
00002 #define _VarEqn_
00003 #include "matrix.h"
00004
00005 using namespace std;
00006
00019     Matrix& VarEqn(double x, Matrix yPhi);
00020 #endif
00021
00022
```

5.73 Accel.cpp File Reference

El archivo contiene las implementaciones de Accel.h.

```
#include "../include/Accel.h"
#include "../include/PrecMatrix.h"
#include "../include/NutMatrix.h"
#include "../include/IERS.h"
#include "../include/timediff.h"
#include "../include/PoleMatrix.h"
#include "../include/AccelHarmonic.h"
#include "../include/GHAMatrix.h"
#include "../include/JPL_Eph_DE430.h"
#include "../include/GLOBAL.h"
#include "../include/AccelPointMass.h"
#include "../include/Mjday_TDB.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for Accel.cpp:

Functions

- Matrix & Accel (double x, Matrix Y)

5.73.1 Detailed Description

El archivo contiene las implementaciones de Accel.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Accel.cpp.

5.73.2 Function Documentation

5.73.2.1 Accel()

```
Matrix & Accel (
            double x,
            Matrix Y)
```

Computes the acceleration of an Earth orbiting satellite due to

- the Earth's harmonic gravity field,
- the gravitational perturbations of the Sun and Moon
- the solar radiation pressure and
- the atmospheric drag

Parameters

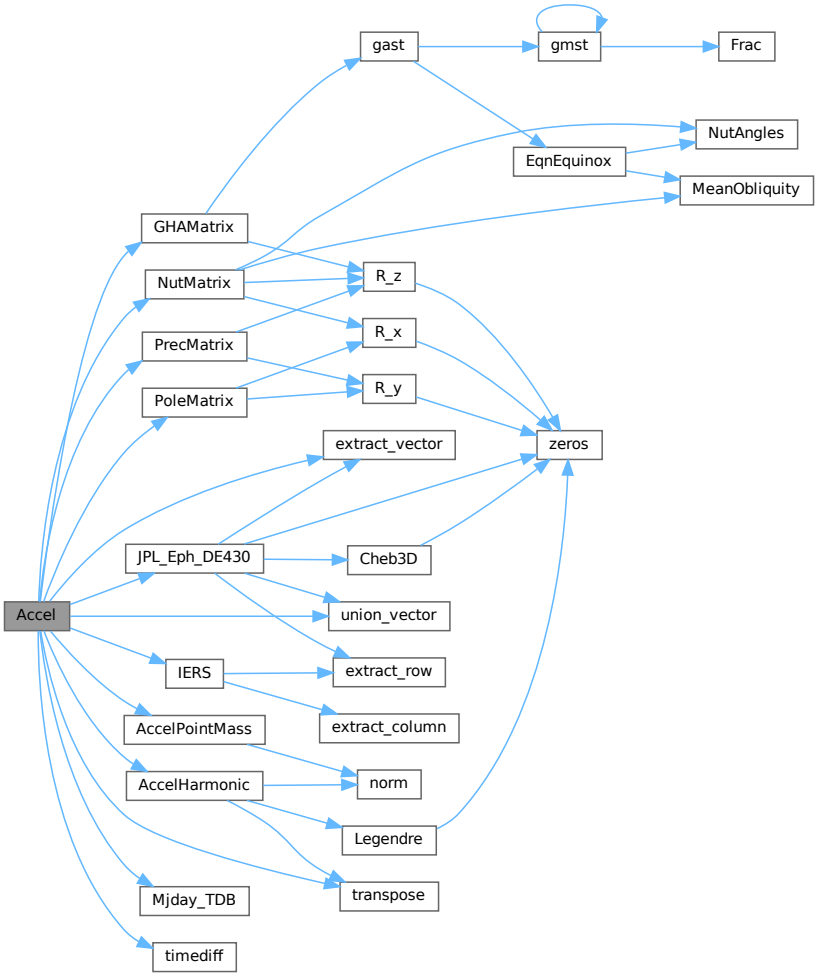
Mjd_TT	Terrestrial Time (Modified Julian Date)
Y	Satellite state vector in the ICRF/EME2000 system

Returns

dY Acceleration (a=d^2r/dt^2) in the ICRF/EME2000 system

Definition at line 20 of file Accel.cpp.

Here is the call graph for this function:



5.74 Accel.cpp

Go to the documentation of this file.

```
00001 #include "../include/Accel.h"
00002 #include "../include/PrecMatrix.h"
00003 #include "../include/NutMatrix.h"
00004 #include "../include/IERS.h"
00005 #include "../include/timediff.h"
00006 #include "../include/PoleMatrix.h"
00007 #include "../include/AccelHarmonic.h"
00008 #include "../include/GHAMatrix.h"
00009 #include "../include/JPL_Eph_DE430.h"
00010 #include "../include/GLOBAL.h"
00011 #include "../include/AccelPointMass.h"
00012 #include "../include/Mjday_TDB.h"
00013 #include "../include/SAT_Const.h"
```

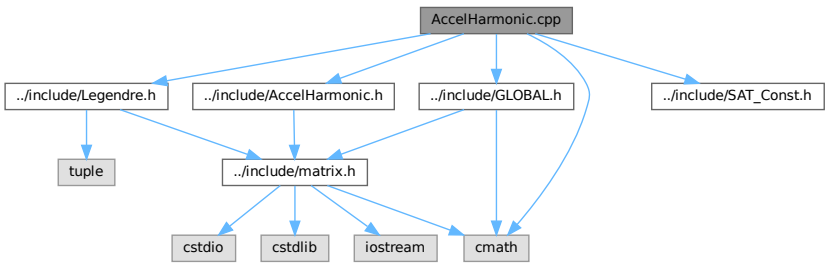
```
00020         Matrix& Accel(double x,Matrix Y){
00021             if(Y.n_row<Y.n_column){
00022                 Y=transpose(Y);
00023             }
00024     auto [x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC] = IERS(eopdata,AuxParam.Mjd_UTC
    + x/86400,'l');
00025     auto [UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);
00026     long double Mjd_UT1 = AuxParam.Mjd_UTC + x/86400 + UT1_UTC/86400;
00027     long double Mjd_TT = AuxParam.Mjd_UTC + x/86400 + TT_UTC/86400;
00028 
00029     Matrix P = PrecMatrix(MJD_J2000,Mjd_TT);
00030     Matrix N = NutMatrix(Mjd_TT);
00031     Matrix T = N * P;
00032     Matrix E = PoleMatrix(x_pole,y_pole) * GHAMatrix(Mjd_UT1) * T;
00033 
00034     long double MJD_TDB = Mjday_TDB(Mjd_TT);
00035     auto [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus,r_Neptune,r_Pluto,r_Moon,r_Sun]
    = JPL_Eph_DE430(MJD_TDB);
00036 
00037     // Acceleration due to harmonic gravity field
00038     Matrix a = AccelHarmonic(transpose(extract_vector(Y,1,3)), E, AuxParam.n, AuxParam.m);
00039 
00040     // Luni-solar perturbations
00041     if (AuxParam.sun){
00042             a = a + AccelPointMass(extract_vector(Y,1,3),r_Sun,GM_Sun);}
00043 
00044     if (AuxParam.moon){
00045             a = a + AccelPointMass(extract_vector(Y,1,3),r_Moon,GM_Moon);}
00046     // Planetary perturbations
00047     if (AuxParam.planets){
00048             a = a + AccelPointMass(extract_vector(Y,1,3),r_Mercury,GM_Mercury);
00049             a = a + AccelPointMass(extract_vector(Y,1,3),r_Venus,GM_Venus);
00050             a = a + AccelPointMass(extract_vector(Y,1,3),r_Mars,GM_Mars);
00051             a = a + AccelPointMass(extract_vector(Y,1,3),r_Jupiter,GM_Jupiter);
00052             a = a + AccelPointMass(extract_vector(Y,1,3),r_Saturn,GM_Saturn);
00053             a = a + AccelPointMass(extract_vector(Y,1,3),r_Uranus,GM_Uranus);
00054             a = a + AccelPointMass(extract_vector(Y,1,3),r_Neptune,GM_Neptune);
00055             a = a + AccelPointMass(extract_vector(Y,1,3),r_Pluto,GM_Pluto);}
00056 
00057     return union_vector(extract_vector(Y,4,6),a);
00058 
00059     }
```

5.75 AccelHarmonic.cpp File Reference

El archivo contiene las implementaciones de AccelHarmonic.h.

```
#include "../include/AccelHarmonic.h"
#include "../include/Legendre.h"
#include "../include/GLOBAL.h"
#include "../include/SAT_Const.h"
#include <cmath>
```

Include dependency graph for AccelHarmonic.cpp:



Functions

- Matrix & AccelHarmonic (Matrix r, Matrix E, int n_max, int m_max)

5.75.1 Detailed Description

El archivo contiene las implementaciones de AccelHarmonic.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AccelHarmonic.cpp.

5.75.2 Function Documentation

5.75.2.1 AccelHarmonic()

```
Matrix & AccelHarmonic (
            Matrix r,
            Matrix E,
            int n_max,
            int m_max)
```

Parameters

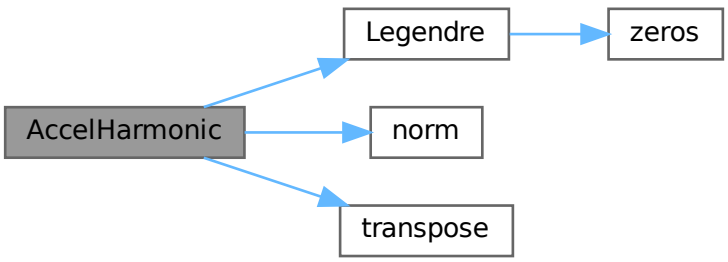
r	Satellite position vector in the inertial system
E	Transformation matrix to body-fixed system
n_max	Maximum degree
m_max	Maximum order (m_max<=n_max; m_max=0 for zonals, only)

Returns

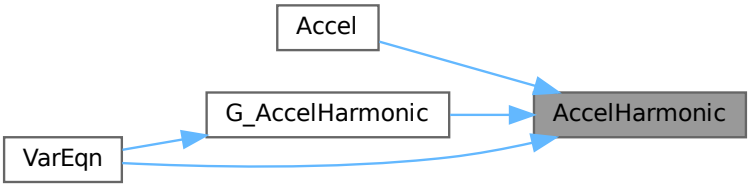
a Acceleration (a=d^2r/dt^2)

Definition at line 12 of file AccelHarmonic.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.76 AccelHarmonic.cpp

Go to the documentation of this file.

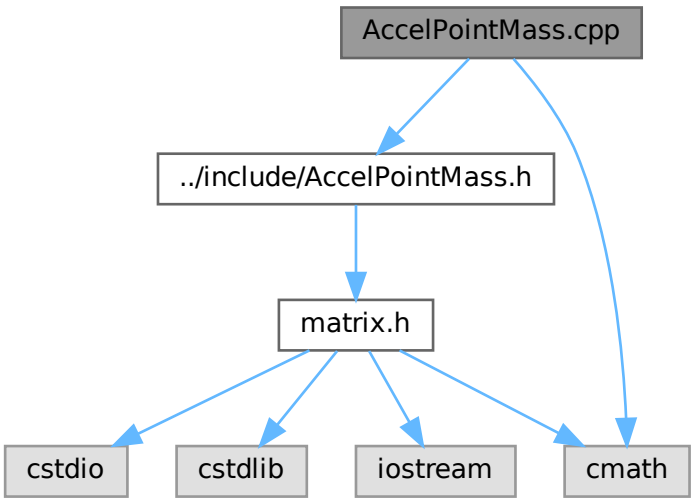
```
00001 #include "../include/AccelHarmonic.h"
00002 #include "../include/Legendre.h"
00003 #include "../include/GLOBAL.h"
00004 #include "../include/SAT_Const.h"
00005 #include <cmath>
00012 Matrix& AccelHarmonic(Matrix r,Matrix E,int n_max,int m_max){
00013         if(r.n_row<r.n_column){
00014             r=transpose(r);
00015         }
00016         double r_ref,gm,d,latgc,lon,b1,b2,b3,dUdr,dUdlatgc,dUdlon,q3,q2,q1,r2xy,ax,ay,az;
00017         Matrix a_bf(3);
00018     r_ref = 6378.1363e3;   // Earth's radius [m]; GGM03S
00019     gm    = 398600.4415e9; // [m^3/s^2]; GGM03S
00020
00021     // Body-fixed position
00022     Matrix r_bf = E * r;
00023
00024     // Auxiliary quantities
00025     d = norm(transpose(r_bf));                      // distance
00026
00027     latgc = asin(r_bf(3)/d);
00028     lon = atan2(r_bf(2),r_bf(1));
00029     auto[pnm, dpnm] = Legendre(n_max,m_max,latgc);
00030     dUdr = 0;
00031     dUdlatgc = 0;
00032     dUdlon = 0;
00033     q3 = 0; q2 = q3; q1 = q2;
00034     for (int n=0;n<=n_max;n++){
00035         b1 = (-gm/pow(d,2))*pow((r_ref/d),n)*(n+1);
00036         b2 =  (gm/d)*pow((r_ref/d),n);
00037         b3 =  (gm/d)*pow((r_ref/d),n);
00038
00039         for (int m=0;m<=m_max;m++){
00040             q1 = q1 + pnm(n+1,m+1)*(Cnm(n+1,m+1)*cos(m*lon)+Snm(n+1,m+1)*sin(m*lon));
00041             q2 = q2 + dpnm(n+1,m+1)*(Cnm(n+1,m+1)*cos(m*lon)+Snm(n+1,m+1)*sin(m*lon));
00042             q3 = q3 + m*pnm(n+1,m+1)*(Snm(n+1,m+1)*cos(m*lon)-Cnm(n+1,m+1)*sin(m*lon));
00043         }
00044         dUdr     = dUdr     + q1*b1;
00045         dUdlatgc = dUdlatgc + q2*b2;
00046         dUdlon   = dUdlon   + q3*b3;
00047         q3 = 0.0; q2 = q3; q1 = q2;
00048     }
00049     // Body-fixed acceleration
00050     r2xy = pow(r_bf(1),2)+pow(r_bf(2),2);
00051
00052     ax = (1.0/d*dUdr-r_bf(3)/(pow(d,2)*sqrt(r2xy))*dUdlatgc)*r_bf(1)-(1.0/r2xy*dUdlon)*r_bf(2);
00053     ay = (1.0/d*dUdr-r_bf(3)/(pow(d,2)*sqrt(r2xy))*dUdlatgc)*r_bf(2)+(1.0/r2xy*dUdlon)*r_bf(1);
00054     az =  1.0/d*dUdr*r_bf(3)+sqrt(r2xy)/pow(d,2)*dUdlatgc;
00055
00056     a_bf (1)=ax;
00057     a_bf (2)=ay ;
00058     a_bf (3)=az;
00059     a_bf=transpose(a_bf);
00060     // Inertial acceleration
00061     Matrix a = transpose(E)*a_bf;
00062     return transpose(a);
00063 }
```

5.77 AccelPointMass.cpp File Reference

El archivo contiene las implementaciones de AccelPointMass.h.

```
#include "../include/AccelPointMass.h"
#include <cmath>
```

Include dependency graph for AccelPointMass.cpp:



Functions

- Matrix & AccelPointMass (Matrix &r, Matrix &s, double GM)

5.77.1 Detailed Description

El archivo contiene las implementaciones de AccelPointMass.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AccelPointMass.cpp.

5.77.2 Function Documentation

5.77.2.1 AccelPointMass()

```
Matrix & AccelPointMass (
            Matrix & r,
            Matrix & s,
            double GM)
```

Parameters

r	Satellite position vector
s	Point mass position vector
GM	Gravitational coefficient of point mass

Returns

Acceleration (a=d^2r/dt^2)

Definition at line 10 of file AccelPointMass.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

5.78 AccelPointMass.cpp

Go to the documentation of this file.

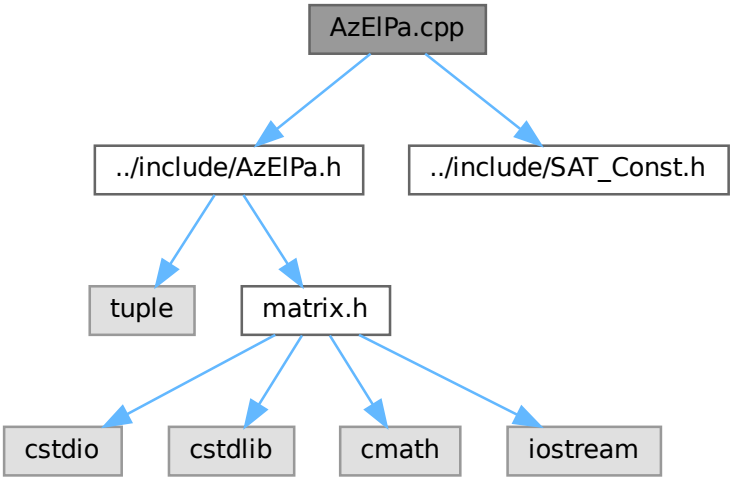
```
00001 #include "../include/AccelPointMass.h"
00002 #include <cmath>
00003
00010     Matrix&   AccelPointMass(Matrix& r,Matrix& s,double GM){
00011         Matrix d = r - s;
00012         Matrix &a= d/pow(norm(d),3) + s/(pow(norm(s),3));
00013         a=a * -GM;
00014         return a;
00015     }
```

5.79 AzElPa.cpp File Reference

El archivo contiene las implementaciones de AzElPa.h.

```
#include "../include/AzElPa.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for AzElPa.cpp:



Functions

- tuple< double, double, Matrix &, Matrix & > AzElPa (Matrix s)

5.79.1 Detailed Description

El archivo contiene las implementaciones de AzElPa.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file AzElPa.cpp.

5.79.2 Function Documentation

5.79.2.1 AzElPa()

```
tuple< double, double, Matrix &, Matrix & > AzElPa (
            Matrix s)
```

Computes azimuth, elevation and partials from local tangent coordinates s

Parameters

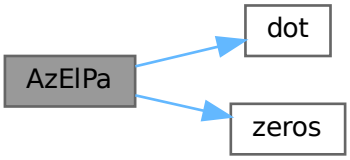
s	Matrix 1x3 Topocentric local tangent coordinates (East-North-Zenith frame)

Returns

tuple<A,E,dAds,dEds> Azimuth [rad],Elevation [rad],Partials of azimuth w.r.t. s,Partials of elevation w.r.t. s

Definition at line 9 of file AzElPa.cpp.

Here is the call graph for this function:



5.80 AzElPa.cpp

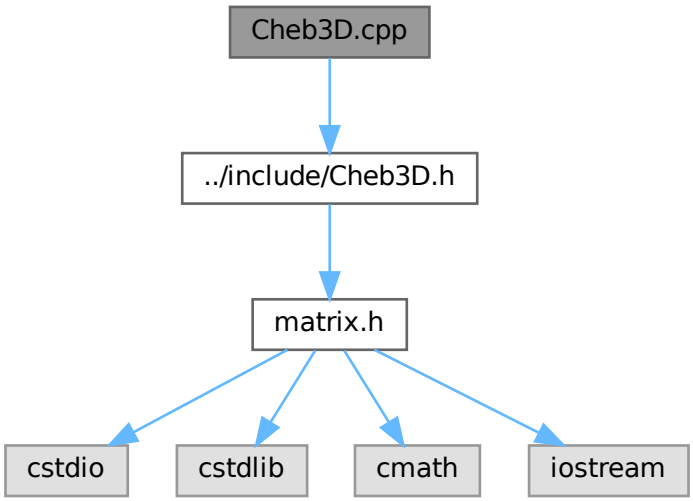
Go to the documentation of this file.

```
00001 #include "../include/AzElPa.h"
00002 #include "../include/SAT_Const.h"
00009     tuple<double,double,Matrix&,Matrix&> AzElPa(Matrix s) {
00010
00011         long double rho = sqrt(s(1)*s(1)+s(2)*s(2));
00012
00013         // Angles
00014         double Az = atan2(s(1),s(2));
00015
00016         if (Az<0.0) {
00017             Az = Az+pi2;
00018         }
00019
00020         double El = atan ( s(3) / rho );
00021
00022         // Partials
00023         Matrix &dAds = zeros(3);
00024         dAds(1)=s(2)/(rho*rho);
00025         dAds(2)=-s(1)/(rho*rho);
00026         dAds(3)=0.0 ;
00027         Matrix &dEds= zeros(3);
00028         dEds(1)=-s(1)*s(3)/rho;
00029         dEds(2)=-s(2)*s(3)/rho;
00030         dEds(3)=rho;
00031          dEds= dEds/ dot(s,s);
00032         return tie(Az, El, dAds, dEds);
00033
00034     }
```

5.81 Cheb3D.cpp File Reference

El archivo contiene las implementaciones de Cheb3D.h.

```
#include "../include/Cheb3D.h"
```
Include dependency graph for Cheb3D.cpp:



Functions

- Matrix & Cheb3D (double t, int N, double Ta, double Tb, Matrix &Cx, Matrix &Cy, Matrix &Cz)

5.81.1 Detailed Description

El archivo contiene las implementaciones de Cheb3D.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Cheb3D.cpp.

5.81.2 Function Documentation

5.81.2.1 Cheb3D()

```
Matrix & Cheb3D (
            double t,
            int N,
            double Ta,
            double Tb,
            Matrix & Cx,
            Matrix & Cy,
            Matrix & Cz)
```

Parameters

t	time
N	Number of coefficients
Ta	Begin interval
Tb	End interval
Cx	Coefficients of Chebyshev polyomial (x-coordinate)
Cy	Coefficients of Chebyshev polyomial (y-coordinate)
Cz	Coefficients of Chebyshev polyomial (z-coordinate)

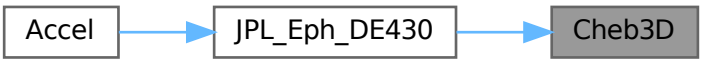
Returns

Chebyshev approximation of 3-dimensional vectors

Definition at line 9 of file Cheb3D.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



5.82 Cheb3D.cpp

Go to the documentation of this file.

```
00001 #include "../include/Cheb3D.h"
00002
00009     Matrix& Cheb3D( double t, int N, double Ta, double Tb, Matrix& Cx,Matrix& Cy,Matrix& Cz){
00010
00011         if ( (t<Ta) || (Tb<t) ){
00012             cerr«"ERROR: Time out of range in Cheb3D::Value\n";
00013             exit(EXIT_FAILURE);
00014             }
00015
00016         double tau = (2*t-Ta-Tb)/(Tb-Ta);
00017
00018         Matrix f1 = zeros(1,3);
```

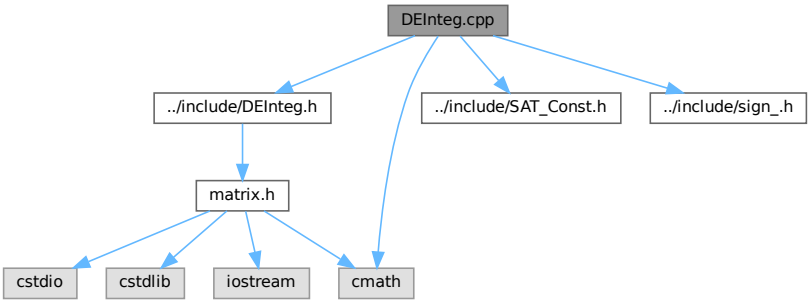
```
00019         Matrix f2 = zeros(1,3);
00020         Matrix old_f1= zeros(1,3);
00021         Matrix aux= zeros(1,3);
00022         for (int i=N;i>=2;i--){
00023             old_f1 = f1;
00024             aux(1)=Cx(i);
00025             aux(2)=Cy(i);
00026             aux(3)=Cz(i);
00027             f1 = (f1*(2*tau))-f2+aux;
00028             f2 = old_f1;
00029         }
00030         Matrix *ChebApp=&zeros(1,3);
00031         aux(1)=Cx(1);
00032         aux(2)=Cy(1);
00033         aux(3)=Cz(1);
00034         (*ChebApp) = f1*tau;
00035         (*ChebApp)=(*ChebApp)-f2;
00036         (*ChebApp)=(*ChebApp)+aux;
00037         return *ChebApp;
00038     }
```

5.83 DEInteg.cpp File Reference

El archivo contiene las implementaciones de DEInteg.h.

```
#include "../include/DEInteg.h"
#include "../include/SAT_Const.h"
#include "../include/sign_.h"
#include <cmath>
```

Include dependency graph for DEInteg.cpp:



Functions

- Matrix & DEInteg (Matrix &f(double t, Matrix z), double t, double tout, double relerr, double abserr, int n_eqn, Matrix &y)

5.83.1 Detailed Description

El archivo contiene las implementaciones de DEInteg.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file DEInteg.cpp.

5.83.2 Function Documentation

5.83.2.1 DEInteg()

```
Matrix & DEInteg (
            Matrix & fdouble t, Matrix z,
            double t,
            double tout,
            double relerr,
            double abserr,
            int n_eqn,
            Matrix & y)
```

Definition at line 11 of file DEInteg.cpp.

5.84 DEInteg.cpp

Go to the documentation of this file.

```
00001 #include "../include/DEInteg.h"
00002 #include "../include/SAT_Const.h"
00003 #include "../include/sign_.h"
00004 #include <cmath>
00011 Matrix& DEInteg(Matrix& f(double t,Matrix z),double t, double tout,double relerr,double abserr,int
      n_eqn,Matrix &y){
00012   if(y.n_row<y.n_column){
00013     y=transpose(y);
00014   }
00015
00016   Matrix yout=zeros(n_eqn,1),ypout=zeros(n_eqn,1),two(14),gstr(14);
00017   bool
      start=false,phase1=false,nornd=false,crash=false,success=false,PermitTOUT=false,OldPermit=false,stiff=false;
00018   long double
      delsgn=0.0,x=0.0,hi=0.0,ki=0.0,kold=0.0,temp1=0.0,term=0.0,psijm1=0.0,eta=0.0,sum=0.0,absh=0.0,hold=0.0,hnew=0.0,
00019
      k=0.0,round=0.0,gamma=0.0,i=0.0,p5eps=0.0,ifail=0.0,kp1=0.0,kp2=0.0,km1=0.0,km2=0.0,ns=0.0,nsp1=0.0,realns=0.0,im1=0.0,t
00020
      temp3=0.0,reali=0.0,temp4=0.0,nsm2=0.0,limit1=0.0,temp5=0.0,temp6=0.0,limit2=0.0,nsp2=0.0,ip1=0.0,tau=0.0,xold=0.0,erkm2
00021
      erkm1=0.0,erk,err=0.0,knew=0.0,rhi=0.0,h=0.0,erkp1=0.0,rhodouble=0.0,told=0.0,epsilon=0.0,del=0.0,absdel=0.0,tend=0.0,no
00022   releps=0.0,abseps=0.0,twou=0.0,fouru=0.0;
00023   double r=0.0;
00024   int l=0;
00025   twou  = 2*eps;
00026   fouru = 4*eps;
00027
00028
00029   struct DE_STATE_t {
00030     int DE_INIT = 1;
00031     int DE_DONE = 2;
00032     int DE_BADACC = 3;
00033     int DE_NUMSTEPS = 4;
00034     int DE_STIFF = 5;
00035     int DE_INVPARAM = 6;
00036   };
00037
00038   DE_STATE_t DE_STATE;
00039
00040   int State_ = DE_STATE.DE_INIT;
00041 PermitTOUT = true,OldPermit;
00042 told = 0;
00043
00044
00045 double arrtwo[]  = {1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0,256.0, 512.0, 1024.0, 2048.0, 4096.0,
      8192.0};
00046 double arrgstr[] = {1.0, 0.5, 0.0833, 0.0417, 0.0264, 0.0188, 0.0143, 0.0114, 0.00936, 0.00789,
      0.00679,0.00592, 0.00524, 0.00468};
00047 for(int wxy=1;wxy<=14;wxy++){
00048   two(wxy)=arrtwo[wxy-1];
00049   gstr(wxy)=arrgstr[wxy-1];
00050 }
00051
00052 Matrix yy    = zeros(n_eqn,1);
00053 Matrix wt    = zeros(n_eqn,1);
```

```
00054 Matrix p     = zeros(n_eqn,1);
00055 Matrix yp    = zeros(n_eqn,1);
00056 Matrix phi   = zeros(n_eqn,17);
00057 Matrix g     = zeros(14,1);
00058 Matrix sig   = zeros(14,1);
00059 Matrix rho   = zeros(14,1);
00060 Matrix w     = zeros(13,1);
00061 Matrix alpha = zeros(13,1);
00062 Matrix beta  = zeros(13,1);
00063 Matrix v     = zeros(13,1);
00064 Matrix psi_  = zeros(13,1);
00065
00066 if (t==tout) {
00067   return y;}
00068
00069   epsilon = fmax(relerr,abserr);
00070   if ( ( relerr <  0.0  ) || ( abserr <  0.0 ) ||  ( epsilon    <= 0.0  ) ||
00071     ( State_  >  DE_STATE.DE_INVPARAM ) ||  ( (State_ != DE_STATE.DE_INIT) &&  (t != told)           )
      )
00072   {
00073      State_ = DE_STATE.DE_INVPARAM;
00074        return y;
00075   }
00076
00077   del    = tout - t;
00078   absdel = fabs(del);
00079
00080   tend   = t + 100.0*del;
00081   if (!PermitTOUT){
00082     tend = tout;
00083   }
00084   nostep = 0;
00085   kle4   = 0;
00086   stiff  = false;
00087   releps = relerr/epsilon;
00088   abseps = abserr/epsilon;
00089
00090   if  ( (State_==DE_STATE.DE_INIT) || (!OldPermit) || (delsgn*del<=0.0) ){
00091
00092
00093     start  = true;
00094     x      = t;
00095     yy     = y;
00096     delsgn = sign_(1.0, del);
00097     h = sign_( fmax(fouru*fabs(x), fabs(tout-x)), tout-x );}
00098 while(true){
00099   if (fabs(x-t) >= absdel){
00100     yout  = zeros(n_eqn,1);
00101     ypout = zeros(n_eqn,1);
00102     g(2)   = 1.0;
00103     rho(2) = 1.0;
00104     hi = tout - x;
00105     ki = kold + 1;
00106
00107     for (int i=1;i<=k;i++){
00108       temp1 = i;
00109       w(i+1) = 1.0/temp1;}
00110
00111       term = 0.0;
00112       for (int j=2;j<=ki;j++){
00113         psijm1 = psi_(j);
00114         gamma = (hi + term)/psijm1;
00115         eta = hi/psijm1;
00116         for (int i=1;i<=ki+1-j;i++){
00117           w(i+1) = gamma*w(i+1) - eta*w(i+2);}
00118           g(j+1) = w(2);
00119           rho(j+1) = gamma*rho(j);
00120           term = psijm1;}
00121
00122         if(yout.n_row>yout.n_column){
00123         yout=transpose(yout);}
00124         if(ypout.n_row>ypout.n_column){
00125         ypout=transpose(ypout);}
00126         if(y.n_row>y.n_column){
00127         y=transpose(y);
00128          }
00129           for (int j=1;j<=ki;j++){
00130             i = ki+1-j;
00131             yout  = yout  + extract_column(phi,i+1)*g(i+1);
00132             ypout = ypout + extract_column(phi,i+1)*rho(i+1);
00133           }
00134             yout = y +yout*hi;
00135             y    = yout;
00136       State_    = DE_STATE.DE_DONE;
00137       t         = tout;
00138       told      = t;
00139       OldPermit = PermitTOUT;
```

```
00140       return y;
00141     }
00142
00143     if ( !PermitTOUT && ( fabs(tout-x) < fouru*fabs(x) ) ){
00144       h = tout - x;
00145       yp = f(x,yy);
00146       y = yy + yp*h;
00147       State_    = DE_STATE.DE_DONE;
00148       t         = tout;
00149       told      = t;
00150       OldPermit = PermitTOUT;
00151       return y;
00152     }
00153
00154     h  = sign_(min(fabs(h), fabs(tend-x)), h);
00155     for (l=1;l<=n_eqn;l++){
00156       wt(l) = releps*fabs(yy(l)) + abseps;
00157     }
00158
00159     if (fabs(h) < fouru*fabs(x)){
00160       h = sign_(fouru*fabs(x),h);
00161       crash = true;
00162       return y;
00163   }
00164
00165   p5eps  = 0.5*epsilon;
00166   crash  = false;
00167   g(2)   = 1.0;
00168   g(3)   = 0.5;
00169   sig(2) = 1.0;
00170
00171   ifail = 0;
00172
00173   round = 0.0;
00174   for (l=1;l<=n_eqn;l++){
00175     round = round + (y(l)*y(l))/(wt(l)*wt(l));
00176   }
00177   round = twou*sqrt(round);
00178   if (p5eps<round){
00179     epsilon = 2.0*round*(1.0+fouru);
00180     crash = true;
00181     return y;
00182   }
00183   if (start){
00184
00185     yp = transpose(f(x,y));
00186     sum = 0.0;
00187     for (l=1;l<=n_eqn;l++){
00188       phi(l,2) = yp(l);
00189       phi(l,3) = 0.0;
00190       sum = sum + (yp(l)*yp(l))/(wt(l)*wt(l));
00191     }
00192     sum  = sqrt(sum);
00193     absh = fabs(h);
00194     if (epsilon<16.0*sum*h*h){
00195       absh=0.25*sqrt(epsilon/sum);
00196     }
00197     h    = sign_(fmax(absh, fouru*fabs(x)), h);
00198     hold = 0.0;
00199     hnew = 0.0;
00200     k    = 1;
00201     kold = 0;
00202     start  = false;
00203     phase1 = true;
00204     nornd  = true;
00205     if (p5eps<=100.0*round){
00206       nornd = false;
00207       for (l=1;l<=n_eqn;l++){
00208         phi(l,16)=0.0;
00209       }
00210     }
00211   }
00212
00213   while(true){
00214
00215     kp1 = k+1;
00216     kp2 = k+2;
00217     km1 = k-1;
00218     km2 = k-2;
00219
00220     if (h !=hold){
00221       ns=0;
00222     }
00223     if (ns<=kold){
00224       ns=ns+1;
00225     }
00226     nsp1 = ns+1;
```

```
00227
00228     if (k>=ns){
00229       beta(ns+1) = 1.0;
00230       realns = ns;
00231       alpha(ns+1) = 1.0/realns;
00232       temp1 = h*realns;
00233       sig(nsp1+1) = 1.0;
00234       if (k>=nsp1){
00235         for (int i=nsp1;i<=k;i++){
00236           im1   = i-1;
00237           temp2 = psi_(im1+1);
00238           psi_(im1+1) = temp1;
00239           beta(i+1)  = beta(im1+1)*psi_(im1+1)/temp2;
00240           temp1    = temp2 + h;
00241           alpha(i+1) = h/temp1;
00242           reali = i;
00243           sig(i+2) = reali*alpha(i+1)*sig(i+1);
00244         }
00245       }
00246       psi_(k+1) = temp1;
00247
00248
00249       if (ns>1){
00250
00251         if (k>kold){
00252           temp4 = k*kp1;
00253           v(k+1) = 1.0/temp4;
00254           nsm2 = ns-2;
00255           for (int j=1;j<=nsm2;j++){
00256             i = k-j;
00257             v(i+1) = v(i+1) - alpha(j+2)*v(i+2);
00258           }
00259         }
00260
00261
00262         limit1 = kp1 - ns;
00263         temp5  = alpha(ns+1);
00264         for (int iq=1;iq<=limit1;iq++){
00265           v(iq+1) = v(iq+1) - temp5*v(iq+2);
00266           w(iq+1) = v(iq+1);
00267         }
00268         g(nsp1+1) = w(2);}
00269         else{
00270           for (int iq=1;iq<=k;iq++){
00271             temp3 = iq*(iq+1);
00272             v(iq+1) = 1.0/temp3;
00273             w(iq+1) = v(iq+1);
00274           }
00275         }
00276
00277
00278         nsp2 = ns + 2;
00279         if (kp1>=nsp2){
00280           for (int i=nsp2;i<=kp1;i++){
00281             limit2 = kp2 - i;
00282             temp6  = alpha(i);
00283             for (int iq=1;iq<=limit2;iq++){
00284               w(iq+1) = w(iq+1) - temp6*w(iq+2);
00285             }
00286             g(i+1) = w(2);
00287           }
00288         }
00289   }
00290
00291
00292   if (k>=nsp1){
00293     for (int i=nsp1;i<=k;i++){
00294       temp1 = beta(i+1);
00295       for (l=1;l<=n_eqn;l++){
00296         phi(l,i+1) = temp1 * phi(l,i+1);
00297       }
00298     }
00299   }
00300
00301   for (l=1;l<=n_eqn;l++){
00302     phi(l,kp2+1) = phi(l,kp1+1);
00303     phi(l,kp1+1) = 0.0;
00304     p(l)         = 0.0;
00305   }
00306   for (int j=1;j<=k;j++){
00307     i     = kp1 - j;
00308     ip1   = i+1;
00309     temp2 = g(i+1);
00310     for (l=1;l<=n_eqn;l++){
00311       p(l)     = p(l) + temp2*phi(l,i+1);
00312       phi(l,i+1) = phi(l,i+1) + phi(l,ip1+1);
00313     }
```

```
00314   }
00315   if (nornd){
00316    p = y + p*h;}
00317    else{
00318     for (l=1;l<=n_eqn;l++){
00319       tau = h*p(l) - phi(l,16);
00320       p(l) = y(l) + tau;
00321       phi(l,17) = (p(l) - y(l)) - tau;
00322     }
00323   }
00324   xold = x;
00325   x = x + h;
00326   absh = fabs(h);
00327   yp = f(x,p);
00328
00329   erkm2 = 0.0;
00330   erkm1 = 0.0;
00331   erk = 0.0;
00332   for (l=1;l<=n_eqn;l++){
00333     temp3 = 1.0/wt(l);
00334     temp4 = yp(l) - phi(l,1+1);
00335     if (km2> 0){
00336       erkm2 = erkm2 + ((phi(l,km1+1)+temp4)*temp3)*((phi(l,km1+1)+temp4)*temp3);
00337     }
00338     if (km2>=0){
00339       erkm1 = erkm1 + ((phi(l,k+1)+temp4)*temp3)*((phi(l,k+1)+temp4)*temp3);
00340     }
00341     erk = erk + (temp4*temp3)*(temp4*temp3);
00342   }
00343
00344   if (km2> 0){
00345     erkm2 = absh*sig(km1+1)*gstr(km2+1)*sqrt(erkm2);
00346   }
00347   if (km2>=0){
00348     erkm1 = absh*sig(k+1)*gstr(km1+1)*sqrt(erkm1);
00349   }
00350   temp5 = absh*sqrt(erk);
00351   err = temp5*(g(k+1)-g(kp1+1));
00352   erk = temp5*sig(kp1+1)*gstr(k+1);
00353   knew = k;
00354
00355
00356   if (km2 >0){
00357     if (fmax(erkm1,erkm2)<=erk){
00358       knew=km1;
00359     }
00360   }
00361   if (km2==0){
00362     if (erkm1<=0.5*erk){
00363       knew=km1;
00364     }
00365   }
00366
00367   success = (err<=epsilon);
00368
00369   if (!success){
00370     phase1 = false;
00371     x = xold;
00372     for (int i=1;i<=k;i++){
00373       temp1 = 1.0/beta(i+1);
00374       ip1 = i+1;
00375       for (l=1;l<=n_eqn;l++){
00376         phi(l,i+1)=temp1*(phi(l,i+1)-phi(l,ip1+1));
00377       }
00378     }
00379
00380     if (k>=2){
00381       for (int i=2;i<=k;i++){
00382         psi_(i) = h-psi_(i+1);
00383       }
00384     }
00385
00386
00387
00388     ifail = ifail+1;
00389     temp2 = 0.5;
00390     if (ifail>3) {
00391       if (p5eps < 0.25*erk){
00392         temp2 = sqrt(p5eps/erk);
00393       }
00394     }
00395     if (ifail>=3){
00396       knew = 1;
00397     }
00398     h = temp2*h;
00399     k = knew;
00400     if (fabs(h)<fouru*fabs(x)){
```

```
00401       crash = true;
00402       h = sign_(fouru*fabs(x), h);
00403       epsilon = epsilon*2.0;
00404         return y;
00405       }
00406   }
00407
00408   if (success){
00409     break;
00410   }
00411
00412 }
00413
00414 kold = k;
00415 hold = h;
00416
00417 temp1 = h*g(kp1+1);
00418 if (nornd){
00419   for (l=1;l<=n_eqn;l++){
00420     y(l) = p(l) + temp1*(yp(l) - phi(l,2));
00421   }
00422 }
00423 else{
00424   for (l=1;l<=n_eqn;l++){
00425     rhodouble = temp1*(yp(l) - phi(l,2)) - phi(l,17);
00426     y(l) = rhi + p(l);
00427     phi(l,16) = (y(l) - p(l)) - rhodouble;
00428   }
00429 }
00430 yp = f(x,y);
00431
00432
00433 for (l=1;l<=n_eqn;l++){
00434   phi(l,kp1+1) = yp(l) - phi(l,2);
00435   phi(l,kp2+1) = phi(l,kp1+1) - phi(l,kp2+1);
00436 }
00437 for (int i=1;i<=k;i++){
00438   for (l=1;l<=n_eqn;l++){
00439     phi(l,i+1) = phi(l,i+1) + phi(l,kp1+1);
00440   }
00441 }
00442
00443
00444 erkp1 = 0.0;
00445 if ( (knew==km1) || (k==12) ){
00446   phase1 = false;
00447 }
00448
00449 if (phase1){
00450   k = kp1;
00451   erk = erkp1;}
00452   else{
00453     if (knew==km1){
00454
00455       k = km1;
00456       erk = erkm1;}
00457       else{
00458         if (kp1<=ns){
00459           for (l=1;l<=n_eqn;l++){
00460             erkp1 = erkp1 + (phi(l,kp2+1)/wt(l))*(phi(l,kp2+1)/wt(l));
00461           }
00462           erkp1 = absh*gstr(kp1+1)*sqrt(erkp1);
00463
00464
00465           if (k>1){
00466             if ( erkm1<=min(erk,erkp1)){
00467
00468               k=km1; erk=erkm1;}
00469               else{
00470                 if ( (erkp1<erk) && (k!=12) ){
00471
00472                   k=kp1;
00473                   erk=erkp1;
00474                 }
00475               }
00476             }
00477             else if (erkp1<0.5*erk){
00478
00479               k = kp1;
00480               erk = erkp1;
00481             }
00482         }
00483     }
00484 }
00485
00486
00487 if ( phase1 || (p5eps>=erk*two(k+2)) ){
```

```
00488   hnew = 2.0*h;}
00489   else{
00490     if (p5eps<erk){
00491       temp2 = k+1;
00492       r = pow(p5eps/erk,(1.0/temp2));
00493       hnew = absh*fmax(0.5, min(0.9,r));
00494       hnew = sign_(fmax(hnew, fouru*fabs(x)), h);}
00495       else{
00496         hnew = h;
00497       }
00498     }
00499     h = hnew;
00500
00501     if (crash){
00502       State_    = DE_STATE.DE_BADACC;
00503       relerr    = epsilon*releps;
00504       abserr    = epsilon*abseps;
00505       y         = yy;
00506       t         = x;
00507       told      = t;
00508       OldPermit = true;
00509       return y;
00510     }
00511
00512   nostep = nostep+1;
00513
00514
00515
00516   kle4 = kle4+1;
00517   if (kold>  4){
00518     kle4 = 0;
00519   }
00520   if (kle4>=50){
00521     stiff = true;
00522   }
00523 }
00524 /*
00525 cout « "delsgn: " « delsgn « endl;
00526 cout « "x: " « x « endl;
00527 cout « "hi: " « hi « endl;
00528 cout « "ki: " « ki « endl;
00529 cout « "kold: " « kold « endl;
00530 cout « "temp1: " « temp1 « endl;
00531 cout « "term: " « term « endl;
00532 cout « "psijm1: " « psijm1 « endl;
00533 cout « "eta: " « eta « endl;
00534 cout « "sum: " « sum « endl;
00535 cout « "absh: " « absh « endl;
00536 cout « "hold: " « hold « endl;
00537 cout « "hnew: " « hnew « endl;
00538 cout « "k: " « k « endl;
00539 cout « "round: " « round « endl;
00540 cout « "gamma: " « gamma « endl;
00541 cout « "i: " « i « endl;
00542 cout « "p5eps: " « p5eps « endl;
00543 cout « "ifail: " « ifail « endl;
00544 cout « "kp1: " « kp1 « endl;
00545 cout « "kp2: " « kp2 « endl;
00546 cout « "km1: " « km1 « endl;
00547 cout « "km2: " « km2 « endl;
00548 cout « "ns: " « ns « endl;
00549 cout « "nsp1: " « nsp1 « endl;
00550 cout « "realns: " « realns « endl;
00551 cout « "im1: " « im1 « endl;
00552 cout « "temp2: " « temp2 « endl;
00553 cout « "temp3: " « temp3 « endl;
00554 cout « "reali: " « reali « endl;
00555 cout « "temp4: " « temp4 « endl;
00556 cout « "nsm2: " « nsm2 « endl;
00557 cout « "limit1: " « limit1 « endl;
00558 cout « "temp5: " « temp5 « endl;
00559 cout « "temp6: " « temp6 « endl;
00560 cout « "limit2: " « limit2 « endl;
00561 cout « "nsp2: " « nsp2 « endl;
00562 cout « "ip1: " « ip1 « endl;
00563 cout « "tau: " « tau « endl;
00564 cout « "xold: " « xold « endl;
00565 cout « "erkm2: " « erkm2 « endl;
00566 cout « "erkm1: " « erkm1 « endl;
00567 cout « "erk: " « erk « endl;
00568 cout « "err: " « err « endl;
00569 cout « "knew: " « knew « endl;
00570 cout « "rhi: " « rhi « endl;
00571 cout « "h: " « h « endl;
00572 cout « "r: " « r « endl;
00573 cout « "erkp1: " « erkp1 « endl;
00574 cout « "rhodouble: " « rhodouble « endl;
```

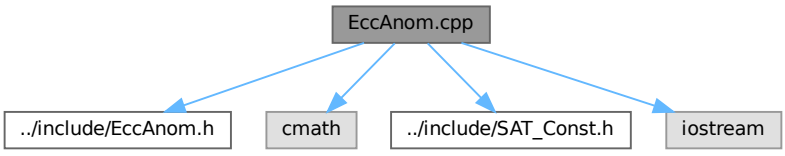
```
00575 cout « "told: " « told « endl;
00576 cout « "epsilon: " « epsilon « endl;
00577 cout « "del: " « del « endl;
00578 cout « "absdel: " « absdel « endl;
00579 cout « "tend: " « tend « endl;
00580 cout « "nostep: " « nostep « endl;
00581 cout « "kle4: " « kle4 « endl;
00582 cout « "releps: " « releps « endl;
00583 cout « "abseps: " « abseps « endl;
00584 cout « "twou: " « twou « endl;
00585 cout « "fouru: " « fouru « endl;
00586
00587 cout « "l: " « l « endl;
00588 cout « "y: \n" « y « endl;*/
00589
00590   cout«69«endl;
00591 return y;
00592 }
```

5.85 EccAnom.cpp File Reference

El archivo contiene las implementaciones de EccAnom.h.

```
#include "../include/EccAnom.h"
#include <cmath>
#include "../include/SAT_Const.h"
#include <iostream>
```

Include dependency graph for EccAnom.cpp:



Functions

- double EccAnom (double M, double e)

5.85.1 Detailed Description

El archivo contiene las implementaciones de EccAnom.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file EccAnom.cpp.

5.85.2 Function Documentation

5.85.2.1 EccAnom()

```
double EccAnom (
            double M,
            double e)
```

Computes the eccentric anomaly for elliptic orbits

Parameters

M	Mean anomaly in [rad]
e	Eccentricity of the orbit [0,1]

Returns

double resultado

Definition at line 12 of file EccAnom.cpp.

5.86 EccAnom.cpp

Go to the documentation of this file.

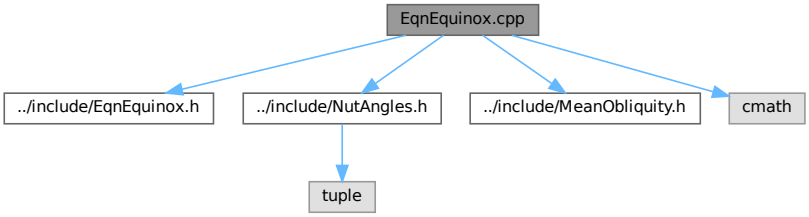
```
00001 #include "../include/EccAnom.h"
00002 #include <cmath>
00003 #include "../include/SAT_Const.h"
00004 #include <iostream>
00005 using namespace std;
00012     double EccAnom (double M,double e){
00013
00014     double maxit = 15;
00015     int i = 1;
00016
00017     M = fmod(M, 2.0*pi);
00018     double E;
00019
00020     if (e<0.8){
00021         E = M; }
00022     else{
00023         E = pi;}
00024
00025     double f = E - e*sin(E) - M;
00026     E = E - f / ( 1.0 - e*cos(E) );
00027
00028     while (abs(f) > 1e2*eps)   {
00029         f = E - e*sin(E) - M;
00030         E = E - f / ( 1.0 - e*cos(E) );
00031         i = i+1;
00032         if (i==maxit){
00033             cerr« "convergence problems in EccAnom";
00034         }
00035     }
00036     return E;
00037     }
```

5.87 EqnEquinox.cpp File Reference

El archivo contiene las implementaciones de EqnEquinox.h.

```
#include "../include/EqnEquinox.h"
#include "../include/NutAngles.h"
#include "../include/MeanObliquity.h"
#include <cmath>
```

Include dependency graph for EqnEquinox.cpp:



Functions

- double EqnEquinox (double Mjd_TT)

5.87.1 Detailed Description

El archivo contiene las implementaciones de EqnEquinox.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file EqnEquinox.cpp.

5.87.2 Function Documentation

5.87.2.1 EqnEquinox()

```
double EqnEquinox (
            double Mjd_TT)
```

Computation of the equation of the equinoxes

Parameters

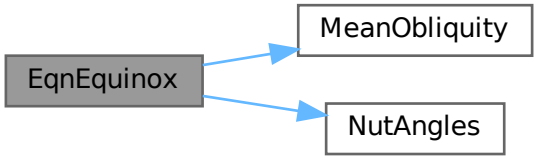
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

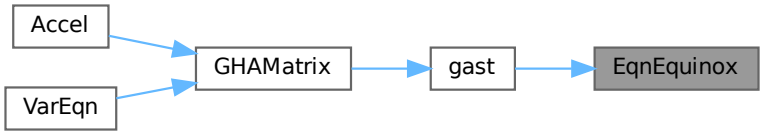
double Equation of the equinoxes

Definition at line 13 of file EqnEquinox.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.88 EqnEquinox.cpp

Go to the documentation of this file.

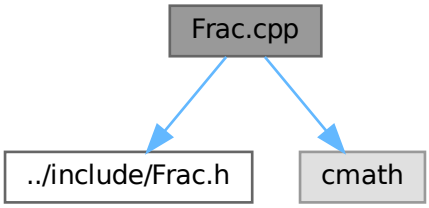
```
00001 #include "../include/EqnEquinox.h"
00002 #include "../include/NutAngles.h"
00003 #include "../include/MeanObliquity.h"
00004 #include <cmath>
00005
00006
00013     double EqnEquinox (double Mjd_TT){
00014
00015     auto [dpsi, deps] = NutAngles (Mjd_TT);
00016
00017     return dpsi * cos ( MeanObliquity(Mjd_TT) );
00018 }
00019
```

5.89 Frac.cpp File Reference

El archivo contiene las implementaciones de Frac.h.

```
#include "../include/Frac.h"
#include <cmath>
```

Include dependency graph for Frac.cpp:



Functions

- double Frac (double x)

5.89.1 Detailed Description

El archivo contiene las implementaciones de Frac.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Frac.cpp.

5.89.2 Function Documentation

5.89.2.1 Frac()

```
double Frac (
            double x)
```

Parameters

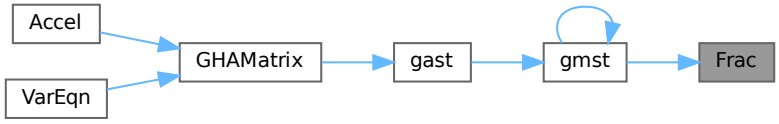
x	double

Returns

double parte fraccion de x

Definition at line 10 of file Frac.cpp.

Here is the caller graph for this function:



5.90 Frac.cpp

Go to the documentation of this file.

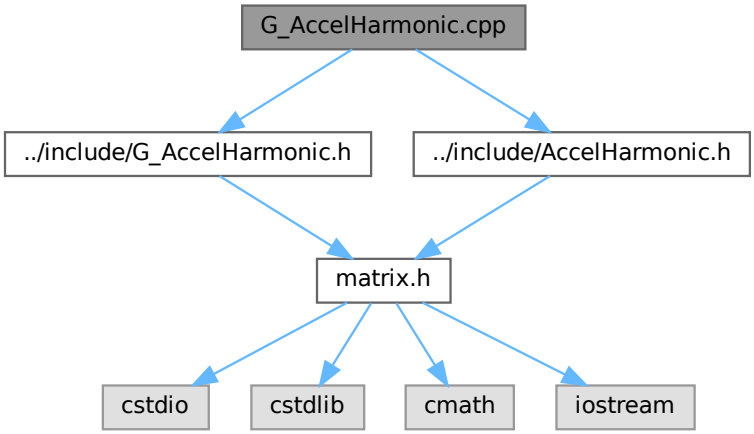
```
00001 #include "../include/Frac.h"
00002 #include <cmath>
00003
00010     double Frac(double x){
00011
00012     return x-floor(x);
00013     }
```

5.91 G_AccelHarmonic.cpp File Reference

El archivo contiene las implementaciones de G_AccelHarmonic.h.

```
#include "../include/G_AccelHarmonic.h"
#include "../include/AccelHarmonic.h"
```

Include dependency graph for G_AccelHarmonic.cpp:



Functions

- Matrix & G_AccelHarmonic (Matrix r, Matrix U, int n_max, int m_max)

5.91.1 Detailed Description

El archivo contiene las implementaciones de G_AccelHarmonic.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file G_AccelHarmonic.cpp.

5.91.2 Function Documentation

5.91.2.1 G_AccelHarmonic()

```
Matrix & G_AccelHarmonic (
            Matrix r,
            Matrix U,
            int n_max,
            int m_max)
```

Parameters

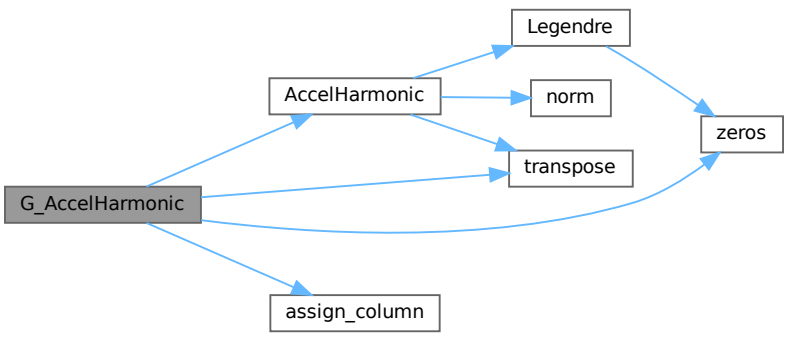
r	Satellite position vector in the true-of-date system
U	Transformation matrix to body-fixed syste
n	Gravity model degree
m	Gravity model order

Returns

G Gradient (G=da/dr) in the true-of-date system

Definition at line 10 of file G_AccelHarmonic.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.92 G_AccelHarmonic.cpp

Go to the documentation of this file.

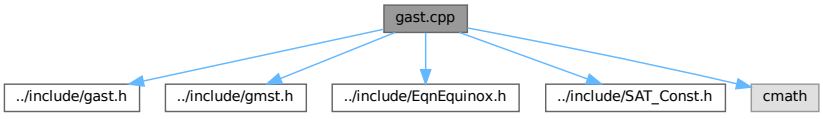
```
00001 #include "../include/G_AccelHarmonic.h"
00002 #include "../include/AccelHarmonic.h"
00003
00010     Matrix& G_AccelHarmonic( Matrix r,Matrix U, int n_max, int m_max ){
00011
00012             if(r.n_row<r.n_column){
00013                 r=transpose(r);
00014             }
00015         Matrix da;
00016     double d = 1.0;
00017
00018     Matrix &G = zeros(3,3);
00019     Matrix dr = zeros(3,1);
00020
00021     for (int i=1;i<=3;i++){
00022             dr = zeros(3,1);
00023             dr(i) = d/2;
00024
00025             da = AccelHarmonic ( r+dr,U, n_max, m_max ) -
00026             AccelHarmonic ( r-dr,U, n_max, m_max );
00027             G=assign_column(G,da/d,i) ;
00028                }
00029     return G;
00030     }
```

5.93 gast.cpp File Reference

El archivo contiene las implementaciones de gast.h.

```
#include "../include/gast.h"
#include "../include/gmst.h"
#include "../include/EqnEquinox.h"
#include "../include/SAT_Const.h"
#include <cmath>
```

Include dependency graph for gast.cpp:



Functions

- double gast (double Mjd_UT1)

5.93.1 Detailed Description

El archivo contiene las implementaciones de gast.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file gast.cpp.

5.93.2 Function Documentation

5.93.2.1 gast()

```
double gast (
            double Mjd_UT1)
```

Greenwich Apparent Sidereal Time

Parameters

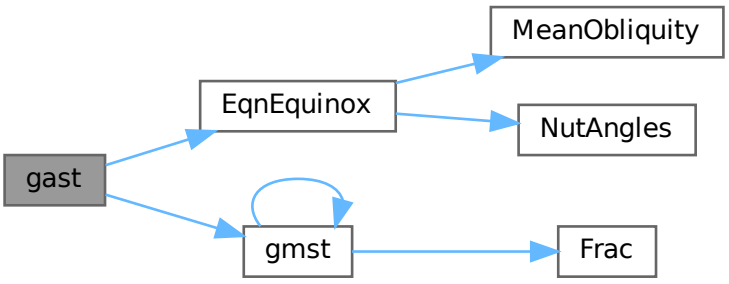
Mjd_UT1	Modified Julian Date UT1

Returns

gstime GAST in [rad]

Definition at line 14 of file gast.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.94 gast.cpp

Go to the documentation of this file.

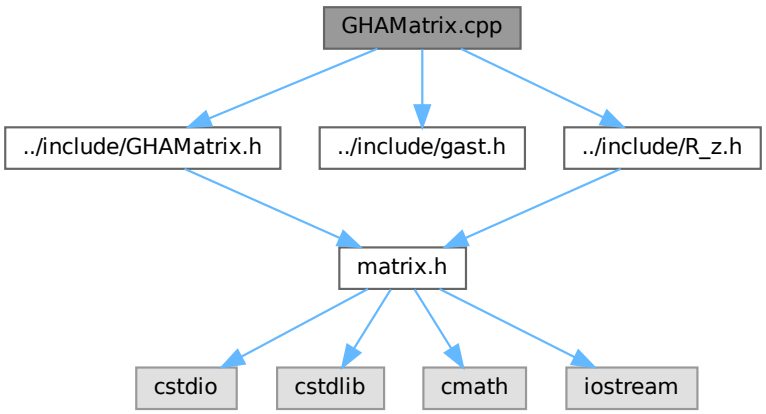
```
00001 #include "../include/gast.h"
00002 #include "../include/gmst.h"
00003 #include "../include/EqnEquinox.h"
00004 #include "../include/SAT_Const.h"
00005 #include <cmath>
00006
00007
00014     double gast(double Mjd_UT1){
00015         return fmod(gmst(Mjd_UT1) + EqnEquinox(Mjd_UT1), 2*pi );
00016     }
```

5.95 GHAMatrix.cpp File Reference

El archivo contiene las implementaciones de GHAMatrix.h.

```
#include "../include/GHAMatrix.h"
#include "../include/gast.h"
#include "../include/R_z.h"
```

Include dependency graph for GHAMatrix.cpp:



Functions

- Matrix & GHAMatrix (double Mjd_UT1)

5.95.1 Detailed Description

El archivo contiene las implementaciones de GHAMatrix.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file GHAMatrix.cpp.

5.95.2 Function Documentation

5.95.2.1 GHAMatrix()

```
Matrix & GHAMatrix (
            double Mjd_UT1)
```

Transformation from true equator and equinox to Earth equator and Greenwich meridian system

Parameters

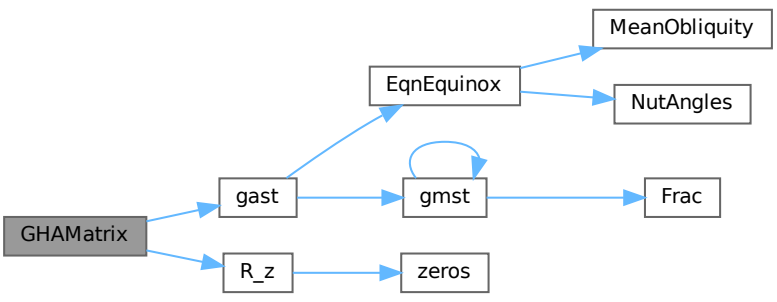
Mjd_UT1	Modified Julian Date UT1

Returns

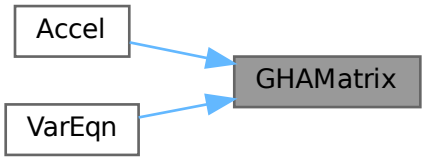
GHAmat Greenwich Hour Angle matrix

Definition at line 11 of file GHAMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.96 GHAMatrix.cpp

Go to the documentation of this file.

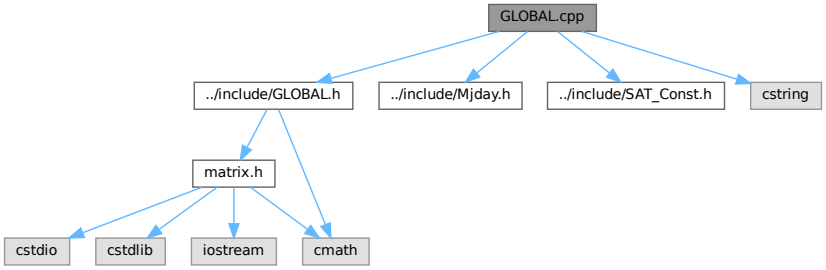
```
00001 #include "../include/GHAMatrix.h"
00002 #include "../include/gast.h"
00003 #include "../include/R_z.h"
00004
00011     Matrix& GHAMatrix (double Mjd_UT1){
00012         return  R_z( gast(Mjd_UT1) );
00013     }
```

5.97 GLOBAL.cpp File Reference

El archivo contiene las implementaciones de GLOBAL.h.

```
#include "../include/GLOBAL.h"
#include "../include/Mjday.h"
#include "../include/SAT_Const.h"
#include <cstring>
```

Include dependency graph for GLOBAL.cpp:



Functions

- void AuxParamLoad ()
- void eop19620101 (int c)
- void GGM03S (int n)
- void DE430Coeff (int row, int column)
- void GEOS3 (int nobs)

Variables

- Param AuxParam
- Matrix eopdata
- Matrix Cnm
- Matrix Snm
- Matrix PC
- Matrix obs

5.97.1 Detailed Description

El archivo contiene las implementaciones de GLOBAL.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file GLOBAL.cpp.

5.97.2 Function Documentation

5.97.2.1 AuxParamLoad()

```
void AuxParamLoad ()
```

Carga AuxParam

Definition at line 17 of file GLOBAL.cpp.

5.97.2.2 DE430Coeff()

```
void DE430Coeff (
            int row = 2285,
            int column = 1020)
```

Lee el archivo DE430Coeff.txt y recoge cada fila y lo asigna a PC

Parameters

row	número de filas a recoger
column	número de columnas a recoger

Definition at line 68 of file GLOBAL.cpp.

Here is the call graph for this function:

5.97.2.3 eop19620101()

```
void eop19620101 (
            int c = 21413)
```

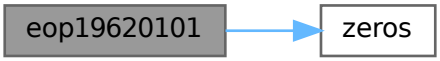
Lee el archivo eop19620101.txt y recoge cada fila y lo asigna a eopdata

Parameters

c	número de filas a recoger

Definition at line 26 of file GLOBAL.cpp.

Here is the call graph for this function:



5.97.2.4 GEOS3()

```
void GEOS3 (
            int nobs = 46)
```

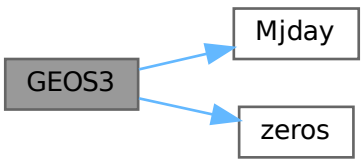
Lee el archivo GEOS3.txt y recoge cada fila y lo asigna a obs

Parameters

nobs	número de filas a recoger

Definition at line 87 of file GLOBAL.cpp.

Here is the call graph for this function:



5.97.2.5 GGM03S()

```
void GGM03S (
             int n = 181)
```

Lee el archivo GGM03S.txt y recoge cada fila y lo asigna a Cnm y Snm

Parameters

c	dimension de la matriz

Definition at line 47 of file GLOBAL.cpp.

Here is the call graph for this function:

5.97.3 Variable Documentation

5.97.3.1 AuxParam

```
Param AuxParam
```

Definition at line 11 of file GLOBAL.cpp.

5.97.3.2 Cnm

```
Matrix Cnm
```

Definition at line 13 of file GLOBAL.cpp.

5.97.3.3 eopdata

```
Matrix eopdata
```

Definition at line 12 of file GLOBAL.cpp.

5.97.3.4 obs

```
Matrix obs
```

Definition at line 16 of file GLOBAL.cpp.

5.97.3.5 PC

```
Matrix PC
```

Definition at line 15 of file GLOBAL.cpp.

5.97.3.6 Snm

```
Matrix Snm
```

Definition at line 14 of file GLOBAL.cpp.

5.98 GLOBAL.cpp

Go to the documentation of this file.

```
00001 #include "../include/GLOBAL.h"
00002 #include "../include/Mjday.h"
00003 #include "../include/SAT_Const.h"
00004 #include <cstring>
00011 Param AuxParam;
00012 Matrix eopdata;
00013 Matrix Cnm;
00014 Matrix Snm;
00015 Matrix PC;
00016 Matrix obs;
00017 void AuxParamLoad(){
00018     AuxParam.Mjd_UTC=4.974611635416653e+04;
00019     AuxParam.Mjd_TT=4.974611706231468e+04;
00020     AuxParam.n=20;
00021     AuxParam.m=20;
00022     AuxParam.sun=1;
00023     AuxParam.moon=1;
00024     AuxParam.planets=1;
00025 }
00026 void eop19620101(int c){
00027     eopdata=zeros(13,c);
00028
00029     FILE *fid = fopen("../data/eop19620101.txt","r");
00030     if(fid==NULL){
00031         cout « "Fail open eop19620101.txt file \n";
00032         perror("Error");
00033         exit(EXIT_FAILURE);
00034     }
00035     for (int j=1;j<=c;j++){
00036         fscanf(fid,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
00037             &(eopdata(1,j)),&(eopdata (2,j)),&(eopdata (3,j)),
00038             &(eopdata (4,j)),&(eopdata (5,j)),&(eopdata (6,j)),
00039             &(eopdata (7,j)),&(eopdata (8,j)),&(eopdata (9,j)),
00040             &(eopdata (10,j)),&(eopdata (11,j)),&(eopdata (12,j)),
00041             &(eopdata (13,j))
00042             );
00043     }
00044     fclose(fid);
00045 }
00046
00047 void GGM03S(int n){
00048     Cnm=zeros(n,n);
00049     Snm=zeros(n,n);
00050     FILE *fid = fopen("../data/GGM03S.txt","r");
00051     if(fid==NULL){
00052         cout « "Fail open GGM03S.txt file \n";
00053         perror("Error");
00054         exit(EXIT_FAILURE);
00055     }
00056     double aux;
00057     for(int i=1;i<=n;i++){
00058         for (int j=1;j<=i;j++){
00059             fscanf(fid,"%lf %lf %lf %lf %lf %lf",
00060                 &aux,&aux,
00061                 &Cnm(i,j),&Snm(i,j),
00062                 &aux,&aux
00063                 );
```

```
00064         }
00065     }
00066     fclose(fid);
00067 }
00068 void DE430Coeff(int row,int column){
00069     PC=zeros(row,column);
00070     FILE *fid = fopen("../data/DE430Coeff.txt","r");
00071     if(fid==NULL){
00072         cout « "Fail open DE430Coeff.txt file \n";
00073         perror("Error");
00074         exit(EXIT_FAILURE);
00075     }
00076     double aux;
00077     for(int i=1;i<=row;i++){
00078         for (int j=1;j<=column;j++){
00079             fscanf(fid,"%lf",
00080                 &PC(i,j)
00081                 );
00082         }
00083     }
00084     fclose(fid);
00085 }
00086
00087 void GEOS3(int nobs){
00088     obs=zeros(nobs,4);
00089     FILE *fid = fopen("../data/GEOS3.txt","r");
00090     if(fid==NULL){
00091         cout « "Fail open GEOS3.txt file \n";
00092         perror("Error");
00093         exit(EXIT_FAILURE);
00094     }
00095     int Y,MO,D,H,M,MI,S;
00096     double AZ,EL,DIST;
00097     char tline[57],y[5],mo[3],d[3],h[3],mi[3],s[6],az[10],el[9],dist[10],aux[2];
00098     for (int i=1;i<=nobs;i++)
00099     {
00100         fgets(tline,sizeof(tline),fid);
00101         strncpy(y,&(tline[0]),4);
00102         y[4]='\0';
00103         Y=atoi(y);
00104         strncpy(mo,&(tline[5]),2);
00105         mo[2]='\0';
00106         MO=atoi(mo);
00107         strncpy(d,&(tline[8]),2);
00108         d[2]='\0';
00109         D=atoi(d);
00110         strncpy(h,&(tline[12]),2);
00111         h[2]='\0';
00112         H=atoi(h);
00113         strncpy(mi,&(tline[15]),2);
00114         mi[2]='\0';
00115         MI=atoi(mi);
00116         strncpy(s,&(tline[18]),5);
00117         s[5]='\0';
00118         S=atof(s);
00119         strncpy(az,&(tline[25]),9);
00120         az[9]='\0';
00121         AZ=atof(az);
00122         strncpy(el,&(tline[35]),8);
00123         el[8]='\0';
00124         EL=atof(el);
00125         strncpy(dist,&(tline[44]),9);
00126         dist[9]='\0';
00127         DIST=atof(dist);
00128         obs(i,1) = Mjday(Y,MO,D,H,MI,S);
00129         obs(i,2) = Rad*AZ;
00130         obs(i,3) = Rad*EL;
00131         obs(i,4) = 1e3*DIST;
00132     }
00133     fclose(fid);
00134 }
```

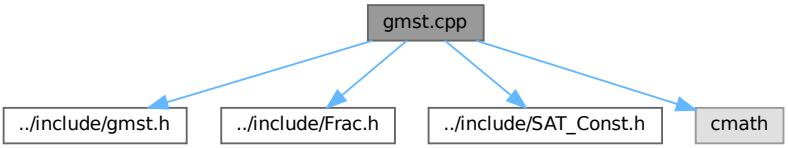
5.99 gmst.cpp File Reference

El archivo contiene las implementaciones de gmst.h.

```
#include "../include/gmst.h"
#include "../include/Frac.h"
#include "../include/SAT_Const.h"
```

```
#include <cmath>
```

Include dependency graph for gmst.cpp:



Functions

- double gmst (double Mjd_UT1)

5.99.1 Detailed Description

El archivo contiene las implementaciones de gmst.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file gmst.cpp.

5.99.2 Function Documentation

5.99.2.1 gmst()

```
double gmst (
            double Mjd_UT1)
```

Greenwich Mean Sidereal Time

Parameters

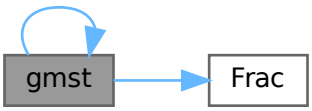
Mjd_UT1	Modified Julian Date UT1

Returns

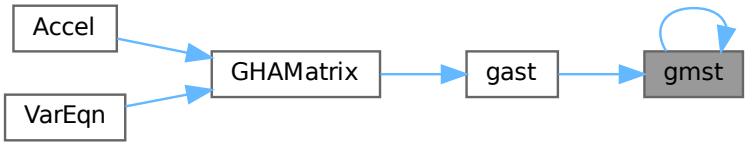
gmstime GMST in [rad]

Definition at line 12 of file gmst.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.100 gmst.cpp

Go to the documentation of this file.

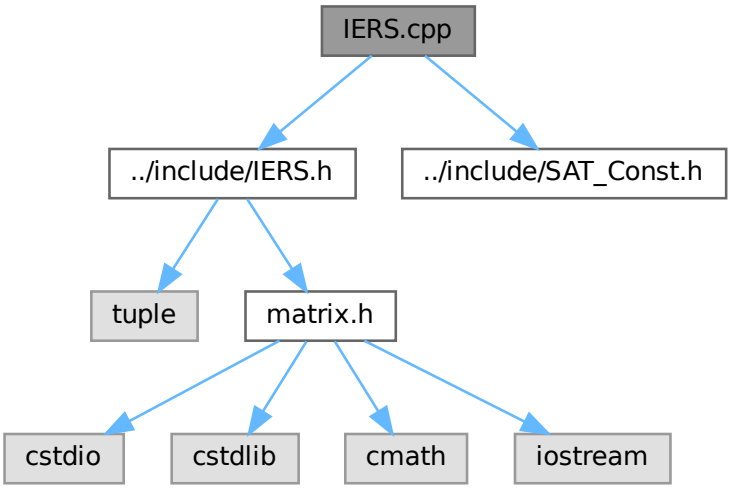
```
00001 #include "../include/gmst.h"
00002 #include "../include/Frac.h"
00003 #include "../include/SAT_Const.h"
00004 #include <cmath>
00005
00012     double gmst(double Mjd_UT1){
00013         double Secs,MJD_J2000,Mjd_0,UT1,T_0,T,gmst;
00014         Secs = 86400.0;                         // Seconds per day
00015         MJD_J2000 = 51544.5;
00016
00017         Mjd_0 = floor(Mjd_UT1);
00018         UT1   = Secs*(Mjd_UT1-Mjd_0);           // [s]
00019         T_0   = (Mjd_0  -MJD_J2000)/36525.0;
00020         T     = (Mjd_UT1-MJD_J2000)/36525.0;
00021
00022         gmst  = 24110.54841 + 8640184.812866*T_0 + 1.002737909350795*UT1 + (0.093104-6.2e-6*T)*T*T;
      // [s]
00023
00024         return 2*pi*Frac(gmst/Secs);       // [rad], 0..2pi
00025
00026     }
```

5.101 IERS.cpp File Reference

El archivo contiene las implementaciones de IERS.h.

```
#include "../include/IERS.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for IERS.cpp:



Functions

- tuple< double, double, double, double, double, double, double, double, double > IERS (Matrix eop, double Mjd_UTC, char interp)

5.101.1 Detailed Description

El archivo contiene las implementaciones de IERS.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file IERS.cpp.

5.101.2 Function Documentation

5.101.2.1 IERS()

```
tuple< double, double, double, double, double, double, double, double, double > IERS (
            Matrix eop,
            double Mjd_UTC,
            char interp = 'n')
```

IERS: Management of IERS time and polar motion data

Parameters

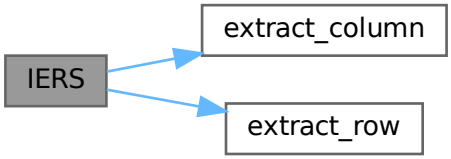
eop	matrix 4x13
Mjd_UTC	double
interp	char

Returns

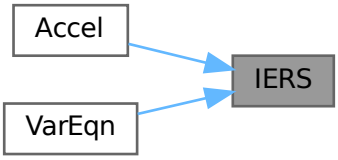
tupla <x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC>

Definition at line 10 of file IERS.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.102 IERS.cpp

Go to the documentation of this file.

```
00001 #include "../include/IERS.h"
00002 #include "../include/SAT_Const.h"
00003
00010     tuple<double,double,double,double,double,double,double,double,double> IERS(Matrix eop,double
      Mjd_UTC,char interp){
00011
00012     double x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC,i,fixf;
00013     int mjd;
00014     Matrix preeop,nexteop,aux;
00015     bool contin=true;
```

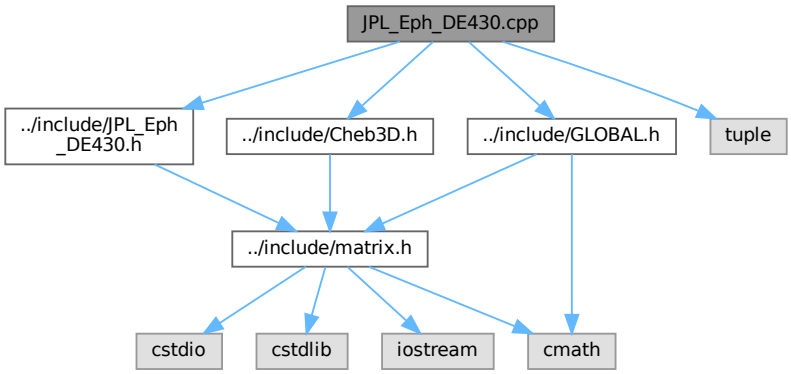
```
00016
00017         mjd = (floor(Mjd_UTC));
00018         i = 1;
00019         aux=extract_row(eop,4);
00020
00021         for (int j = 1;  contin && j <= aux.n_column ; j++) {
00022             if (aux(j)==mjd){
00023                 i = j;
00024                 contin=false;
00025             }
00026         }
00027     if (interp =='l'){
00028         // linear interpolation
00029         preeop = extract_column(eop,i);
00030         nexteop = extract_column(eop,i+1);
00031         fixf = (Mjd_UTC-floor(Mjd_UTC));
00032         // Setting of IERS Earth rotation parameters
00033         // (UT1-UTC [s], TAI-UTC [s], x ["], y ["])
00034         x_pole  = preeop(5)+(nexteop(5)-preeop(5))*fixf;
00035         y_pole  = preeop(6)+(nexteop(6)-preeop(6))*fixf;
00036         UT1_UTC = preeop(7)+(nexteop(7)-preeop(7))*fixf;
00037         LOD     = preeop(8)+(nexteop(8)-preeop(8))*fixf;
00038         dpsi    = preeop(9)+(nexteop(9)-preeop(9))*fixf;
00039         deps    = preeop(10)+(nexteop(10)-preeop(10))*fixf;
00040         dx_pole = preeop(11)+(nexteop(11)-preeop(11))*fixf;
00041         dy_pole = preeop(12)+(nexteop(12)-preeop(12))*fixf;
00042         TAI_UTC = preeop(13);
00043
00044         x_pole  = x_pole/Arcs;  // Pole coordinate [rad]
00045         y_pole  = y_pole/Arcs;  // Pole coordinate [rad]
00046         dpsi    = dpsi/Arcs;
00047         deps    = deps/Arcs;
00048         dx_pole = dx_pole/Arcs; // Pole coordinate [rad]
00049         dy_pole = dy_pole/Arcs; // Pole coordinate [rad]
00050     }
00051     else if (interp =='n')    {
00052         aux = extract_row(eop,i);
00053         eop=aux;
00054         // Setting of IERS Earth rotation parameters
00055         // (UT1-UTC [s], TAI-UTC [s], x ["], y ["])
00056         x_pole  = eop(5)/Arcs;  // Pole coordinate [rad]
00057         y_pole  = eop(6)/Arcs;  // Pole coordinate [rad]
00058         UT1_UTC = eop(7);             // UT1-UTC time difference [s]
00059         LOD     = eop(8);             // Length of day [s]
00060         dpsi    = eop(9)/Arcs;
00061         deps    = eop(10)/Arcs;
00062         dx_pole = eop(11)/Arcs; // Pole coordinate [rad]
00063         dy_pole = eop(12)/Arcs; // Pole coordinate [rad]
00064         TAI_UTC = eop(13);            // TAI-UTC time difference [s]
00065     }
00066
00067 return tie(x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC);
00068
00069     }
```

5.103 JPL_Eph_DE430.cpp File Reference

El archivo contiene las implementaciones de JPL_Eph_DE430.h.

```
#include "../include/JPL_Eph_DE430.h"
#include "../include/Cheb3D.h"
#include "../include/GLOBAL.h"
#include <tuple>
```

Include dependency graph for JPL_Eph_DE430.cpp:



Functions

- tuple< Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix & > JPL_Eph_DE430 (double Mjd_TDB)

5.103.1 Detailed Description

El archivo contiene las implementaciones de JPL_Eph_DE430.h.

Author

Pedro Zhuzhan

Bug Noknownbugs

Definition in file JPL_Eph_DE430.cpp.

5.103.2 Function Documentation

5.103.2.1 JPL_Eph_DE430()

```
tuple< Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix &, Matrix
&, Matrix &, Matrix & > JPL_Eph_DE430 (
            double Mjd_TDB)
```

Parameters

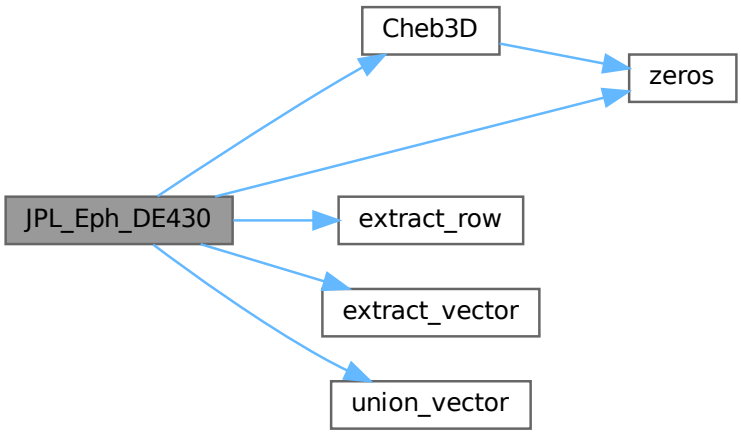
P	Matrix
Phi	Matrix
Qdt	Matrix

Returns

Matrix

Definition at line 12 of file JPL_Eph_DE430.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.104 JPL_Eph_DE430.cpp

Go to the documentation of this file.

```
00001 #include"../include/JPL_Eph_DE430.h"
00002 #include"../include/Cheb3D.h"
00003 #include"../include/GLOBAL.h"
00004 #include<tuple>
00005
00012 tuple<Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&,Matrix&>
      JPL_Eph_DE430(double Mjd_TDB){
00013
00014 double JD,t1,dt,j,Mjd0;
00015 Matrix Cx_Earth(12),Cy_Earth(12),Cz_Earth(12),Cx(12),Cy(12),Cz(12),
00016
      Cx_Moon(12),Cy_Moon(12),Cz_Moon(12),temp(4),Cx_Sun(12),Cy_Sun(12),Cz_Sun(12),Cx_Venus(12),Cy_Venus(12),Cz_Venus(12),Cx_M
00017
      Cx_Mars(12),Cy_Mars(12),Cz_Mars(12),Cx_Jupiter(12),Cy_Jupiter(12),Cz_Jupiter(12),Cx_Saturn(12),Cy_Saturn(12),Cz_Saturn(1
00018 ,Cx_Neptune(12),Cy_Neptune(12),Cz_Neptune(12),Cx_Pluto(12),Cy_Pluto(12),Cz_Pluto(12),
```

```
00019 &r_Mercury=zeros(3),&r_Venus=zeros(3),&r_Earth=zeros(3),&r_Mars=zeros(3),&r_Jupiter=zeros(3),
00020
      &r_Saturn=zeros(3),&r_Uranus=zeros(3),&r_Neptune=zeros(3),&r_Pluto=zeros(3),&r_Moon=zeros(3),&r_Sun=zeros(3);
00021 JD=Mjd_TDB+2400000.5;
00022 int i;
00023 for(i=1;i<=PC.n_row;i++){
00024     if(PC(i,1)<=JD&&JD<=PC(i,2)){
00025         break;
00026     }
00027 }
00028 Matrix PCtemp=extract_row(PC,i);
00029 t1=PCtemp(1)-2400000.5;
00030
00031 dt=Mjd_TDB-t1;
00032 for(int k=1;k<=4;k++){
00033     temp(k)=231+(k-1)*13;
00034 }
00035     Cx_Earth=extract_vector(PCtemp,temp(1),temp(2)-1);
00036     Cy_Earth=extract_vector(PCtemp,temp(2),temp(3)-1);
00037     Cz_Earth=extract_vector(PCtemp,temp(3),temp(4)-1);
00038     temp=temp+39;
00039     Cx=extract_vector(PCtemp,temp(1),temp(2)-1);
00040     Cy=extract_vector(PCtemp,temp(2),temp(3)-1);
00041     Cz=extract_vector(PCtemp,temp(3),temp(4)-1);
00042     Cx_Earth=union_vector(Cx_Earth,Cx);
00043     Cy_Earth=union_vector(Cy_Earth,Cy);
00044     Cz_Earth=union_vector(Cz_Earth,Cz);
00045 if(0<=dt&&dt<=16){
00046     j=0;
00047 Mjd0=t1;
00048 }
00049 else if(16<dt&&dt<=32){
00050     j=1;
00051 Mjd0=t1+16*j;
00052 }
00053
      r_Earth=Cheb3D(Mjd_TDB,13,Mjd0,Mjd0+16,extract_vector(Cx_Earth,13*j+1,13*j+13),extract_vector(Cy_Earth,13*j+1,13*j+13),e
00054
00055
00056 for(int k=1;k<=4;k++){
00057     temp(k)=441+(k-1)*13;
00058 }
00059 Cx_Moon=extract_vector(PCtemp,temp(1),temp(2)-1);
00060 Cy_Moon=extract_vector(PCtemp,temp(2),temp(3)-1);
00061 Cz_Moon=extract_vector(PCtemp,temp(3),temp(4)-1);
00062 for(i=1;i<7;i++){
00063     temp=temp+39;
00064 Cx=extract_vector(PCtemp,temp(1),temp(2)-1);
00065 Cy=extract_vector(PCtemp,temp(2),temp(3)-1);
00066 Cz=extract_vector(PCtemp,temp(3),temp(4)-1);
00067 Cx_Moon=union_vector(Cx_Moon,Cx);
00068 Cy_Moon=union_vector(Cy_Moon,Cy);
00069 Cz_Moon=union_vector(Cz_Moon,Cz);
00070 }
00071 if(0<=dt&&dt<=4){
00072 j=0;
00073 Mjd0=t1;}
00074 else if(4<dt&&dt<=8){
00075 j=1;
00076 Mjd0=t1+4*j;}
00077 else if(8<dt&&dt<=12){
00078 j=2;
00079 Mjd0=t1+4*j;}
00080 else if(12<dt&&dt<=16){
00081 j=3;
00082 Mjd0=t1+4*j;}
00083 else if(16<dt&&dt<=20){
00084 j=4;
00085 Mjd0=t1+4*j;}
00086 else if(20<dt&&dt<=24){
00087 j=5;
00088 Mjd0=t1+4*j;}
00089 else if(24<dt&&dt<=28){
00090 j=6;
00091 Mjd0=t1+4*j;}
00092 else if(28<dt&&dt<=32){
00093 j=7;
00094 Mjd0=t1+4*j;}
00095
      r_Moon=Cheb3D(Mjd_TDB,13,Mjd0,Mjd0+4,extract_vector(Cx_Moon,13*j+1,13*j+13),extract_vector(Cy_Moon,13*j+1,13*j+13),extra
00096 for(int k=1;k<=4;k++){
00097     temp(k)=753+(k-1)*11;
00098 }
00099 Cx_Sun=extract_vector(PCtemp,temp(1),temp(2)-1);
00100 Cy_Sun=extract_vector(PCtemp,temp(2),temp(3)-1);
00101 Cz_Sun=extract_vector(PCtemp,temp(3),temp(4)-1);
00102 temp=temp+33;
```

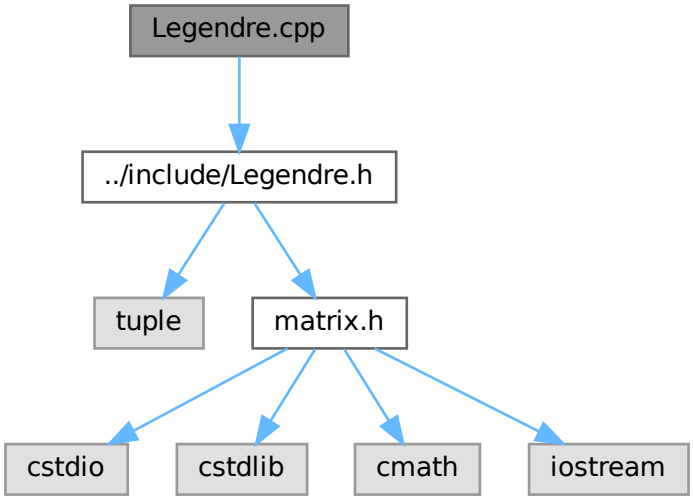
```
00103 Cx=extract_vector(PCtemp,temp(1),temp(2)-1);
00104 Cy=extract_vector(PCtemp,temp(2),temp(3)-1);
00105 Cz=extract_vector(PCtemp,temp(3),temp(4)-1);
00106 Cx_Sun=union_vector(Cx_Sun,Cx);
00107 Cy_Sun=union_vector(Cy_Sun,Cy);
00108 Cz_Sun=union_vector(Cz_Sun,Cz);
00109 if(0<=dt&&dt<=16){
00110 j=0;
00111 Mjd0=t1;}
00112 else if(16<dt&&dt<=32){
00113 j=1;
00114 Mjd0=t1+16*j;}
00115
      r_Sun=Cheb3D(Mjd_TDB,11,Mjd0,Mjd0+16,extract_vector(Cx_Sun,11*j+1,11*j+11),extract_vector(Cy_Sun,11*j+1,11*j+11),extract
00116 for(int k=1;k<=4;k++){
00117     temp(k)=3+(k-1)*14;
00118 }
00119 Cx_Mercury=extract_vector(PCtemp,temp(1),temp(2)-1);
00120 Cy_Mercury=extract_vector(PCtemp,temp(2),temp(3)-1);
00121 Cz_Mercury=extract_vector(PCtemp,temp(3),temp(4)-1);
00122 temp=temp+42;
00123 Cx=extract_vector(PCtemp,temp(1),temp(2)-1);
00124 Cy=extract_vector(PCtemp,temp(2),temp(3)-1);
00125 Cz=extract_vector(PCtemp,temp(3),temp(4)-1);
00126 Cx_Mercury=union_vector(Cx_Mercury,Cx);
00127 Cy_Mercury=union_vector(Cy_Mercury,Cy);
00128 Cz_Mercury=union_vector(Cz_Mercury,Cz);
00129 if(0<=dt&&dt<=8){
00130 j=0;
00131 Mjd0=t1;}
00132 else if(8<dt&&dt<=16){
00133 j=1;
00134 Mjd0=t1+8*j;}
00135 else if(16<dt&&dt<=24){
00136 j=2;
00137 Mjd0=t1+8*j;}
00138 else if(24<dt&&dt<=32){
00139 j=3;
00140 Mjd0=t1+8*j;}
00141
      r_Mercury=Cheb3D(Mjd_TDB,14,Mjd0,Mjd0+8,extract_vector(Cx_Mercury,14*j+1,14*j+14),extract_vector(Cy_Mercury,14*j+1,14*j
00142 for(int k=1;k<=4;k++){
00143     temp(k)=171+(k-1)*10;
00144 }
00145 Cx_Venus=extract_vector(PCtemp,temp(1),temp(2)-1);
00146 Cy_Venus=extract_vector(PCtemp,temp(2),temp(3)-1);
00147 Cz_Venus=extract_vector(PCtemp,temp(3),temp(4)-1);
00148 temp=temp+30;
00149 Cx=extract_vector(PCtemp,temp(1),temp(2)-1);
00150 Cy=extract_vector(PCtemp,temp(2),temp(3)-1);
00151 Cz=extract_vector(PCtemp,temp(3),temp(4)-1);
00152 Cx_Venus=union_vector(Cx_Venus,Cx);
00153 Cy_Venus=union_vector(Cy_Venus,Cy);
00154 Cz_Venus=union_vector(Cz_Venus,Cz);
00155 if(0<=dt&&dt<=16){
00156 j=0;
00157 Mjd0=t1;}
00158 else if(16<dt&&dt<=32){
00159 j=1;
00160 Mjd0=t1+16*j;}
00161
      r_Venus=Cheb3D(Mjd_TDB,10,Mjd0,Mjd0+16,extract_vector(Cx_Venus,10*j+1,10*j+10),extract_vector(Cy_Venus,10*j+1,10*j+10),e
00162 for(int k=1;k<=4;k++){
00163     temp(k)=309+(k-1)*11;
00164 }
00165 Cx_Mars=extract_vector(PCtemp,temp(1),temp(2)-1);
00166 Cy_Mars=extract_vector(PCtemp,temp(2),temp(3)-1);
00167 Cz_Mars=extract_vector(PCtemp,temp(3),temp(4)-1);
00168 Mjd0=t1;
00169 r_Mars=Cheb3D(Mjd_TDB,11,Mjd0,Mjd0+32,Cx_Mars,Cy_Mars,Cz_Mars)*1e3;
00170
00171 for(int k=1;k<=4;k++){
00172     temp(k)=342+(k-1)*8;
00173 }
00174 Cx_Jupiter=extract_vector(PCtemp,temp(1),temp(2)-1);
00175 Cy_Jupiter=extract_vector(PCtemp,temp(2),temp(3)-1);
00176 Cz_Jupiter=extract_vector(PCtemp,temp(3),temp(4)-1);
00177 Mjd0=t1;
00178 r_Jupiter=Cheb3D(Mjd_TDB,8,Mjd0,Mjd0+32,Cx_Jupiter,Cy_Jupiter,Cz_Jupiter)*1e3;
00179
00180 for(int k=1;k<=4;k++){
00181     temp(k)=366+(k-1)*7;
00182 }
00183 Cx_Saturn=extract_vector(PCtemp,temp(1),temp(2)-1);
00184 Cy_Saturn=extract_vector(PCtemp,temp(2),temp(3)-1);
00185 Cz_Saturn=extract_vector(PCtemp,temp(3),temp(4)-1);
00186 Mjd0=t1;
```

```
00187 r_Saturn=Cheb3D(Mjd_TDB,7,Mjd0,Mjd0+32,Cx_Saturn,
00188 Cy_Saturn,Cz_Saturn)*1e3;
00189
00190 for(int k=1;k<=4;k++){
00191     temp(k)=387+(k-1)*6;
00192 }
00193 Cx_Uranus=extract_vector(PCtemp,temp(1),temp(2)-1);
00194 Cy_Uranus=extract_vector(PCtemp,temp(2),temp(3)-1);
00195 Cz_Uranus=extract_vector(PCtemp,temp(3),temp(4)-1);
00196 Mjd0=t1;
00197 r_Uranus=Cheb3D(Mjd_TDB,6,Mjd0,Mjd0+32,Cx_Uranus,
00198 Cy_Uranus,Cz_Uranus)*1e3;
00199
00200 for(int k=1;k<=4;k++){
00201     temp(k)=405+(k-1)*6;
00202 }
00203 Cx_Neptune=extract_vector(PCtemp,temp(1),temp(2)-1);
00204 Cy_Neptune=extract_vector(PCtemp,temp(2),temp(3)-1);
00205 Cz_Neptune=extract_vector(PCtemp,temp(3),temp(4)-1);
00206 Mjd0=t1;
00207 r_Neptune=Cheb3D(Mjd_TDB,6,Mjd0,Mjd0+32,Cx_Neptune,
00208 Cy_Neptune,Cz_Neptune)*1e3;
00209 for(int k=1;k<=4;k++){
00210     temp(k)=423+(k-1)*6;
00211 }
00212 Cx_Pluto=extract_vector(PCtemp,temp(1),temp(2)-1);
00213 Cy_Pluto=extract_vector(PCtemp,temp(2),temp(3)-1);
00214 Cz_Pluto=extract_vector(PCtemp,temp(3),temp(4)-1);
00215 Mjd0=t1;
00216 r_Pluto=Cheb3D(Mjd_TDB,6,Mjd0,Mjd0+32,Cx_Pluto,Cy_Pluto,Cz_Pluto)*1e3;
00217 double EMRAT=81.30056907419062;
00218 double EMRAT1=1/(1+EMRAT);
00219 Matrix aux(3);
00220 aux=r_Moon;
00221  r_Earth=r_Earth-aux*EMRAT1;
00222  aux=r_Earth;
00223  aux=aux*(-1);
00224  r_Mercury=aux+r_Mercury;
00225  r_Venus=aux+r_Venus;
00226  r_Mars=aux+r_Mars;
00227  r_Jupiter=aux+r_Jupiter;
00228  r_Saturn=aux+r_Saturn;
00229  r_Uranus=aux+r_Uranus;
00230  r_Neptune=aux+r_Neptune;
00231  r_Pluto=aux+r_Pluto;
00232  r_Sun=aux+r_Sun;
00233  return
     tie(r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus,r_Neptune,r_Pluto,r_Moon,r_Sun);
00234     }
```

5.105 Legendre.cpp File Reference

El archivo contiene las implementaciones de Legendre.h.

```
#include "../include/Legendre.h"
```
Include dependency graph for Legendre.cpp:



Functions

- tuple< Matrix &, Matrix & > Legendre (int n, int m, double fi)

5.105.1 Detailed Description

El archivo contiene las implementaciones de Legendre.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Legendre.cpp.

5.105.2 Function Documentation

5.105.2.1 Legendre()

```
tuple< Matrix &, Matrix & > Legendre (
            int n,
            int m,
            double fi)
```

Time differences [s]

Parameters

n	int
m	int
fi	double [rad]

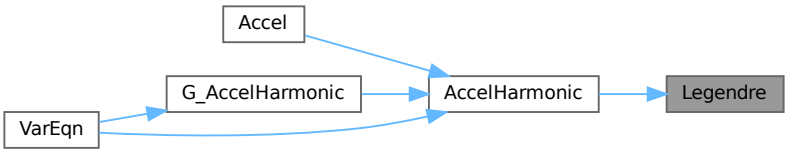
Returns

tupla <pnm,dpnm>

Definition at line 8 of file Legendre.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



5.106 Legendre.cpp

Go to the documentation of this file.

```
00001 #include "../include/Legendre.h"
00008     tuple<Matrix&,Matrix&>  Legendre(int n,int m,double fi){
00009     Matrix &pnm = zeros(n+1,m+1);
00010     Matrix &dpnm = zeros(n+1,m+1);
00011     int i=0;
00012 
00013     pnm(1,1)=1;
00014     dpnm(1,1)=0;
00015     pnm(2,2)=sqrt(3)*cos(fi);
00016     dpnm(2,2)=-sqrt(3)*sin(fi);
00017     // diagonal coefficients
00018     for (i=2;i<=n;i++){
00019         pnm(i+1,i+1)= sqrt((2.0*i+1.0)/(2.0*i))*cos(fi)*pnm(i,i);
00020     }
00021     for (i=2;i<=n;i++){
00022         dpnm(i+1,i+1)= sqrt((2.0*i+1.0)/(2.0*i))*((cos(fi)*dpnm(i,i))-(sin(fi)*pnm(i,i)));}
00023     // horizontal first step coefficients
00024     for (i=1;i<=n;i++){
```

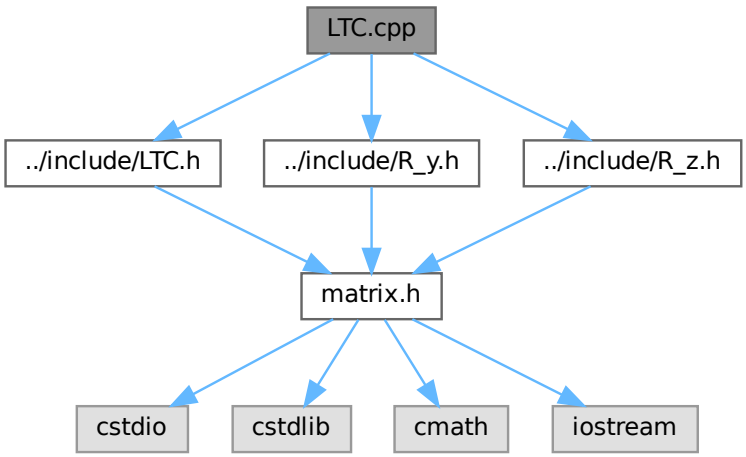
```
00025         pnm(i+1,i)= sqrt(2.0*i+1)*sin(fi)*pnm(i,i);}
00026     for (i=1;i<=n;i++){
00027         dpnm(i+1,i)= sqrt(2.0*i+1)*((cos(fi)*pnm(i,i))+(sin(fi)*dpnm(i,i)));}
00028     // horizontal second step coefficients
00029     int j=0;
00030     int k=2;
00031     while(true){
00032         for (i=k;i<=n;i++){
00033             pnm(i+1,j+1)=sqrt((2.0*i+1.0)/((i-j)*(i+j)))*((sqrt(2.0*i-1.0)*sin(fi)*pnm(i,j+1.0))
      -(sqrt(((i+j-1.0)*(i-j-1.0))/(2.0*i-3.0))*pnm(i-1,j+1.0)));}
00034         j = j+1;
00035         k = k+1;
00036         if (j>m){
00037             break;
00038         }
00039     }
00040
00041     j = 0;
00042     k = 2;
00043     while(true){
00044         for (i=k;i<=n;i++){
00045
      dpnm(i+1,j+1)=sqrt((2.0*i+1.0)/((i-j)*(i+j)))*((sqrt(2.0*i-1)*sin(fi)*dpnm(i,j+1))+(sqrt(2.0*i-1.0)*cos(fi)*pnm(i,j+1.0
00046                 -(sqrt(((i+j-1.0)*(i-j-1))/(2.0*i-3.0))*dpnm(i-1,j+1)));}
00047         j = j+1;
00048         k = k+1;
00049         if (j>m){
00050             break;
00051         }
00052
00053         }
00054         return tie(pnm, dpnm);
00055 }
```

5.107 LTC.cpp File Reference

El archivo contiene las implementaciones de LTC.h.

```
#include "../include/LTC.h"
#include "../include/R_y.h"
#include "../include/R_z.h"
```

Include dependency graph for LTC.cpp:



Functions

- Matrix & LTC (double lon, double lat)

5.107.1 Detailed Description

El archivo contiene las implementaciones de LTC.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file LTC.cpp.

5.107.2 Function Documentation

5.107.2.1 LTC()

```
Matrix & LTC (
            double lon,
            double lat)
```

Transformation from Greenwich meridian system to local tangent coordinates

Parameters

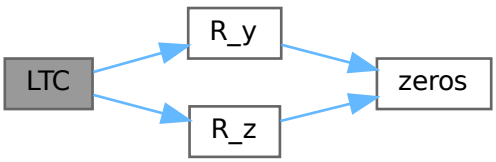
lon	-Geodetic East longitude [rad]
lat	-Geodetic latitude [rad]

Returns

M -Rotation matrix from the Earth equator and Greenwich meridian to the local tangent (East-North-Zenith) coordinate system

Definition at line 10 of file LTC.cpp.

Here is the call graph for this function:



5.108 LTC.cpp

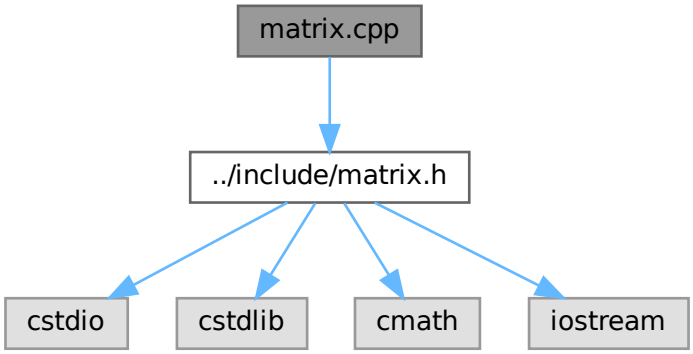
Go to the documentation of this file.

```
00001 #include "../include/LTC.h"
00002 #include "../include/R_y.h"
00003 #include "../include/R_z.h"
00010     Matrix& LTC(double lon,double lat){
00011         Matrix &M= R_y(-1.0*lat)*R_z(lon);
00012         double Aux;
00013     for (int j=1;j<=3;j++){
00014         Aux=M(1,j);
00015         M(1,j)=M(2,j);
00016         M(2,j)=M(3,j);
00017         M(3,j)= Aux;}
00018     return M;
00019 
00020     }
```

5.109 matrix.cpp File Reference

El archivo contiene las implementaciones de matrix.h.

`#include "../include/matrix.h"`
Include dependency graph for matrix.cpp:



Functions

- ostream & operator<< (ostream &o, Matrix &m)
- Matrix & zeros (const int n_row, const int n_column)
- Matrix & eye (const int size)
- Matrix & transpose (Matrix &m)
- void swap_row (Matrix &m, int i, int index)
- Matrix & inv (Matrix &m)
- Matrix & zeros (const int n)
- double norm (Matrix &m)
- double dot (Matrix &v, Matrix &w)
- Matrix & cross (Matrix &v, Matrix &w)
- Matrix & extract_vector (Matrix &v, int start, int end)

- Matrix & union_vector (Matrix &v, Matrix &w)
- Matrix & extract_row (Matrix &v, int j)
- Matrix & extract_column (Matrix &v, int j)
- Matrix & assign_row (Matrix &v, Matrix &w, int j)
- Matrix & assign_column (Matrix &v, Matrix &w, int j)

5.109.1 Detailed Description

El archivo contiene las implementaciones de matrix.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file matrix.cpp.

5.109.2 Function Documentation

5.109.2.1 assign_column()

```
Matrix & assign_column (
            Matrix & v,
            Matrix & w,
            int i)
```

Asigna la columna i-1 con w y lo devuelve

Parameters

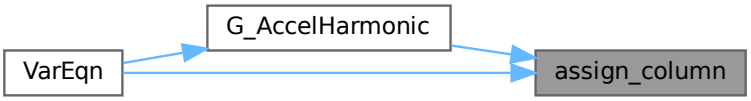
v	Matrix con tamaño x x y ,x pertenece a N y pertenece a N
w	Matrix con tamaño 1 x y ,y pertenece a N
i	es la columna tiene que ser >=1 && <=v.n_column

Returns

Matrix v con la columna i-1 cambiada por los elementos de w

Definition at line 424 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.2 assign_row()

```
Matrix & assign_row (
            Matrix & v,
            Matrix & w,
            int i)
```

Asigna la fila i-1 con w y lo devuelve

Parameters

v	Matrix con tamaño x x y ,x pertenece a N y pertenece a N
w	Matrix con tamaño 1 x y ,y pertenece a N
i	es la fila tiene que ser >=1 && <=v.n_row

Returns

Matrix v con la fila i-1 cambiada por los elementos de w

Definition at line 412 of file matrix.cpp.

5.109.2.3 cross()

```
Matrix & cross (
            Matrix & v,
            Matrix & w)
```

Devulve el producto vectorial

Parameters

v	Matrix con tamaño 1 x 3
w	Matrix con tamaño 1 x 3

Returns

producto escalar de v x w

Definition at line 353 of file matrix.cpp.

5.109.2.4 dot()

```
double dot (
            Matrix & v,
            Matrix & w)
```

Devulve el producto escalar

Parameters

v	Matrix con tamaño 1 x 3
w	Matrix con tamaño 1 x 3

Returns

producto escalar de v·w

Definition at line 342 of file matrix.cpp.

Here is the caller graph for this function:

5.109.2.5 extract_column()

```
Matrix & extract_column (
            Matrix & v,
            int i)
```

Extrae la columna i-1 de v y lo devuelve

Parameters

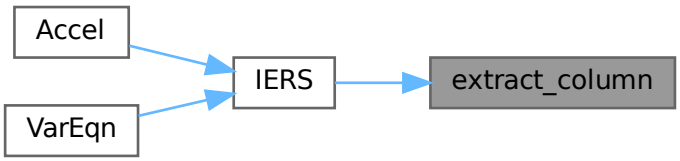
v	Matrix
i	es la columna tiene que ser >=1 && <=v.n_column

Returns

Matrix con la columna i-1 de v

Definition at line 401 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.6 extract_row()

```
Matrix & extract_row (
            Matrix & v,
            int i)
```

Extrae la fila i-1 de v y lo devuelve

Parameters

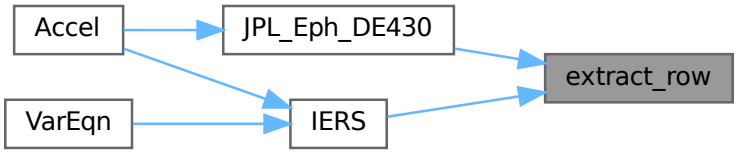
v	Matrix
i	es la fila tiene que ser >=1 && <=v.n_row

Returns

Matrix con la fila i-1 de v

Definition at line 389 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.7 extract_vector()

```
Matrix & extract_vector (
            Matrix & v,
            int start,
            int end)
```

Extrae del vector v desde la posicion start hasta end, incluidos

Parameters

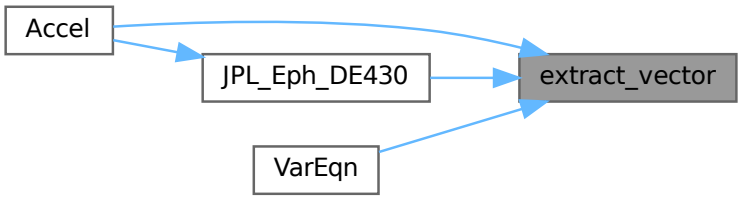
v	Matrix 1 x n
start	inicio del vector resultado
end	fin del vector resultado

Returns

Matrix 1 x (end - start + 1)

Definition at line 364 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.8 eye()

```
Matrix & eye (
            const int size)
```

Crea una Matrix identidad con tamaño size x size

Parameters

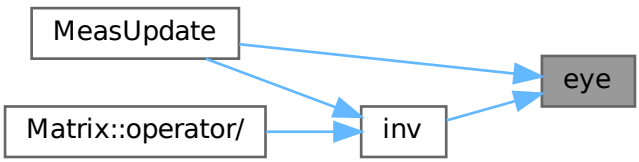
size	dimension de la matriz

Returns

una Matrix tamaño size x size

Definition at line 233 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.9 inv()

```
Matrix & inv (
            Matrix & m)
```

Crea una Matrix inversa de m, sin modificar m

Parameters

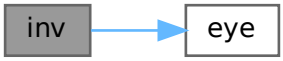
m	Matrix que tiene que ser cuadrada, es decir, con el mismo numero de columnas que filas

Returns

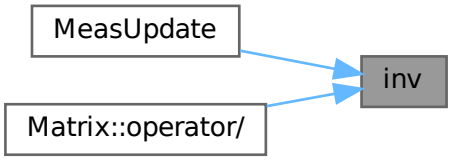
una Matrix inversa de m

Definition at line 267 of file matrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.109.2.10 norm()

```
double norm (
            Matrix & m)
```

Devulve la norma 2 de una Matrix que simula un vector

Parameters

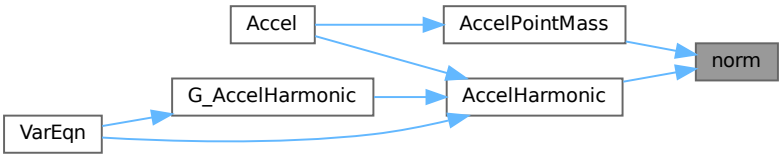
m	Matrix 1 x n_column

Returns

la norma 2 de m

Definition at line 333 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.11 operator<<()

```
ostream & operator<< (
            ostream & o,
            Matrix & m)
```

Definition at line 212 of file matrix.cpp.

5.109.2.12 swap_row()

```
void swap_row (
            Matrix & m,
            int i,
            int index)
```

Definition at line 258 of file matrix.cpp.

5.109.2.13 transpose()

```
Matrix & transpose (
            Matrix & m)
```

Crea una Matrix traspuesta de m,sin modificar m

Parameters

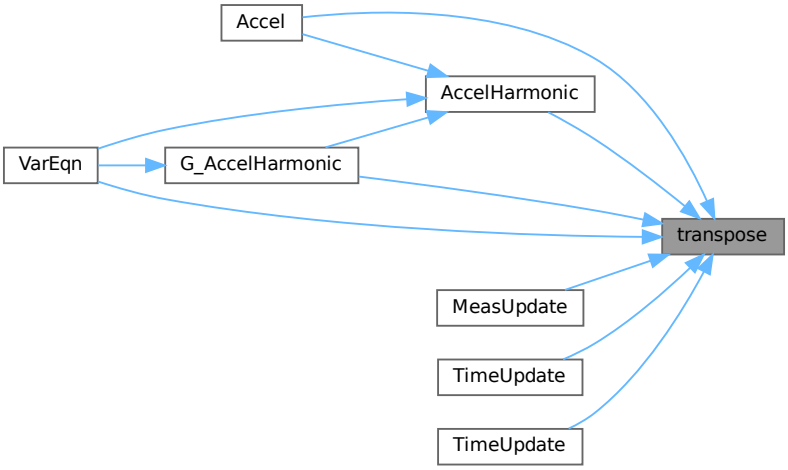
m	Matrix

Returns

una Matrix traspuesta de m

Definition at line 246 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.14 union_vector()

```
Matrix & union_vector (
            Matrix & v,
            Matrix & w)
```

Devuelve la matriz resultado de realizar la union entre v y w

Parameters

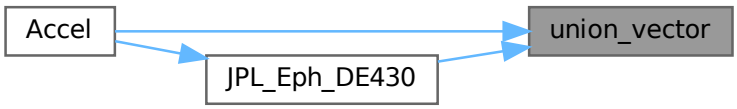
v	Matrix con tamaño 1 x n ,n pertenece a N
w	Matrix con tamaño 1 x n ,n pertenece a N

Returns

producto escalar de v x w

Definition at line 375 of file matrix.cpp.

Here is the caller graph for this function:



5.109.2.15 zeros() [1/2]

```
Matrix & zeros (
            const int n)
```

Crea una Matrix con todas sus componentes a 0

Parameters

n	numero de columnas que tiene la matriz

Returns

una Matrix tamaño 1 x n

Definition at line 323 of file matrix.cpp.

5.109.2.16 zeros() [2/2]

```
Matrix & zeros (
            const int n_row,
            const int n_column)
```

Crea una Matrix con todas sus componentes a 0

Parameters

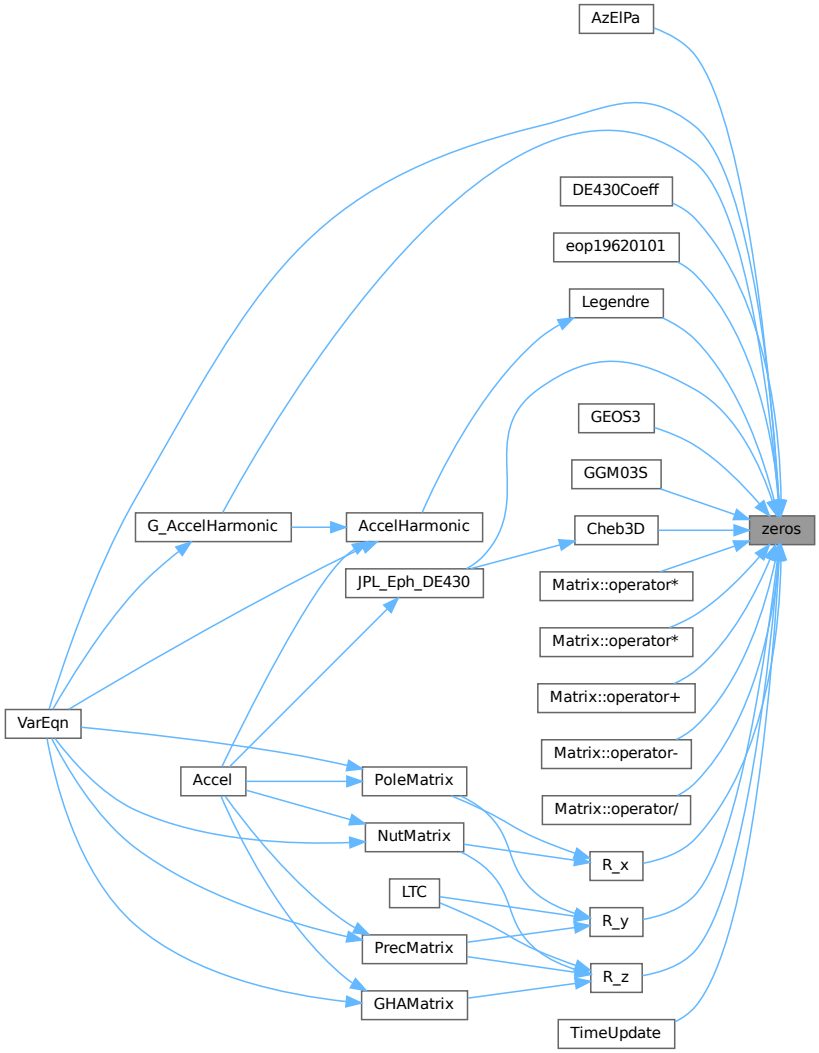
n_row	numero de filas que tiene la matriz
n_column	numero de columnas que tiene la matriz

Returns

una Matrix tamaño n_row x n_column

Definition at line 222 of file matrix.cpp.

Here is the caller graph for this function:



5.110 matrix.cpp

Go to the documentation of this file.

```
00001 #include "../include/matrix.h"
00002
00009 //----------------------------------
00010 Matrix::Matrix() {
00011     this->n_row = 0;
00012     this->n_column = 0;
00013     this->data = NULL;
00014
00015 }
```

```
00016 //---------------------------------
00017 Matrix::Matrix(const int n_size) {
00018     if (n_size <= 0) {
00019         cout « "Vector create: error in n_row/n_column\n";
00020         exit(EXIT_FAILURE);
00021     }
00022
00023     this->n_row = 1;
00024     this->n_column = n_size;
00025     this->data = (double **) malloc(n_row*sizeof(double *));
00026
00027     if (this->data == NULL) {
00028         cout « "Vector create: error in data\n";
00029         exit(EXIT_FAILURE);
00030     }
00031     this->data[0] = (double *) calloc(n_size,sizeof(double));
00032
00033 }
00034 //---------------------------------------
00035 Matrix::Matrix(const int n_row, const int n_column) {
00036     if (n_row <= 0 || n_column <= 0) {
00037         cout « "Matrix create: error in n_row/n_column\n";
00038         exit(EXIT_FAILURE);
00039     }
00040
00041     this->n_row = n_row;
00042     this->n_column = n_column;
00043     this->data = (double **) malloc(n_row*sizeof(double *));
00044
00045     if (this->data == NULL) {
00046         cout « "Matrix create: error in data\n";
00047         exit(EXIT_FAILURE);
00048     }
00049
00050     for(int i = 0; i < n_row; i++) {
00051         this->data[i] = (double *) malloc(n_column*sizeof(double));
00052     }
00053 }
00054 //---------------------------------
00055 double& Matrix::operator () (const int n) {
00056     if (n <= 0 || n > this->n_column* this->n_row) {
00057         cout « "Vector get: error in get:"«n«" size: " «this->n_column*this->n_row«" row/column\n";
00058         exit(EXIT_FAILURE);
00059     }
00060
00061     return this->data[(n - 1)/this->n_column][(n-1)%this->n_column];
00062 }
00063 //---------------------------------
00064 double& Matrix::operator () (const int row, const int column) {
00065     if (row <= 0 || row > this->n_row || column <= 0 || column > this->n_column) {
00066         cout « "Matrix get: error in " «row«" row/ "«column«" column\n";
00067         cout « "Matrix " «this->n_row«" n_row/ "«this->n_column«" n_column\n";
00068         exit(EXIT_FAILURE);
00069     }
00070
00071     return this->data[row - 1][column - 1];
00072 }
00073 //---------------------------------
00074 Matrix& Matrix::operator + (Matrix &m) {
00075     if (this->n_row != m.n_row || this->n_column != m.n_column) {
00076         cout « "Matrix sum: error in n_row/n_column\n";
00077         exit(EXIT_FAILURE);
00078     }
00079
00080     Matrix *m_aux = new Matrix(this->n_row, this->n_column);
00081
00082     for(int i = 1; i <= this->n_row; i++) {
00083         for(int j = 1; j <= this->n_column; j++) {
00084             (*m_aux)(i,j) = (*this)(i,j) + m(i,j);
00085         }
00086     }
00087
00088     return *m_aux;
00089 }
00090 //---------------------------------
00091 Matrix& Matrix::operator - (Matrix &m) {
00092     if (this->n_row != m.n_row || this->n_column != m.n_column) {
00093         cout « "Matrix sub: error in n_row/n_column\n";
00094         exit(EXIT_FAILURE);
00095     }
00096
00097     Matrix *m_aux = new Matrix(this->n_row, this->n_column);
00098
00099     for(int i = 1; i <= this->n_row; i++) {
00100         for(int j = 1; j <= this->n_column; j++) {
00101             (*m_aux)(i,j) = (*this)(i,j) - m(i,j);
00102         }
```

```
00103     }
00104
00105     return *m_aux;
00106 }
00107 //---------------------------------
00108 Matrix& Matrix::operator * (Matrix &m){
00109     if (this->n_column != m.n_row) {
00110         cout « "Matrix muliplication: error in n->n_column, m.n_row\n";
00111         exit(EXIT_FAILURE);
00112     }
00113
00114     Matrix &m_aux=zeros(this->n_row, m.n_column);
00115
00116     for(int i = 1; i <= this->n_row; i++) {
00117         for(int j = 1; j <= m.n_column; j++) {
00118                     m_aux(i,j)=0;
00119             for(int k = 1; k <= this->n_column; k++) {
00120                 m_aux(i,j) += (*this)(i,k) * m(k,j);
00121             }
00122         }
00123     }
00124     return m_aux;
00125 }
00126 //---------------------------------
00127 Matrix& Matrix::operator / (Matrix &m){
00128     if (this->n_column != m.n_row) {
00129         cout « "Matrix sub: error in n->n_column, m.n_row\n";
00130         exit(EXIT_FAILURE);
00131     }
00132
00133     Matrix *m_aux= new Matrix(this->n_row,m.n_column);
00134     *m_aux=(*this)*inv(m);
00135     return *m_aux;
00136 }
00137 //---------------------------------
00138 Matrix& Matrix::operator = (Matrix &m){
00139
00140     Matrix *m_aux = new Matrix(m.n_row, m.n_column);
00141
00142     for (int i = 1; i <= m.n_row; i++) {
00143         for (int j = 1; j <= m.n_column; j++){
00144                     (*m_aux)(i,j)=m(i,j);
00145         }
00146     }
00147
00148     this->n_row = m.n_row;
00149     this->n_column = m.n_column;
00150     this->data = (double **) malloc(m.n_row*sizeof(double *));
00151
00152     if (this->data == NULL) {
00153         cout « "Matrix create: error in data\n";
00154         exit(EXIT_FAILURE);
00155     }
00156
00157     for(int i = 0; i < m.n_row; i++) {
00158         this->data[i] = (double *) malloc(m.n_column*sizeof(double));
00159     }
00160     for (int i = 1; i <= this->n_row; i++) {
00161         for (int j = 1; j <= this->n_column; j++){
00162                     (*this)(i,j)=(*m_aux)(i,j);
00163         }
00164     }
00165     return *this;
00166 }
00167 //---------------------------------
00168 Matrix& Matrix::operator + (double d){
00169
00170     Matrix &m_aux=zeros(this->n_row, this->n_column);
00171
00172     for(int i = 1; i <= this->n_row; i++) {
00173         for(int j = 1; j <= this->n_column; j++) {
00174             m_aux(i,j) = (*this)(i,j) + d;
00175         }
00176     }
00177
00178     return m_aux;
00179 }
00180 Matrix& Matrix::operator - (double d){
00181
00182     Matrix &m_aux=zeros(this->n_row, this->n_column);
00183
00184     for(int i = 1; i <= this->n_row; i++) {
00185         for(int j = 1; j <= this->n_column; j++) {
00186             (m_aux)(i,j) =(*this)(i,j)- d;
00187         }
00188     }
00189     return m_aux;
```

```
00190 }
00191 //---------------------------------
00192 Matrix& Matrix::operator * (double d){
00193     Matrix &m_aux=zeros(this->n_row, this->n_column);
00194     for(int i = 1; i <= this->n_row; i++) {
00195         for(int j = 1; j <= this->n_column; j++) {
00196             (m_aux)(i,j) =(*this)(i,j)* d;
00197         }
00198     }
00199     return m_aux;
00200 }
00201 //---------------------------------
00202 Matrix& Matrix::operator / (double d){
00203     Matrix &m_aux=zeros(this->n_row, this->n_column);
00204     for(int i = 1; i <= this->n_row; i++) {
00205         for(int j = 1; j <= this->n_column; j++) {
00206             (m_aux)(i,j) =(*this)(i,j)/ d;
00207         }
00208     }
00209     return m_aux;
00210 }
00211 //---------------------------------
00212 ostream& operator « (ostream &o, Matrix &m) {
00213     for (int i = 1; i <= m.n_row; i++) {
00214         for (int j = 1; j <= m.n_column; j++)
00215             printf("%5.2lf ", m(i,j));
00216         o « "\n";
00217     }
00218
00219     return o;
00220 }
00221 //---------------------------------
00222 Matrix& zeros(const int n_row, const int n_column) {
00223     Matrix *m_aux = new Matrix(n_row, n_column);
00224
00225     for(int i = 1; i <= n_row; i++) {
00226         for(int j = 1; j <= n_column; j++) {
00227             (*m_aux)(i,j) = 0;
00228         }
00229     }
00230     return (*m_aux);
00231 }
00232 //---------------------------------
00233 Matrix& eye(const int size){
00234     Matrix *m_aux = new Matrix(size,size);
00235
00236     for(int i = 1; i <= size; i++) {
00237         for(int j = 1; j <= size; j++) {
00238             (*m_aux)(i,j) = 0;
00239         }
00240         (*m_aux)(i,i) = 1;
00241     }
00242
00243     return (*m_aux);
00244 }
00245 //---------------------------------
00246 Matrix& transpose(Matrix &m) {
00247     Matrix *m_aux = new Matrix(m.n_column,m.n_row);
00248
00249     for(int i = 1; i <= m.n_row; i++) {
00250         for(int j = 1; j <= m.n_column; j++) {
00251             (*m_aux)(j,i) = m(i,j);
00252         }
00253     }
00254
00255     return (*m_aux);
00256 }
00257 //---------------------------------
00258 void swap_row(Matrix &m,int i,int index){
00259     double aux;
00260     for(int k=1;k<=m.n_column;k++){
00261         aux=m(i,k);
00262         m(i,k)=m(index,k);
00263         m(index,k)=aux;
00264     }
00265 }
00266 //---------------------------------
00267 Matrix& inv(Matrix &m) {
00268     if (m.n_column != m.n_row) {
00269         cout « "Matrix sub: error in m.n_column, m.n_row\n";
00270         exit(EXIT_FAILURE);
00271     }
00272     Matrix *m_aux=new Matrix(m.n_row,m.n_column);
00273     Matrix *m_aux1=new Matrix(m.n_row,m.n_column);
00274     double ratio,aux;
00275     *m_aux=m;
00276     *m_aux1=eye(m.n_row);
```

```
00277     int index;
00278     for(int i=1;i<=m.n_column;i++){
00279         index=i;
00280         aux=fabs((*m_aux)(i,i));
00281         for(int j=i+1;j<=m.n_column;j++){
00282             if(aux<fabs((*m_aux)(j,i))){
00283                 aux=fabs((*m_aux)(j,i));
00284                 index=j;
00285             }
00286         }
00287         swap_row(*m_aux,i,index);
00288         swap_row(*m_aux1,i,index);
00289         if((*m_aux)(i,i)==0){
00290             cout « "Error singular Matrix\n";
00291             exit(EXIT_FAILURE);
00292         }
00293         for(int j=i+1;j<=m.n_column;j++){
00294             if((*m_aux)(j,i)!=0){
00295                 ratio=(*m_aux)(j,i)/(*m_aux)(i,i);
00296                 if(ratio!=0){
00297                     for(int k=1;k<m.n_column+1;k++){
00298                         (*m_aux)(j,k)-=ratio*(*m_aux)(i,k);
00299                         (*m_aux1)(j,k)-=ratio*(*m_aux1)(i,k);
00300                     }
00301                 }
00302             }
00303         }
00304     }
00305     for(int i=m.n_row;i>=1;i--){
00306         ratio=1/(*m_aux)(i,i);
00307             for(int k=1;k<=m.n_column;k++){
00308                         (*m_aux)(i,k)*=ratio;
00309                         (*m_aux1)(i,k)*=ratio;
00310             }
00311             for(int j=i-1;j>=1;j--){
00312                 ratio=(*m_aux)(j,i);
00313                 for(int k=1;k<=m.n_column;k++){
00314                         (*m_aux)(j,k)-=ratio*(*m_aux)(i,k);
00315                         (*m_aux1)(j,k)-=ratio*(*m_aux1)(i,k);
00316                 }
00317             }
00318         }
00319         free(m_aux);
00320     return *m_aux1;
00321 }
00322 //---------------------------------
00323 Matrix& zeros(const int n) {
00324     Matrix *m_aux = new Matrix(n);
00325
00326     for(int i = 1; i <= n; i++) {
00327             (*m_aux)(1,i) = 0;
00328     }
00329
00330     return (*m_aux);
00331 }
00332 //---------------------------------
00333 double norm(Matrix &m) {
00334     double r=0;
00335
00336     for(int i = 1; i <= m.n_column*m.n_row; i++) {
00337             r+=m(i)*m(i);
00338     }
00339     return sqrt(r);
00340 }
00341 //---------------------------------
00342 double dot(Matrix &v,Matrix &w){
00343     if(v.n_column!=w.n_column){
00344         cout « "Vector dot: error in v.n_column, w.n_column\n";
00345         exit(EXIT_FAILURE);}
00346     double result=0;
00347     for(int i=1;i<=v.n_column*v.n_row;i++)
00348         result += v(i)*w(i);
00349     return result;
00350 }
00351
00352 //---------------------------------
00353 Matrix& cross(Matrix &v,Matrix &w){
00354     if(v.n_column!=w.n_column){
00355         cout « "Vector cross: error in v.n_column, w.n_column\n";
00356         exit(EXIT_FAILURE);}
00357     Matrix *m_aux = new Matrix(v.n_column);
00358     (*m_aux)(1) = v(2)*w(3)-w(2)*v(3);
00359     (*m_aux)(2) = v(3)*w(1)-w(3)*v(1);
00360     (*m_aux)(3) = v(1)*w(2)-w(1)*v(2);
00361     return (*m_aux);
00362 }
00363 //---------------------------------
```

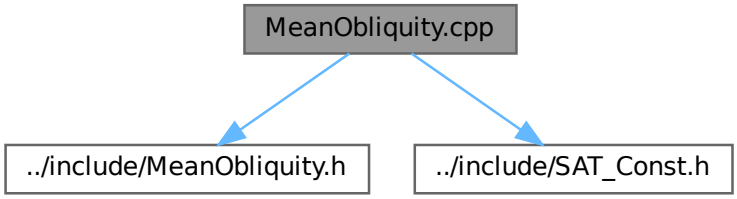
```
00364     Matrix& extract_vector(Matrix &v,int start,int end){
00365
00366     Matrix *m_aux = new Matrix(end-start+1);
00367     int x=1;
00368     for (int i=start; i<=end;i++){
00369         (*m_aux)(x)=v(i);
00370         x++;
00371         }
00372     return *m_aux;
00373     }
00374 //-----------------------------------
00375     Matrix& union_vector(Matrix &v,Matrix &w){
00376         int x=1,length=v.n_column*v.n_row+w.n_column*w.n_row;
00377         Matrix *v_aux=new Matrix(length);
00378         for(int i=1; i<=v.n_column*v.n_row;i++){
00379             (*v_aux)(x)=v(i);
00380             x++;
00381         }
00382         for(int i=1; i<=w.n_column*w.n_row;i++){
00383             (*v_aux)(x)=w(i);
00384             x++;
00385         }
00386         return (*v_aux);
00387     }
00388 //-----------------------------------
00389     Matrix& extract_row(Matrix &v,int j){
00390         if(v.n_row<j || 1>j){
00391             cout « "Matrix extract_row: error in"« v.n_row «" "«j«"\n";
00392             exit(EXIT_FAILURE);}
00393
00394             Matrix *m_aux = new Matrix(v.n_column);
00395             for (int i=1;i<=v.n_column;i++){
00396                 (*m_aux)(i)=v(j,i);
00397             }
00398             return (*m_aux);
00399     }
00400 //-----------------------------------
00401     Matrix& extract_column(Matrix &v,int j){
00402         if(v.n_column<j || j<1){
00403             cout « "Matrix extract_column: error in "« j «" "«v.n_column«"\n";
00404             exit(EXIT_FAILURE);}
00405             Matrix *m_aux = new Matrix(v.n_row);
00406             for (int i=1;i<=v.n_row;i++){
00407                 (*m_aux)(i)=v(i,j);
00408             }
00409             return (*m_aux);
00410     }
00411 //-----------------------------------
00412     Matrix& assign_row(Matrix &v,Matrix &w,int j){
00413         if(v.n_row<j || j<1 || v.n_row!=w.n_column){
00414             cout « "Matrix assign_row: error in v.n_row<j\n";
00415             exit(EXIT_FAILURE);}
00416             Matrix *m_aux=new Matrix(v.n_row,v.n_column);
00417             (*m_aux) = v;
00418             for (int i=1;i<=m_aux->n_column;i++){
00419                 (*m_aux)(j,i)=w(i);
00420             }
00421             return (*m_aux);
00422     }
00423 //-----------------------------------
00424     Matrix& assign_column(Matrix &v,Matrix &w,int j){
00425         if(v.n_column<j || j<1 || v.n_row!=w.n_column){
00426             cout « "Matrix assign_column: error in v.n_column<j\n";
00427             exit(EXIT_FAILURE);}
00428             Matrix *m_aux=new Matrix(v.n_row,v.n_column);
00429             (*m_aux) = v;
00430             for (int i=1;i<=m_aux->n_row;i++){
00431                 (*m_aux)(i,j)=w(i);
00432             }
00433             return (*m_aux);
00434
00435     }
00436 //-----------------------------------
```

5.111 MeanObliquity.cpp File Reference

El archivo contiene las implementaciones de MeanObliquity.h.

```
#include "../include/MeanObliquity.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for MeanObliquity.cpp:



Functions

- double MeanObliquity (double Mjd_TT)

5.111.1 Detailed Description

El archivo contiene las implementaciones de MeanObliquity.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file MeanObliquity.cpp.

5.111.2 Function Documentation

5.111.2.1 MeanObliquity()

```
double MeanObliquity (
            double Mjd_TT)
```

Computes the mean obliquity of the ecliptic

Parameters

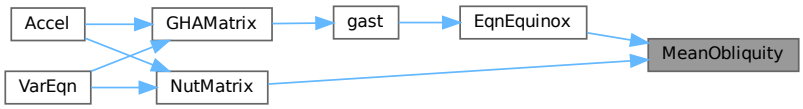
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

obliquity of the ecliptic [rad]

Definition at line 10 of file MeanObliquity.cpp.

Here is the caller graph for this function:



5.112 MeanObliquity.cpp

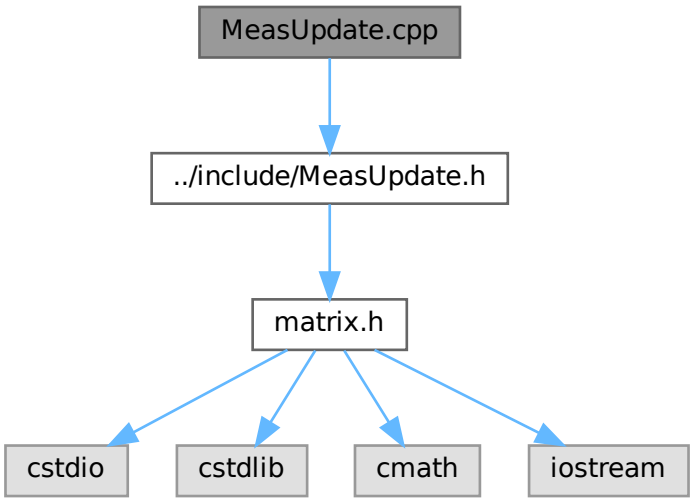
Go to the documentation of this file.

```
00001 #include "../include/MeanObliquity.h"
00002 #include "../include/SAT_Const.h"
00003
00010      double MeanObliquity (double Mjd_TT){
00011
00012
00013         double T = (Mjd_TT-MJD_J2000)/36525;
00014
00015         return Rad *( 84381.448/3600-(46.8150+(0.00059-0.001813*T)*T)*T/3600 );
00016     }
```

5.113 MeasUpdate.cpp File Reference

El archivo contiene las implementaciones de MeasUpdate.h.

`#include "../include/MeasUpdate.h"`
Include dependency graph for MeasUpdate.cpp:



Functions

- tuple< Matrix &, Matrix &, Matrix & > MeasUpdate (Matrix x, double z, double g, double s, Matrix G, Matrix P, int n)

5.113.1 Detailed Description

El archivo contiene las implementaciones de MeasUpdate.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file MeasUpdate.cpp.

5.113.2 Function Documentation

5.113.2.1 MeasUpdate()

```
tuple< Matrix &, Matrix &, Matrix & > MeasUpdate (
            Matrix x,
            double z,
            double g,
            double s,
            Matrix G,
            Matrix P,
            int n)
```

Parameters

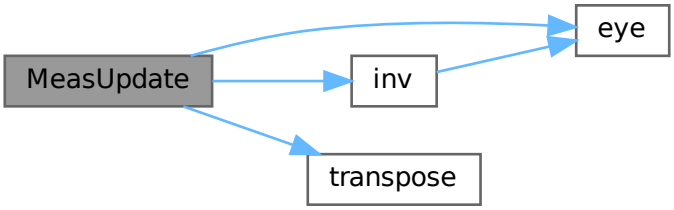
x	Matrix n∗1
z	double
g	double
s	double
G	Matrix 1∗n
P	Matrix n∗n
n	int

Returns

tupla de 3 Matrix

Definition at line 8 of file MeasUpdate.cpp.

Here is the call graph for this function:



5.114 MeasUpdate.cpp

Go to the documentation of this file.

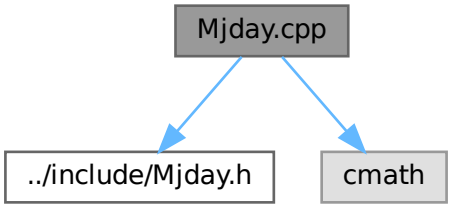
```
00001 #include "../include/MeasUpdate.h"
00008     tuple<Matrix&,Matrix&,Matrix&> MeasUpdate(Matrix x, double z,double g,double s,Matrix G,Matrix P,
      int n){
00009
00010             if(x.n_row<x.n_column){
00011                 x=transpose(x);
00012             }
00013         Matrix Inv_W(1);Inv_W(1) = s*s;
00014
00015         Matrix &K = P*transpose(G)*inv(Inv_W+G*P*transpose(G));
00016
00017         Matrix &nx = x + K*(z-g);
00018
00019         Matrix &nP = (eye(n)-K*G)*P;
00020
00021         return tie(K, nx, nP);
00022
00023     }
```

5.115 Mjday.cpp File Reference

El archivo contiene las implementaciones de Mjday.h.

```
#include "../include/Mjday.h"
#include <cmath>
```

Include dependency graph for Mjday.cpp:



Functions

- double Mjday (int yr, int mon, int day, int hr, int min, int sec)

5.115.1 Detailed Description

El archivo contiene las implementaciones de Mjday.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Mjday.cpp.

5.115.2 Function Documentation

5.115.2.1 Mjday()

```
double Mjday (
            int yr,
            int mon,
            int day,
            int hr = 0,
            int min = 0,
            int sec = 0)
```

Parameters

year	- year
mon	- month

day	- day
hr	- universal time hour
min	- universal time min
sec	- universal time sec

Returns

Modified julian date

Definition at line 9 of file Mjday.cpp.

Here is the caller graph for this function:

5.116 Mjday.cpp

Go to the documentation of this file.

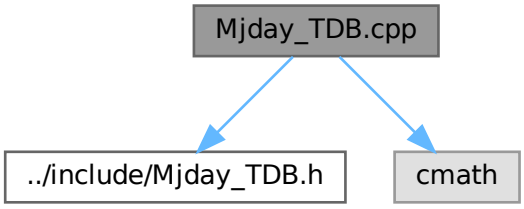
```
00001 #include "../include/Mjday.h"
00002 #include <cmath>
00009     double Mjday(int yr, int mon,int day,int hr,int min,int sec){
00010
00011 double jd = 367.0 * yr- floor( (7 * (yr + floor( (mon + 9) / 12.0) ) ) * 0.25 )+ floor( 275 * mon /
      9.0 ) + day + 1721013.5 + ( (sec/60.0 + min ) / 60.0 + hr ) / 24.0;
00012
00013 return jd-2400000.5;
00014
00015     }
```

5.117 Mjday_TDB.cpp File Reference

El archivo contiene las implementaciones de Mjday_TDB.h.

```
#include "../include/Mjday_TDB.h"
#include <cmath>
```

Include dependency graph for Mjday_TDB.cpp:



Functions

- double Mjday_TDB (double Mjd_TT)

5.117.1 Detailed Description

El archivo contiene las implementaciones de Mjday_TDB.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Mjday_TDB.cpp.

5.117.2 Function Documentation

5.117.2.1 Mjday_TDB()

```
double Mjday_TDB (
            double Mjd_TT)
```

Parameters

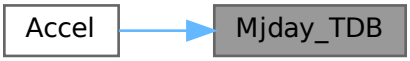
Mjd_TT	- Modified julian date (TT)

Returns

Modified julian date (TDB)

Definition at line 10 of file Mjday_TDB.cpp.

Here is the caller graph for this function:



5.118 Mjday_TDB.cpp

Go to the documentation of this file.

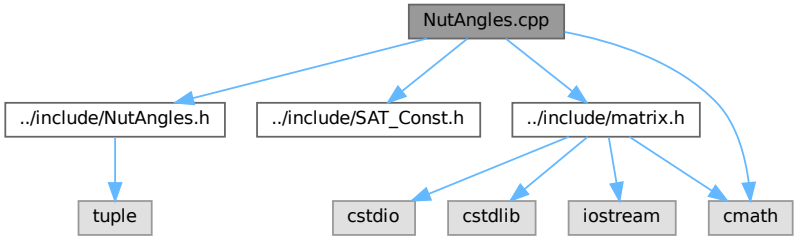
```
00001 #include "../include/Mjday_TDB.h"
00002 #include <cmath>
00003
00010     double Mjday_TDB(double Mjd_TT){
00011         double T_TT = (Mjd_TT - 51544.5)/36525;
00012
00013 return Mjd_TT + ( 0.001658*sin(628.3076*T_TT + 6.2401)+ 0.000022*sin(575.3385*T_TT+4.2970)+
      0.000014*sin(1256.6152*T_TT + 6.1969)+
00014     0.000005*sin(606.9777*T_TT+4.0212)+ 0.000005*sin(52.9691*T_TT+0.4444)+
      0.000002*sin(21.3299*T_TT+5.5431) +   0.000010*sin(628.3076*T_TT+4.2490) )/86400;
00015     }
```

5.119 NutAngles.cpp File Reference

El archivo contiene las implementaciones de NutAngles.h.

```
#include "../include/NutAngles.h"
#include "../include/SAT_Const.h"
#include "../include/matrix.h"
#include <cmath>
```

Include dependency graph for NutAngles.cpp:



Functions

- tuple< double, double > NutAngles (double Mjd_TT)

5.119.1 Detailed Description

El archivo contiene las implementaciones de NutAngles.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file NutAngles.cpp.

5.119.2 Function Documentation

5.119.2.1 NutAngles()

```
tuple< double, double > NutAngles (
            double Mjd_TT)
```

Nutation in longitude and obliquity

Parameters

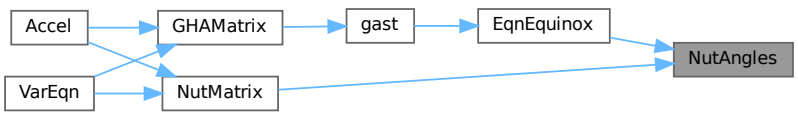
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

tupla < dpsi,deps> Nutation Angles

Definition at line 11 of file NutAngles.cpp.

Here is the caller graph for this function:



5.120 NutAngles.cpp

Go to the documentation of this file.

```
00001 #include "../include/NutAngles.h"
00002 #include "../include/SAT_Const.h"
00003 #include "../include/matrix.h"
00004 #include <cmath>
00011     tuple<double,double> NutAngles (double Mjd_TT){
00012         double T  = (Mjd_TT-MJD_J2000)/36525;
00013         double T2 = T*T;
00014         double T3 = T2*T;
00015         double rev = 360*3600; // arcsec/revolution
00016 
00017         int N_coeff = 106;
00018         Matrix C(106,9);
00019         double values[106][9]={
00020          // l  l' F  D Om    dpsi     *T     deps     *T
00021 
00022 {       0, 0, 0, 0, 1,-1719960,-1742,   920250,    89},    //   1
00023 {       0, 0, 0, 0, 2,    20620,     2,    -8950,     5},    //   2
00024 {      -2, 0, 2, 0, 1,      460,     0,     -240,     0},    //   3
00025 {       2, 0,-2, 0, 0,      110,     0,        0,     0},    //   4
00026 {      -2, 0, 2, 0, 2,      -30,     0,       10,     0},    //   5
00027 {       1,-1, 0,-1, 0,      -30,     0,        0,     0},    //   6
00028 {       0,-2, 2,-2, 1,      -20,     0,       10,     0},    //   7
00029 {       2, 0,-2, 0, 1,       10,     0,        0,     0},    //   8
00030 {       0, 0, 2,-2, 2, -131870,   -16,    57360,   -31},    //   9
00031 {       0, 1, 0, 0, 0,    14260,   -34,      540,    -1},    //  10
00032 {       0, 1, 2,-2, 2,    -5170,    12,     2240,    -6},    //  11
00033 {       0,-1, 2,-2, 2,     2170,    -5,     -950,     3},    //  12
00034 {       0, 0, 2,-2, 1,     1290,     1,     -700,     0},    //  13
00035 {       2, 0, 0,-2, 0,      480,     0,       10,     0},    //  14
00036 {       0, 0, 2,-2, 0,     -220,     0,        0,     0},    //  15
00037 {       0, 2, 0, 0, 0,      170,    -1,        0,     0},    //  16
00038 {       0, 1, 0, 0, 1,     -150,     0,       90,     0},    //  17
00039 {       0, 2, 2,-2, 2,     -160,     1,       70,     0},    //  18
00040 {       0,-1, 0, 0, 1,     -120,     0,       60,     0},    //  19
00041 {      -2, 0, 0, 2, 1,      -60,     0,       30,     0},    //  20
00042 {       0,-1, 2,-2, 1,      -50,     0,       30,     0},    //  21
00043 {       2, 0, 0,-2, 1,       40,     0,      -20,     0},    //  22
00044 {       0, 1, 2,-2, 1,       40,     0,      -20,     0},    //  23
00045 {       1, 0, 0,-1, 0,      -40,     0,        0,     0},    //  24
00046 {       2, 1, 0,-2, 0,       10,     0,        0,     0},    //  25
00047 {       0, 0,-2, 2, 1,       10,     0,        0,     0},    //  26
00048 {       0, 1,-2, 2, 0,      -10,     0,        0,     0},    //  27
00049 {       0, 1, 0, 0, 2,       10,     0,        0,     0},    //  28
00050 {      -1, 0, 0, 1, 1,       10,     0,        0,     0},    //  29
00051 {       0, 1, 2,-2, 0,      -10,     0,        0,     0},    //  30
00052 {       0, 0, 2, 0, 2,   -22740,    -2,     9770,    -5},    //  31
00053 {       1, 0, 0, 0, 0,     7120,     1,      -70,     0},    //  32
00054 {       0, 0, 2, 0, 1,    -3860,    -4,     2000,     0},    //  33
00055 {       1, 0, 2, 0, 2,    -3010,     0,     1290,    -1},    //  34
00056 {       1, 0, 0,-2, 0,    -1580,     0,      -10,     0},    //  35
00057 {      -1, 0, 2, 0, 2,     1230,     0,     -530,     0},    //  36
00058 {       0, 0, 0, 2, 0,      630,     0,      -20,     0},    //  37
00059 {       1, 0, 0, 0, 1,      630,     1,     -330,     0},    //  38
00060 {      -1, 0, 0, 0, 1,     -580,    -1,      320,     0},    //  39
00061 {      -1, 0, 2, 2, 2,     -590,     0,      260,     0},    //  40
00062 {       1, 0, 2, 0, 1,     -510,     0,      270,     0},    //  41
00063 {       0, 0, 2, 2, 2,     -380,     0,      160,     0},    //  42
00064 {       2, 0, 0, 0, 0,      290,     0,      -10,     0},    //  43
00065 {       1, 0, 2,-2, 2,      290,     0,     -120,     0},    //  44
00066 {       2, 0, 2, 0, 2,     -310,     0,      130,     0},    //  45
00067 {       0, 0, 2, 0, 0,      260,     0,      -10,     0},    //  46
00068 {      -1, 0, 2, 0, 1,      210,     0,     -100,     0},    //  47
00069 {      -1, 0, 0, 2, 1,      160,     0,      -80,     0},    //  48
00070 {       1, 0, 0,-2, 1,     -130,     0,       70,     0},    //  49
00071 {      -1, 0, 2, 2, 1,     -100,     0,       50,     0},    //  50
00072 {       1, 1, 0,-2, 0,      -70,     0,        0,     0},    //  51
00073 {       0, 1, 2, 0, 2,       70,     0,      -30,     0},    //  52
00074 {       0,-1, 2, 0, 2,      -70,     0,       30,     0},    //  53
00075 {       1, 0, 2, 2, 2,      -80,     0,       30,     0},    //  54
00076 {       1, 0, 0, 2, 0,       60,     0,        0,     0},    //  55
00077 {       2, 0, 2,-2, 2,       60,     0,      -30,     0},    //  56
00078 {       0, 0, 0, 2, 1,      -60,     0,       30,     0},    //  57
00079 {       0, 0, 2, 2, 1,      -70,     0,       30,     0},    //  58
00080 {       1, 0, 2,-2, 1,       60,     0,      -30,     0},    //  59
00081 {       0, 0, 0,-2, 1,      -50,     0,       30,     0},    //  60
00082 {       1,-1, 0, 0, 0,       50,     0,        0,     0},    //  61
00083 {       2, 0, 2, 0, 1,      -50,     0,       30,     0},    //  62
00084 {       0, 1, 0,-2, 0,      -40,     0,        0,     0},    //  63
00085 {       1, 0,-2, 0, 0,       40,     0,        0,     0},    //  64
00086 {       0, 0, 0, 1, 0,      -40,     0,        0,     0},    //  65
00087 {       1, 1, 0, 0, 0,      -30,     0,        0,     0},    //  66
00088 {       1, 0, 2, 0, 0,       30,     0,        0,     0},    //  67
```

```
00089 {       1,-1, 2, 0, 2,     -30,    0,       10,    0},   //  68
00090 {      -1,-1, 2, 2, 2,     -30,    0,       10,    0},   //  69
00091 {      -2, 0, 0, 0, 1,     -20,    0,       10,    0},   //  70
00092 {       3, 0, 2, 0, 2,     -30,    0,       10,    0},   //  71
00093 {       0,-1, 2, 2, 2,     -30,    0,       10,    0},   //  72
00094 {       1, 1, 2, 0, 2,      20,    0,      -10,    0},   //  73
00095 {      -1, 0, 2,-2, 1,     -20,    0,       10,    0},   //  74
00096 {       2, 0, 0, 0, 1,      20,    0,      -10,    0},   //  75
00097 {       1, 0, 0, 0, 2,     -20,    0,       10,    0},   //  76
00098 {       3, 0, 0, 0, 0,      20,    0,        0,    0},   //  77
00099 {       0, 0, 2, 1, 2,      20,    0,      -10,    0},   //  78
00100 {      -1, 0, 0, 0, 2,      10,    0,      -10,    0},   //  79
00101 {       1, 0, 0,-4, 0,     -10,    0,        0,    0},   //  80
00102 {      -2, 0, 2, 2, 2,      10,    0,      -10,    0},   //  81
00103 {      -1, 0, 2, 4, 2,     -20,    0,       10,    0},   //  82
00104 {       2, 0, 0,-4, 0,     -10,    0,        0,    0},   //  83
00105 {       1, 1, 2,-2, 2,      10,    0,      -10,    0},   //  84
00106 {       1, 0, 2, 2, 1,     -10,    0,       10,    0},   //  85
00107 {      -2, 0, 2, 4, 2,     -10,    0,       10,    0},   //  86
00108 {      -1, 0, 4, 0, 2,      10,    0,        0,    0},   //  87
00109 {       1,-1, 0,-2, 0,      10,    0,        0,    0},   //  88
00110 {       2, 0, 2,-2, 1,      10,    0,      -10,    0},   //  89
00111 {       2, 0, 2, 2, 2,     -10,    0,        0,    0},   //  90
00112 {       1, 0, 0, 2, 1,     -10,    0,        0,    0},   //  91
00113 {       0, 0, 4,-2, 2,      10,    0,        0,    0},   //  92
00114 {       3, 0, 2,-2, 2,      10,    0,        0,    0},   //  93
00115 {       1, 0, 2,-2, 0,     -10,    0,        0,    0},   //  94
00116 {       0, 1, 2, 0, 1,      10,    0,        0,    0},   //  95
00117 {      -1,-1, 0, 2, 1,      10,    0,        0,    0},   //  96
00118 {       0, 0,-2, 0, 1,     -10,    0,        0,    0},   //  97
00119 {       0, 0, 2,-1, 2,     -10,    0,        0,    0},   //  98
00120 {       0, 1, 0, 2, 0,     -10,    0,        0,    0},   //  99
00121 {       1, 0,-2,-2, 0,     -10,    0,        0,    0},   // 100
00122 {       0,-1, 2, 0, 1,     -10,    0,        0,    0},   // 101
00123 {       1, 1, 0,-2, 1,     -10,    0,        0,    0},   // 102
00124 {       1, 0,-2, 2, 0,     -10,    0,        0,    0},   // 103
00125 {       2, 0, 0, 2, 0,      10,    0,        0,    0},   // 104
00126 {       0, 0, 2, 4, 2,     -10,    0,        0,    0},   // 105
00127 {       0, 1, 0, 1, 0,      10,    0,        0,    0}   // 106
00128         };
00129         for (int i = 0; i < C.n_row; i++) {
00130            for (int j = 0; j < C.n_column; j++) {
00131                C.data[i][j] = values[i][j];
00132            }
00133         }
00134
00135         // Mean arguments of luni-solar motion
00136
00137         //   l   mean anomaly of the Moon
00138         //   l'  mean anomaly of the Sun
00139         //   F   mean argument of latitude
00140         //   D   mean longitude elongation of the Moon from the Sun
00141         //   Om  mean longitude of the ascending node
00142
00143         double l  = fmod (  485866.733 + (1325.0*rev +  715922.633)*T + 31.310*T2 + 0.064*T3, rev );
00144         double lp = fmod ( 1287099.804 + (  99.0*rev + 1292581.224)*T -  0.577*T2 - 0.012*T3, rev );
00145         double F  = fmod (  335778.877 + (1342.0*rev +  295263.137)*T - 13.257*T2 + 0.011*T3, rev );
00146         double D  = fmod ( 1072261.307 + (1236.0*rev + 1105601.328)*T -  6.891*T2 + 0.019*T3, rev );
00147         double Om = fmod (  450160.280 - (   5.0*rev +  482890.539)*T +  7.455*T2 + 0.008*T3, rev );
00148
00149         // Nutation in longitude and obliquity [rad]
00150
00151         double dpsi = 0;
00152         double deps = 0;
00153         double arg;
00154         for (int i=1;i<=N_coeff;i++){
00155           arg  =  ( C(i,1)*l+C(i,2)*lp+C(i,3)*F+C(i,4)*D+C(i,5)*Om )/Arcs;
00156           dpsi = dpsi + ( C(i,6)+C(i,7)*T ) * sin(arg);
00157           deps = deps + ( C(i,8)+C(i,9)*T ) * cos(arg);
00158         }
00159         dpsi = 1.0e-5 * dpsi/Arcs;
00160         deps = 1.0e-5 * deps/Arcs;
00161         return tie(dpsi,deps);
00162     }
```

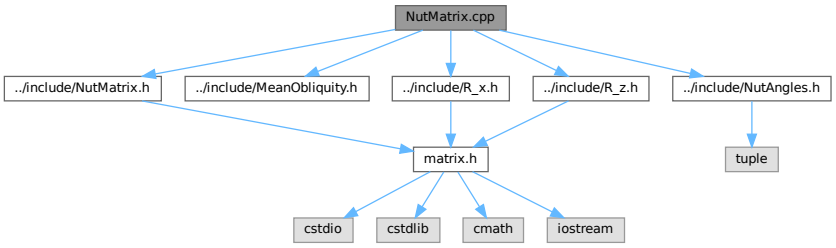
5.121 NutMatrix.cpp File Reference

El archivo contiene las implementaciones de NutMatrix.h.

```
#include "../include/NutMatrix.h"
#include "../include/MeanObliquity.h"
```

```
#include "../include/NutAngles.h"
#include "../include/R_x.h"
#include "../include/R_z.h"
```

Include dependency graph for NutMatrix.cpp:



Functions

- Matrix & NutMatrix (double Mjd_TT)

5.121.1 Detailed Description

El archivo contiene las implementaciones de NutMatrix.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file NutMatrix.cpp.

5.121.2 Function Documentation

5.121.2.1 NutMatrix()

```
Matrix & NutMatrix (
            double Mjd_TT)
```

Transformation from mean to true equator and equinox

Parameters

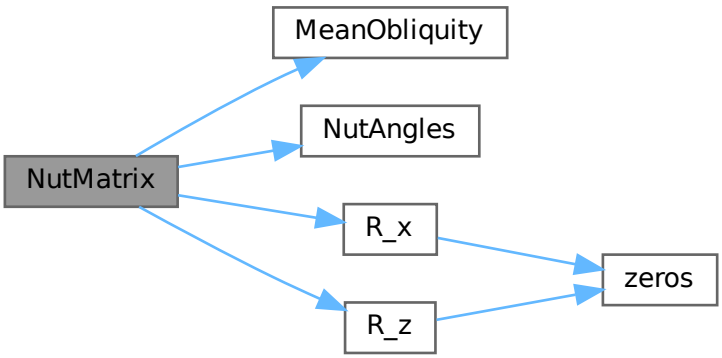
Mjd_TT	Modified Julian Date (Terrestrial Time)

Returns

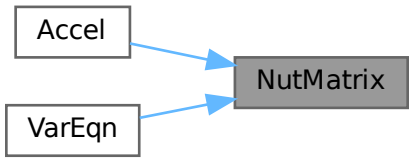
NutMat Nutation matrix

Definition at line 13 of file NutMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.122 NutMatrix.cpp

Go to the documentation of this file.

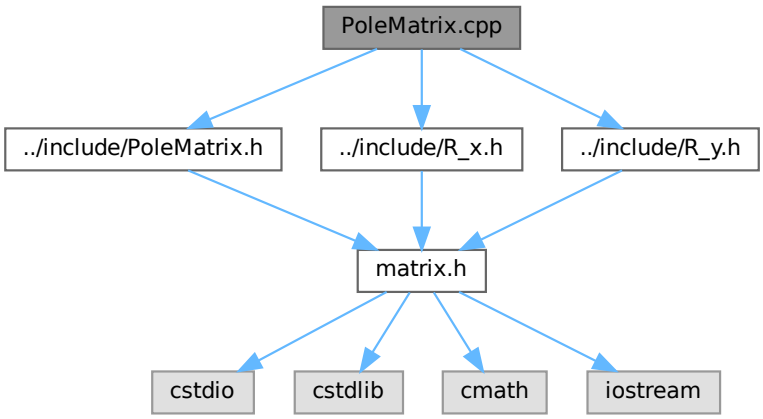
```
00001 #include "../include/NutMatrix.h"
00002 #include "../include/MeanObliquity.h"
00003 #include "../include/NutAngles.h"
00004 #include "../include/R_x.h"
00005 #include "../include/R_z.h"
00006
00013     Matrix& NutMatrix (double Mjd_TT){
00014
00015 double eps = MeanObliquity (Mjd_TT);
00016
00017 auto [dpsi, deps] = NutAngles (Mjd_TT);
00018
00019 return R_x(-eps-deps)*R_z(-dpsi)*R_x(+eps);
00020 }
```

5.123 PoleMatrix.cpp File Reference

El archivo contiene las implementaciones de PoleMatrix.h.

```
#include "../include/PoleMatrix.h"
#include "../include/R_x.h"
#include "../include/R_y.h"
```

Include dependency graph for PoleMatrix.cpp:



Functions

- Matrix & PoleMatrix (double xp, double yp)

5.123.1 Detailed Description

El archivo contiene las implementaciones de PoleMatrix.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file PoleMatrix.cpp.

5.123.2 Function Documentation

5.123.2.1 PoleMatrix()

```
Matrix & PoleMatrix (
            double xp,
            double yp)
```

Transformation from pseudo Earth-fixed to Earth-fixed coordinates for a given date

Parameters

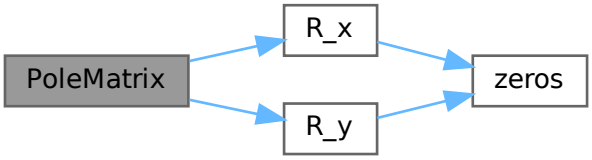
Pole	coordinte(xp,yp)

Returns

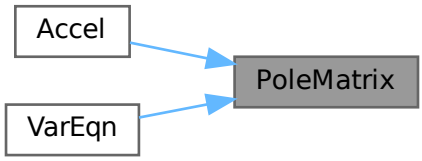
PoleMat Pole matrix

Definition at line 11 of file PoleMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.124 PoleMatrix.cpp

Go to the documentation of this file.

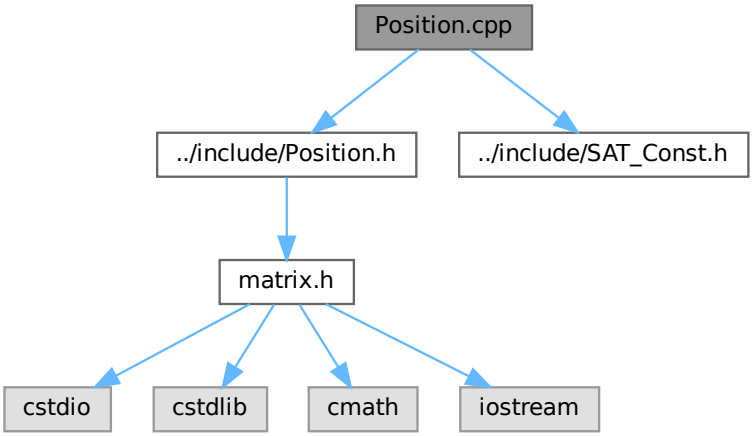
```
00001 #include "../include/PoleMatrix.h"
00002 #include "../include/R_x.h"
00003 #include "../include/R_y.h"
00004
00011     Matrix& PoleMatrix (double xp,double yp){
00012 return R_y(-xp) * R_x(-yp);
00013 }
```

5.125 Position.cpp File Reference

El archivo contiene las implementaciones de Position.h.

```
#include "../include/Position.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for Position.cpp:



Functions

- Matrix & Position (double lon, double lat, double h)

5.125.1 Detailed Description

El archivo contiene las implementaciones de Position.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file Position.cpp.

5.125.2 Function Documentation

5.125.2.1 Position()

```
Matrix & Position (
            double lon,
            double lat,
            double h)
```

Parameters

lon	longitude [rad]
lat	latitude [rad]
h	altitude [m]

Returns

Position vector (r [m]) from geodetic coordinates (Longitude [rad],latitude [rad], altitude [m])

Definition at line 10 of file Position.cpp.

5.126 Position.cpp

Go to the documentation of this file.

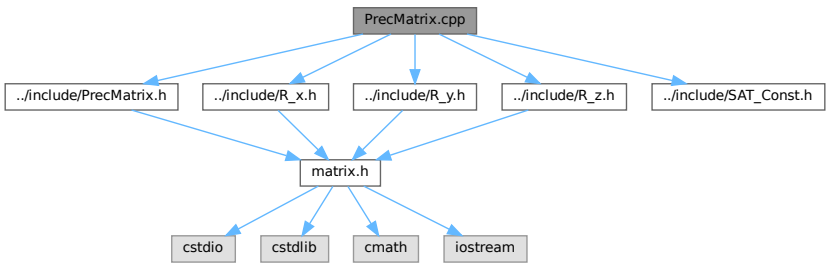
```
00001 #include "../include/Position.h"
00002 #include "../include/SAT_Const.h"
00003
00010     Matrix& Position(double lon,double lat,double h){
00011         double R_equ = R_Earth;
00012         double f     = f_Earth;
00013
00014         double e2     = f*(2.0-f);
00015         double CosLat = cos(lat);
00016         double SinLat = sin(lat);
00017
00018         double N = R_equ / sqrt(1.0-e2*SinLat*SinLat);
00019
00020         Matrix *r=new Matrix(3);
00021
00022         (*r)(1) =  (          N+h)*CosLat*cos(lon);
00023         (*r)(2) =  (          N+h)*CosLat*sin(lon);
00024         (*r)(3) =  ((1.0-e2)*N+h)*SinLat;
00025
00026         return *r;
00027     }
```

5.127 PrecMatrix.cpp File Reference

El archivo contiene las implementaciones de PrecMatrix.h.

```
#include "../include/PrecMatrix.h"
#include "../include/R_x.h"
#include "../include/R_y.h"
#include "../include/R_z.h"
#include "../include/SAT_Const.h"
```

Include dependency graph for PrecMatrix.cpp:



Functions

- Matrix & PrecMatrix (double Mjd_1, double Mjd_2)

5.127.1 Detailed Description

El archivo contiene las implementaciones de PrecMatrix.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file PrecMatrix.cpp.

5.127.2 Function Documentation

5.127.2.1 PrecMatrix()

```
Matrix & PrecMatrix (
            double Mjd_1,
            double Mjd_2)
```

Precession transformation of equatorial coordinates

Parameters

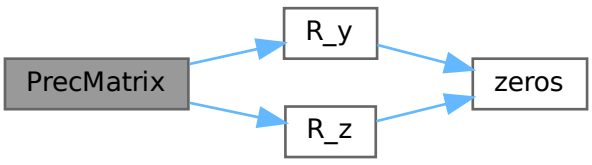
Mjd_1	Epoch given (Modified Julian Date TT)
MjD_2	Epoch to precess to (Modified Julian Date TT)

Returns

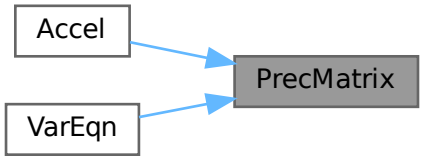
PrecMat Precession transformation matrix

Definition at line 13 of file PrecMatrix.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.128 PrecMatrix.cpp

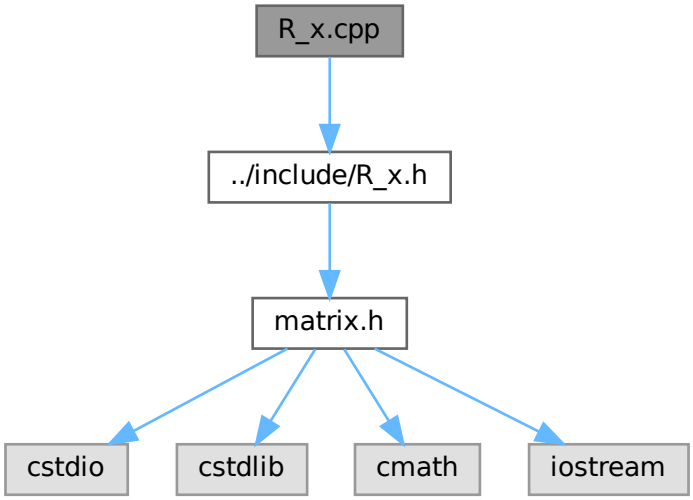
Go to the documentation of this file.

```
00001 #include "../include/PrecMatrix.h"
00002 #include "../include/R_x.h"
00003 #include "../include/R_y.h"
00004 #include "../include/R_z.h"
00005 #include "../include/SAT_Const.h"
00006
00013     Matrix& PrecMatrix (double Mjd_1, double Mjd_2){
00014         double zeta,z,theta,dT,T;
00015         T  = (Mjd_1-MJD_J2000)/36525;
00016         dT = (Mjd_2-Mjd_1)/36525;
00017
00018
00019         zeta  =  ( (2306.2181+(1.39656-0.000139*T)*T)+ ((0.30188-0.000344*T)+0.017998*dT)*dT
      )*dT/Arcs;
00020         z     =  zeta + ( (0.79280+0.000411*T)+0.000205*dT)*dT*dT/Arcs;
00021         theta =  ( (2004.3109-(0.85330+0.000217*T)*T)-((0.42665+0.000217*T)+0.041833*dT)*dT )*dT/Arcs;
00022
00023         return R_z(-z) * R_y(theta) * R_z(-zeta);
00024 }
```

5.129 R_x.cpp File Reference

El archivo contiene las implementaciones de R_x.h.

```
#include "../include/R_x.h"
```
Include dependency graph for R_x.cpp:



Functions

- Matrix & R_x (double angle)

5.129.1 Detailed Description

El archivo contiene las implementaciones de R_x.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_x.cpp.

5.129.2 Function Documentation

5.129.2.1 R_x()

```
Matrix & R_x (
            double angle)
```

Parameters

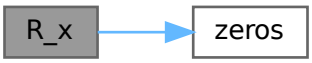
angle	angulo de rotacion

Returns

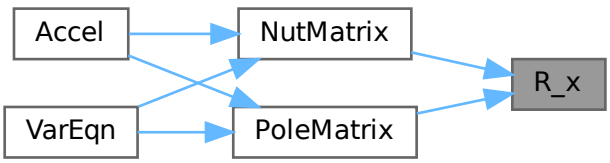
Matrix resultado

Definition at line 9 of file R_x.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.130 R_x.cpp

Go to the documentation of this file.

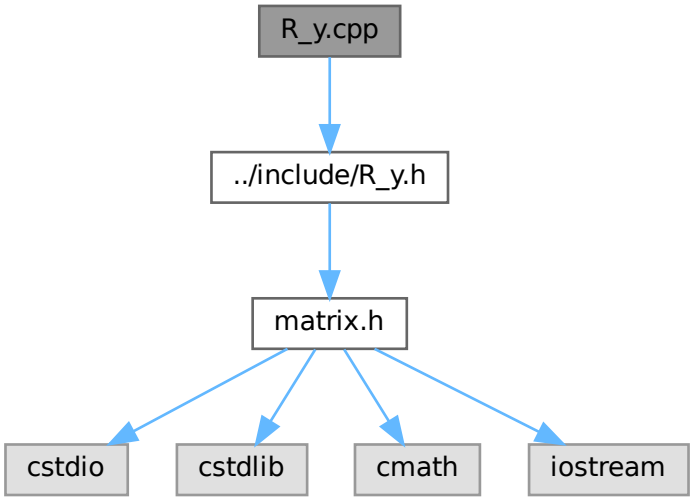
```
00001 #include "../include/R_x.h"
00002
00009     Matrix& R_x(double angle){
00010         double C = cos(angle);
00011         double S = sin(angle);
00012         Matrix *rotmat=&zeros(3,3);
00013
00014         (*rotmat)(1,1) = 1.0;   (*rotmat)(1,2) =    0.0;   (*rotmat)(1,3) = 0.0;
00015         (*rotmat)(2,1) = 0.0;   (*rotmat)(2,2) =      C;   (*rotmat)(2,3) =   S;
00016         (*rotmat)(3,1) = 0.0;   (*rotmat)(3,2) = -1.0*S;   (*rotmat)(3,3) =   C;
00017         return *rotmat;
00018     }
```

5.131 R_y.cpp File Reference

El archivo contiene las implementaciones de R_y.h.

```
#include "../include/R_y.h"
```

Include dependency graph for R_y.cpp:



Functions

- Matrix & R_y (double angle)

5.131.1 Detailed Description

El archivo contiene las implementaciones de R_y.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_y.cpp.

5.131.2 Function Documentation

5.131.2.1 R_y()

```
Matrix & R_y (
            double angle)
```

Parameters

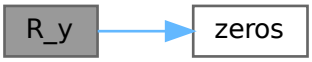
angle	angulo de rotacion

Returns

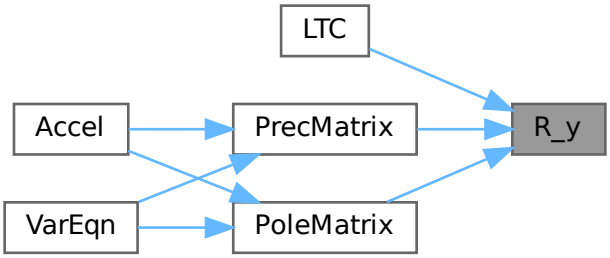
Matrix resultado

Definition at line 9 of file R_y.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.132 R_y.cpp

Go to the documentation of this file.

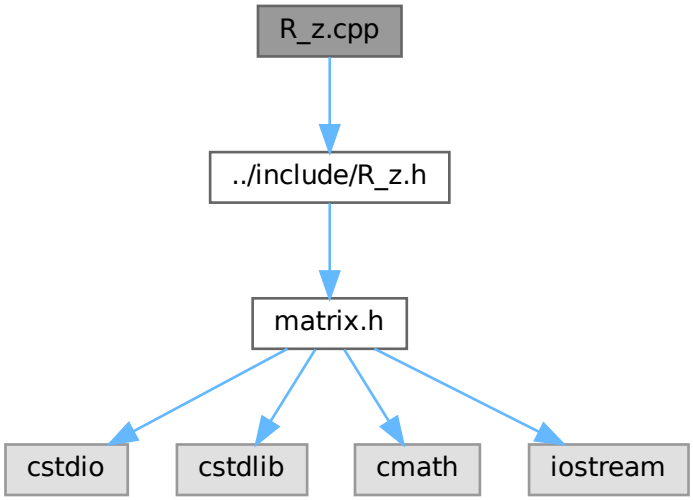
```
00001 #include "../include/R_y.h"
00002
00009     Matrix& R_y(double angle){
00010         double C = cos(angle);
00011         double S = sin(angle);
00012         Matrix *rotmat=&zeros(3,3);
00013
00014         (*rotmat)(1,1) =   C;   (*rotmat)(1,2) = 0.0;   (*rotmat)(1,3) = -1.0*S;
00015         (*rotmat)(2,1) = 0.0;   (*rotmat)(2,2) = 1.0;   (*rotmat)(2,3) =    0.0;
00016         (*rotmat)(3,1) =   S;   (*rotmat)(3,2) = 0.0;   (*rotmat)(3,3) =      C;
00017         return *rotmat;
00018     }
```

5.133 R_z.cpp File Reference

El archivo contiene las implementaciones de R_z.h.

```
#include "../include/R_z.h"
```

Include dependency graph for R_z.cpp:



Functions

- Matrix & R_z (double angle)

5.133.1 Detailed Description

El archivo contiene las implementaciones de R_z.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file R_z.cpp.

5.133.2 Function Documentation

5.133.2.1 R_z()

```
Matrix & R_z (
            double angle)
```

Parameters

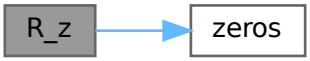
angle	angulo de rotacion

Returns

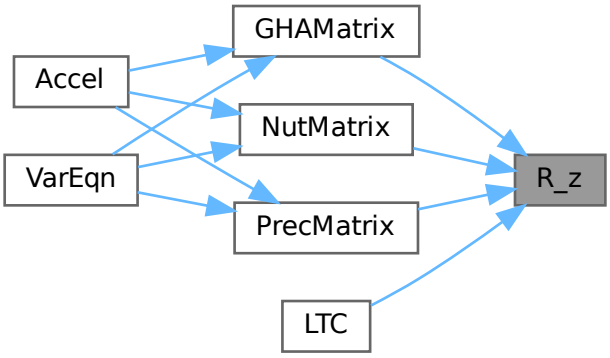
Matrix resultado

Definition at line 9 of file R_z.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



5.134 R_z.cpp

Go to the documentation of this file.

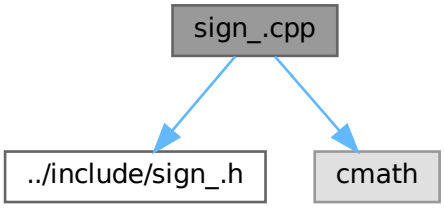
```
00001 #include "../include/R_z.h"
00002
00009     Matrix& R_z(double angle){
00010         double C = cos(angle);
00011         double S = sin(angle);
00012         Matrix *rotmat=&zeros(3,3);
00013
00014         (*rotmat)(1,1) =      C;  (*rotmat)(1,2) =   S;  (*rotmat)(1,3) = 0.0;
00015         (*rotmat)(2,1) = -1.0*S;  (*rotmat)(2,2) =   C;  (*rotmat)(2,3) = 0.0;
00016         (*rotmat)(3,1) =    0.0;  (*rotmat)(3,2) = 0.0;  (*rotmat)(3,3) = 1.0;
00017         return *rotmat;
00018     }
```

5.135 sign_.cpp File Reference

El archivo contiene las implementaciones de sign_.h.

```
#include "../include/sign_.h"
#include <cmath>
```

Include dependency graph for sign_.cpp:



Functions

- double sign_ (double a, double b)

5.135.1 Detailed Description

El archivo contiene las implementaciones de sign_.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file sign_.cpp.

5.135.2 Function Documentation

5.135.2.1 sign_()

```
double sign_ (
            double a,
            double b)
```

Parameters

a	double
b	double

Returns

absolute value of a with sign of b

Definition at line 10 of file sign_.cpp.

5.136 sign_.cpp

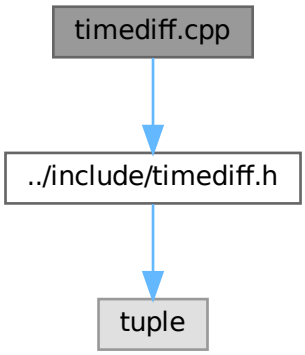
Go to the documentation of this file.

```
00001 #include "../include/sign_.h"
00002 #include <cmath>
00003
00010     double sign_(double a, double b){
00011     if (b>=0.0){
00012         return abs(a);}
00013     else{
00014         return -abs(a);}
00015     }
```

5.137 timediff.cpp File Reference

El archivo contiene las implementaciones de timediff.h.

`#include "../include/timediff.h"`
Include dependency graph for timediff.cpp:



Functions

- tuple< double, double, double, double, double > timediff (double UT1_UTC, double TAI_UTC)

5.137.1 Detailed Description

El archivo contiene las implementaciones de timediff.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file timediff.cpp.

5.137.2 Function Documentation

5.137.2.1 timediff()

```
tuple< double, double, double, double, double > timediff (
            double UT1_UTC,
            double TAI_UTC)
```

Time differences [s]

Parameters

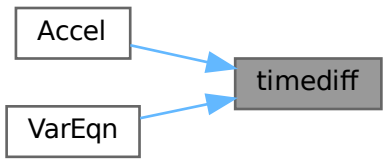
UT1_UTC	double
TAI_UTC	double

Returns

tupla <UT1_TAI, UTC_GPS, UT1_GPS, TT_UTC, GPS_UTC>

Definition at line 9 of file timediff.cpp.

Here is the caller graph for this function:



5.138 timediff.cpp

Go to the documentation of this file.

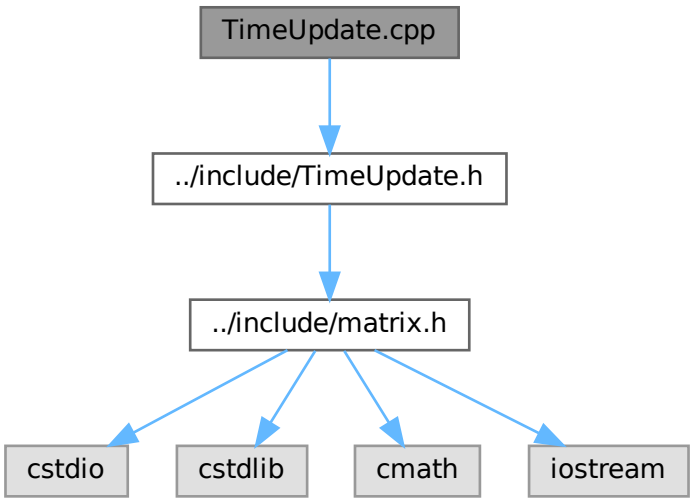
```
00001 #include "../include/timediff.h"
00002
00009     tuple<double,double,double,double,double>  timediff(double UT1_UTC,double TAI_UTC){
00010
00011         double TT_TAI  = +32.184;          // TT-TAI time difference [s]
00012
00013         double GPS_TAI = -19.0;            // GPS-TAI time difference [s]
00014 /*
00015         double TT_GPS  =  TT_TAI-GPS_TAI;  // TT-GPS time difference [s]
00016
00017         double TAI_GPS = -GPS_TAI;         // TAI-GPS time difference [s]
00018 */
00019         double UT1_TAI = UT1_UTC-TAI_UTC;  // UT1-TAI time difference [s]
00020
00021         double UTC_TAI = -TAI_UTC;         // UTC-TAI time difference [s]
00022
00023         double UTC_GPS = UTC_TAI-GPS_TAI;  // UTC_GPS time difference [s]
00024
00025         double UT1_GPS = UT1_TAI-GPS_TAI;  // UT1-GPS time difference [s]
00026
00027         double TT_UTC  = TT_TAI-UTC_TAI;   //  TT-UTC time difference [s]
00028
00029         double GPS_UTC = GPS_TAI-UTC_TAI;  // GPS-UTC time difference [s]
00030
00031         return tie(UT1_TAI, UTC_GPS, UT1_GPS, TT_UTC, GPS_UTC);
00032
00033     }
```

5.139 TimeUpdate.cpp File Reference

El archivo contiene las implementaciones de TimeUpdate.h.

```
#include "../include/TimeUpdate.h"
```

Include dependency graph for TimeUpdate.cpp:



Functions

- Matrix & TimeUpdate (Matrix P, Matrix Phi, Matrix Qdt)
- Matrix & TimeUpdate (Matrix P, Matrix Phi)

5.139.1 Detailed Description

El archivo contiene las implementaciones de TimeUpdate.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file TimeUpdate.cpp.

5.139.2 Function Documentation

5.139.2.1 TimeUpdate() [1/2]

```
Matrix & TimeUpdate (
            Matrix P,
            Matrix Phi)
```

Parameters

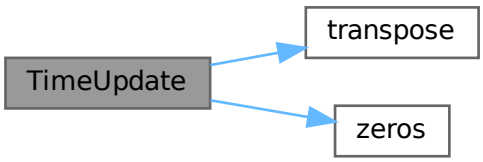
P	Matrix
Phi	Matrix

Returns

Matrix

Definition at line 15 of file TimeUpdate.cpp.

Here is the call graph for this function:



5.139.2.2 TimeUpdate() [2/2]

```
Matrix & TimeUpdate (
            Matrix P,
            Matrix Phi,
            Matrix Qdt)
```

Parameters

P	Matrix
Phi	Matrix
Qdt	Matrix

Returns

Matrix

Definition at line 9 of file TimeUpdate.cpp.

Here is the call graph for this function:

5.140 TimeUpdate.cpp

Go to the documentation of this file.

```
00001 #include "../include/TimeUpdate.h"
00002
00009     Matrix& TimeUpdate(Matrix P,Matrix Phi,Matrix Qdt){
00010
00011         return Phi*P*transpose(Phi)+ Qdt;
00012
00013     }
00014
00015     Matrix& TimeUpdate(Matrix P,Matrix Phi){
00016         Matrix Qdt=zeros(P.n_column,P.n_row);
00017         return Phi*P*transpose(Phi)+ Qdt;
00018
00019     }
```

5.141 VarEqn.cpp File Reference

El archivo contiene las implementaciones de VarEqn.h.

```
#include "../include/VarEqn.h"
#include "../include/IERS.h"
#include "../include/timediff.h"
#include "../include/GLOBAL.h"
#include "../include/SAT_Const.h"
#include "../include/PrecMatrix.h"
#include "../include/NutMatrix.h"
#include "../include/PoleMatrix.h"
#include "../include/GHAMatrix.h"
#include "../include/AccelHarmonic.h"
#include "../include/G_AccelHarmonic.h"
```

Include dependency graph for VarEqn.cpp:

Functions

- Matrix & VarEqn (double x, Matrix yPhi)

5.141.1 Detailed Description

El archivo contiene las implementaciones de VarEqn.h.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file VarEqn.cpp.

5.141.2 Function Documentation

5.141.2.1 VarEqn()

```
Matrix & VarEqn (
            double x,
            Matrix yPhi)
```

Computes the variational equations, i.e. the derivative of the state vector and the state transition matrix

Parameters

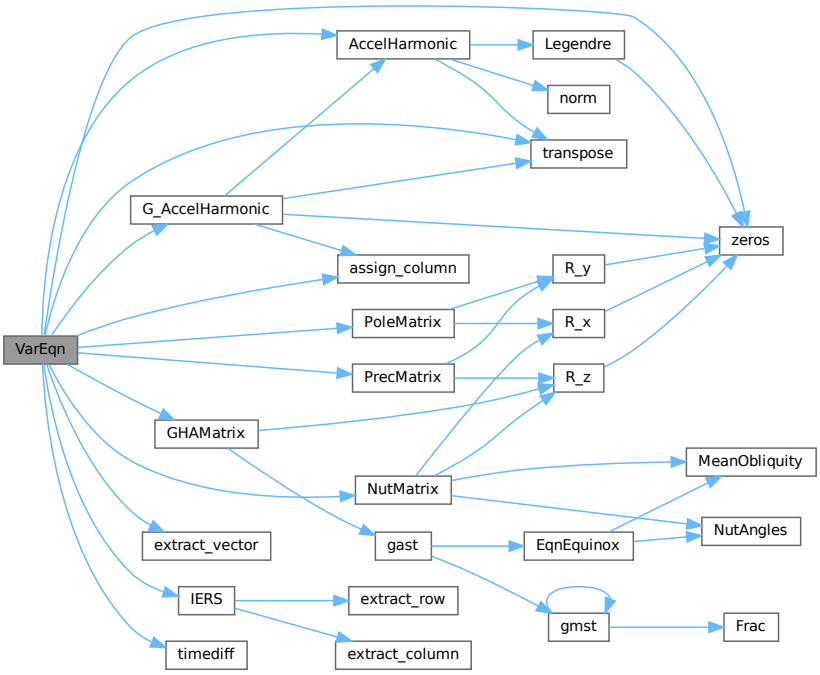
x	Time since epoch in [s]
yPhi	(6+36)-dim vector comprising the state vector (y) and the state transition matrix (Phi) in column wise storage order

Returns

yPhip Derivative of yPhi

Definition at line 18 of file VarEqn.cpp.

Here is the call graph for this function:



5.142 VarEqn.cpp

Go to the documentation of this file.

```
00001 #include "../include/VarEqn.h"
00002 #include "../include/IERS.h"
00003 #include "../include/timediff.h"
00004 #include "../include/GLOBAL.h"
00005 #include "../include/SAT_Const.h"
00006 #include "../include/PrecMatrix.h"
00007 #include "../include/NutMatrix.h"
00008 #include "../include/PoleMatrix.h"
00009 #include "../include/GHAMatrix.h"
00010 #include "../include/AccelHarmonic.h"
00011 #include "../include/G_AccelHarmonic.h"
00018 Matrix& VarEqn(double x, Matrix yPhi){
00019     if(yPhi.n_row<yPhi.n_column){
00020         yPhi=transpose(yPhi);
00021     }
00022     auto [x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC] =
      IERS(eopdata,AuxParam.Mjd_UTC,'l');
00023     auto [UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);
00024     double Mjd_UT1 = AuxParam.Mjd_TT + (UT1_UTC-TT_UTC)/86400;
00025
00026 // Transformation matrix
00027     Matrix P = PrecMatrix(MJD_J2000,AuxParam.Mjd_TT + x/86400);
00028     Matrix N = NutMatrix(AuxParam.Mjd_TT + x/86400);
00029     Matrix T = N * P;
00030     Matrix E = PoleMatrix(x_pole,y_pole) * GHAMatrix(Mjd_UT1) * T;
00031
00032 // State vector components
00033     Matrix r = extract_vector(yPhi,1,3);
00034     Matrix v = extract_vector(yPhi,4,6);
00035     Matrix Phi = zeros(6,6);
00036 // State transition matrix
00037     for (int j=1;j<=6;j++){
00038         Phi=assign_column(Phi,extract_vector(yPhi,6*j+1,6*j+6),j);
00039     }
00040
00041 // Acceleration and gradient
00042     Matrix a = AccelHarmonic ( r, E, AuxParam.n, AuxParam.m );
00043     Matrix G = G_AccelHarmonic ( r, E, AuxParam.n, AuxParam.m );
00044
00045 // Time derivative of state transition matrix
00046     Matrix &yPhip = zeros(42,1);
00047     Matrix dfdy = zeros(6,6);
00048
00049     for (int i=1;i<=3;i++){
00050         for (int j=1;j<=3;j++){
00051             dfdy(i,j) = 0.0;                 // dv/dr(i,j)
00052             dfdy(i+3,j) = G(i,j);            // da/dr(i,j)
00053             if ( i==j ){
00054                 dfdy(i,j+3) = 1;}
00055                 else{
00056                 dfdy(i,j+3) = 0;             // dv/dv(i,j)
00057             }
00058             dfdy(i+3,j+3) = 0.0;             // da/dv(i,j)
00059         }
00060     }
00061
00062     Matrix Phip = dfdy*Phi;
00063
00064 // Derivative of combined state vector and state transition matrix
00065     for (int i=1;i<=3;i++){
00066     yPhip(i)   = v(i);                  // dr/dt(i)
00067     yPhip(i+3) = a(i);                  // dv/dt(i)
00068 }
00069
00070 for (int i=1;i<=6;i++){
00071     for (int j=1;j<=6;j++){
00072         yPhip(6*j+i) = Phip(i,j);     // dPhi/dt(i,j)
00073     }
00074 }
00075 return yPhip;
00076 }
```

5.143 EKF_GEOS3.cpp File Reference

Es el archivo principal del programa.

```
#include <cmath>
#include <tuple>
#include <iostream>
#include "../include/matrix.h"
#include "../include/Accel.h"
#include "../include/PrecMatrix.h"
#include "../include/NutMatrix.h"
#include "../include/IERS.h"
#include "../include/timediff.h"
#include "../include/PoleMatrix.h"
#include "../include/AccelHarmonic.h"
#include "../include/GHAMatrix.h"
#include "../include/JPL_Eph_DE430.h"
#include "../include/GLOBAL.h"
#include "../include/AccelPointMass.h"
#include "../include/Mjday_TDB.h"
#include "../include/SAT_Const.h"
#include "../include/Position.h"
#include "../include/Mjday.h"
#include "../include/DEInteg.h"
#include "../include/TimeUpdate.h"
#include "../include/AzElPa.h"
#include "../include/R_x.h"
#include "../include/R_y.h"
#include "../include/R_z.h"
#include "../include/gmst.h"
#include "../include/VarEqn.h"
#include "../include/LTC.h"
#include "../include/MeasUpdate.h"
```

Include dependency graph for EKF_GEOS3.cpp:

Functions

- int main ()

5.143.1 Detailed Description

Es el archivo principal del programa.

Author

Pedro Zhuzhan

Bug No known bugs

Definition in file EKF_GEOS3.cpp.

5.143.2 Function Documentation

5.143.2.1 main()

```
int main ()
```

Definition at line 36 of file EKF_GEOS3.cpp.

5.144 EKF_GEOS3.cpp

Go to the documentation of this file.

```
00001 #include <cmath>
00002 #include<tuple>
00003 #include <iostream>
00004 #include"../include/matrix.h"
00005 #include "../include/Accel.h"
00006 #include "../include/PrecMatrix.h"
00007 #include "../include/NutMatrix.h"
00008 #include "../include/IERS.h"
00009 #include "../include/timediff.h"
00010 #include "../include/PoleMatrix.h"
00011 #include "../include/AccelHarmonic.h"
00012 #include "../include/GHAMatrix.h"
00013 #include "../include/JPL_Eph_DE430.h"
00014 #include "../include/GLOBAL.h"
00015 #include "../include/AccelPointMass.h"
00016 #include "../include/Mjday_TDB.h"
00017 #include "../include/SAT_Const.h"
00018 #include "../include/Position.h"
00019 #include "../include/Mjday.h"
00020 #include "../include/DEInteg.h"
00021 #include "../include/TimeUpdate.h"
00022 #include "../include/AzElPa.h"
00023 #include "../include/R_x.h"
00024 #include "../include/R_y.h"
00025 #include "../include/R_z.h"
00026 #include "../include/gmst.h"
00027 #include "../include/VarEqn.h"
00028 #include "../include/LTC.h"
00029 #include "../include/MeasUpdate.h"
00036         int main(){
00037
00038     AuxParamLoad();
00039     eop19620101();
00040     GGM03S();
00041     DE430Coeff();
00042     GEOS3();
00043
00044     int i=0,j,ii,nobs = 46;
00045             double sigma_range,sigma_az,sigma_el,lat,lon,alt,Mjd1,Mjd2,Mjd3,Mjd0,Mjd_UTC=obs(9,1),
00046             n_eqn,theta,t_old,Mjd_TT,Dist,Mjd_UT1,UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC,
00047             x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC,Azim, Elev,t;
00048
00049             Matrix Rs,Y0_apr,P,LT,yPhi,Phi,Y_true=zeros(6),Y0=zeros(6),U,Y_old,dDdY,dDds,r,s,dEdY,
00050             K, Y, dAds, dEds,dAdY;
00051             sigma_range = 92.5;
00052 sigma_az = 0.0224*Rad;
00053 sigma_el = 0.0139*Rad;
00054
00055
00056 lat = Rad*21.5748;
00057 lon = Rad*(-158.2706);
00058 alt = 300.20;
00059 Rs = transpose(Position(lon, lat, alt));
00060
00061 Mjd1 = obs(1,1);
00062 Mjd2 = obs(9,1);
00063 Mjd3 = obs(18,1);
00064 Matrix r2(3),v2(3);
00065 r2(1)=6221397.62857869;
00066 r2(2)=2867713.77965738;
00067 r2(3)=3006155.98509949;
00068
00069 v2(1)= 4645.04725161806;
00070 v2(2)=-2752.21591588204;
00071 v2(3)=-7507.99940987031;
```

```
00072 //auto [r2,v2] =
      anglesg(obs(1,2),obs(9,2),obs(18,2),obs(1,3),obs(9,3),obs(18,3),Mjd1,Mjd2,Mjd3,Rs,Rs,Rs);
00073
00074
00075
00076 Y0_apr = union_vector(r2,v2);
00077
00078 Mjd0 = Mjday(1995,1,29,02,38,0);
00079
00080 AuxParam.Mjd_UTC = Mjd_UTC;
00081
00082 Mjd_UTC = obs(9,1);
00083 n_eqn  = 6;
00084
00085 Y = DEInteg(Accel,0,-(obs(9,1)-Mjd0)*86400.0,1e-13,1e-6,6,Y0_apr);
00086 P = zeros(6,6);
00087 for (i=1;i<=3;i++){
00088     P(i,i)=1e8;}
00089 for (i=4;i<=6;i++){
00090     P(i,i)=1e3;}
00091
00092 LT = LTC(lon,lat);
00093
00094 yPhi = zeros(42,1);
00095 Phi  = zeros(6,6);
00096
00097
00098 t = 0;
00099
00100 for (i=1;i<=nobs;i++){
00101     t_old = t;
00102     Y_old = Y;
00103
00104     Mjd_UTC = obs(i,1);
00105     t       = (Mjd_UTC-Mjd0)*86400.0;
00106
00107     tie (x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC) = IERS(eopdata,Mjd_UTC,'l');
00108     tie (UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC) = timediff(UT1_UTC,TAI_UTC);
00109     Mjd_TT = Mjd_UTC + TT_UTC/86400;
00110     Mjd_UT1 = Mjd_TT + (UT1_UTC-TT_UTC)/86400.0;
00111     AuxParam.Mjd_UTC = Mjd_UTC;
00112     AuxParam.Mjd_TT = Mjd_TT;
00113     for ( ii=1;ii<=6;ii++){
00114         yPhi(ii) = Y_old(ii);
00115         for (j=1;j<=6;j++)  {
00116             if (ii==j) {
00117                 yPhi(6*j+ii) = 1;
00118             }
00119             else{
00120                 yPhi(6*j+ii) = 0;
00121
00122             }
00123         }
00124     }
00125     yPhi = DEInteg (VarEqn,0,t-t_old,1e-13,1e-6,42,yPhi);
00126
00127     for (j=1;j<=6;j++){
00128         Phi = assign_column(Phi,extract_vector(yPhi,6*j+1,6*j+6),j);
00129
00130     }
00131     Y = DEInteg (Accel,0,t-t_old,1e-13,1e-6,6,Y_old);
00132     theta = gmst(Mjd_UT1);
00133     U = R_z(theta);
00134     r = extract_vector(Y,1,3);
00135     r=transpose(r);
00136     s = LT*(U*r-Rs);
00137
00138     P = TimeUpdate(P, Phi);
00139
00140
00141     tie( Azim, Elev, dAds, dEds)= AzElPa(s);
00142     dAdY = union_vector(dAds*LT*U,zeros(1,3));
00143
00144
00145      tie( K, Y, P) = MeasUpdate ( Y, obs(i,2), Azim, sigma_az, dAdY, P, 6 );
00146
00147     r = extract_vector(Y,1,3);
00148     r=transpose(r);
00149     s = LT*(U*r-Rs);
00150     tie( Azim, Elev, dAds, dEds)= AzElPa(s);
00151
00152
00153     dEdY = union_vector(dEds*LT*U,zeros(1,3));
00154
00155
00156      tie( K, Y, P) = MeasUpdate ( Y, obs(i,3), Elev, sigma_el, dEdY, P, 6 );
00157
```

```
00158     r = extract_vector(Y,1,3);
00159     r=transpose(r);
00160     s = LT*(U*r-Rs);
00161 
00162     Dist = norm(s);
00163     dDds = transpose(s/Dist);
00164     dDdY = union_vector(dDds*LT*U,zeros(1,3));
00165 
00166 
00167      tie(K, Y,P) = MeasUpdate ( Y, obs(i,4), Dist, sigma_range, dDdY, P, 6 );
00168 }
00169 
00170 tie(x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC) = IERS(eopdata,obs(46,1),'l');
00171 tie(UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC) = timediff(UT1_UTC,TAI_UTC);
00172 Mjd_TT = Mjd_UTC + TT_UTC/86400;
00173 AuxParam.Mjd_UTC = Mjd_UTC;
00174 AuxParam.Mjd_TT = Mjd_TT;
00175 
00176 Y0 = DEInteg (Accel,0,-(obs(46,1)-obs(1,1))*86400.0,1e-13,1e-6,6,Y);
00177 
00178  double aux[]= {5753.173e3, 2673.361e3, 3440.304e3, 4.324207e3, -1.924299e3, -5.728216e3};
00179  Y_true(6);
00180  for(i=0;i<6;i++){
00181      Y_true(i+1)=aux[i];
00182  }
00183 
00184 cout«"\nError of Position Estimation\n";
00185 cout«Y0(1)-Y_true(1)«" [m]\n";
00186 cout«Y0(2)-Y_true(2)«" [m]\n";
00187 cout«Y0(3)-Y_true(3)«" [m]\n";
00188 cout«"\nError of Velocity Estimation\n";
00189 cout«Y0(4)-Y_true(4)«" [m/s]\n";
00190 cout«Y0(5)-Y_true(5)«" [m/s]\n";
00191 cout«Y0(6)-Y_true(6)«" [m/s]\n";
00192 return 0;
00193     }
```

5.145 tests.cpp

```
00001 #include "../include/matrix.h"
00002 #include "../include/R_x.h"
00003 #include "../include/R_y.h"
00004 #include "../include/R_z.h"
00005 #include "../include/AccelPointMass.h"
00006 #include "../include/Cheb3D.h"
00007 #include "../include/EccAnom.h"
00008 #include "../include/Frac.h"
00009 #include "../include/MeanObliquity.h"
00010 #include "../include/Mjday.h"
00011 #include "../include/Mjday_TDB.h"
00012 #include "../include/Position.h"
00013 #include "../include/sign_.h"
00014 #include "../include/timediff.h"
00015 #include "../include/AzElPa.h"
00016 #include "../include/IERS.h"
00017 #include "../include/Legendre.h"
00018 #include "../include/NutAngles.h"
00019 #include "../include/TimeUpdate.h"
00020 #include "../include/GLOBAL.h"
00021 #include "../include/AccelHarmonic.h"
00022 #include "../include/EqnEquinox.h"
00023 #include "../include/JPL_Eph_DE430.h"
00024 #include "../include/LTC.h"
00025 #include "../include/NutMatrix.h"
00026 #include "../include/PoleMatrix.h"
00027 #include "../include/PrecMatrix.h"
00028 #include "../include/gmst.h"
00029 #include "../include/gast.h"
00030 #include "../include/MeasUpdate.h"
00031 #include "../include/G_AccelHarmonic.h"
00032 #include "../include/GHAMatrix.h"
00033 #include "../include/Accel.h"
00034 #include "../include/VarEqn.h"
00035 #include "../include/DEInteg.h"
00036 #include <cstdio>
00037 #include <cmath>
00038 #include <tuple>
00039 
00040 using namespace std;
00041 int tests_run = 0;
00042 
00043 #define FAIL() printf("\nfailure in %s() line %d\n", __func__, __LINE__)
00044 #define _assert(test) do { if (!(test)) { FAIL(); return 1; } } while(0)
```

```
00045 #define _verify(test) do { int r=test(); tests_run++; if(r) return r; } while(0)
00046
00047 int m_equals(Matrix A, Matrix B, double p) {
00048     if (A.n_row != B.n_row || A.n_column != B.n_column)
00049         return 0;
00050     else
00051         for(int i = 1; i <= A.n_row; i++)
00052             for(int j = 1; j <= A.n_column; j++)
00053                 if(fabs(A(i,j)-B(i,j)) > p) {
00054                     printf("%2.20lf %2.20lf\n",A(i,j),B(i,j));
00055                     return 0;
00056                 }
00057
00058     return 1;
00059 }
00060 int m_sum_01() {
00061     int f = 3;
00062     int c = 4;
00063
00064     Matrix A(f, c);
00065     A(1,1) = 0; A(1,2) =  2; A(1,3) = 8; A(1,4) = 0;
00066     A(2,1) = 1; A(2,2) = -1; A(2,3) = 0; A(2,4) = 0;
00067     A(3,1) = 0; A(3,2) =  1; A(3,3) = 0; A(3,4) = 5;
00068
00069     Matrix B(f, c);
00070     B(1,1) = 2; B(1,2) =  0; B(1,3) = 0; B(1,4) = 0;
00071     B(2,1) = 7; B(2,2) = -2; B(2,3) = 1; B(2,4) = 0;
00072     B(3,1) = 0; B(3,2) = -3; B(3,3) = 0; B(3,4) = 2;
00073
00074     Matrix C(f, c);
00075     C(1,1) = 2; C(1,2) =  2; C(1,3) = 8; C(1,4) = 0;
00076     C(2,1) = 8; C(2,2) = -3; C(2,3) = 1; C(2,4) = 0;
00077     C(3,1) = 0; C(3,2) = -2; C(3,3) = 0; C(3,4) = 7;
00078
00079     Matrix R = A + B;
00080
00081     _assert(m_equals(C, R, 1e-11));
00082
00083     return 0;
00084 }
00085
00086 int m_sub_01() {
00087     int f = 3;
00088     int c = 4;
00089
00090     Matrix A(f, c);
00091     A(1,1) = 0; A(1,2) = 2; A(1,3) = 8; A(1,4) = 0;
00092     A(2,1) = 1; A(2,2) = -1; A(2,3) = 0; A(2,4) = 0;
00093     A(3,1) = 0; A(3,2) = 1; A(3,3) = 0; A(3,4) = 5;
00094
00095     Matrix B(f, c);
00096     B(1,1) = 2; B(1,2) = 0; B(1,3) = 0; B(1,4) = 0;
00097     B(2,1) = 7; B(2,2) = -2; B(2,3) = 1; B(2,4) = 0;
00098     B(3,1) = 0; B(3,2) = -3; B(3,3) = 0; B(3,4) = 2;
00099
00100     Matrix C(f, c);
00101     C(1,1) = -2; C(1,2) = 2; C(1,3) = 8; C(1,4) = 0;
00102     C(2,1) = -6; C(2,2) = 1; C(2,3) = -1; C(2,4) = 0;
00103     C(3,1) = 0; C(3,2) = 4; C(3,3) = 0; C(3,4) = 3;
00104
00105     Matrix R = A - B;
00106
00107     _assert(m_equals(C, R, 1e-11));
00108
00109     return 0;
00110 }
00111
00112 int m_mul_01() {
00113     int f = 4;
00114     int c = 4;
00115
00116     Matrix A(f, c);
00117     A(1,1) = 5; A(1,2) = 8; A(1,3) = 2; A(1,4) = 4;
00118     A(2,1) = 2; A(2,2) = 2; A(2,3) = 2; A(2,4) = 2;
00119     A(3,1) = 2; A(3,2) = 2; A(3,3) = 1; A(3,4) = 3;
00120     A(4,1) = 2; A(4,2) = 2; A(4,3) = 2; A(4,4) = 1;
00121
00122     Matrix B(f, c);
00123
00124     B(1,1) = 4; B(1,2) = 2; B(1,3) = 9; B(1,4) = 1;
00125     B(2,1) = 2; B(2,2) = 9; B(2,3) = 2; B(2,4) = 7;
00126     B(3,1) = 2; B(3,2) = 2; B(3,3) = 4; B(3,4) = 9;
00127     B(4,1) = 3; B(4,2) = 4; B(4,3) = 2; B(4,4) = 5;
00128
00129     Matrix C(f, c);
00130
00131     C(1,1) = 52 ; C(1,2) = 102  ; C(1,3) = 77   ; C(1,4) = 99   ;
```

```
00132     C(2,1) = 22 ; C(2,2) = 34   ; C(2,3) = 34   ; C(2,4) = 44   ;
00133     C(3,1) = 23 ; C(3,2) = 36   ; C(3,3) = 32   ; C(3,4) = 40   ;
00134     C(4,1) = 19 ; C(4,2) = 30   ; C(4,3) = 32   ; C(4,4) = 39   ;
00135
00136     Matrix R = A * B;
00137     _assert(m_equals(R, C, 1e-11));
00138
00139     return 0;
00140 }
00141 int m_div_01() {
00142     int f = 4;
00143     int c = 4;
00144
00145     Matrix A(f, c);
00146     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00147     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00148     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00149     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00150
00151     Matrix B(f, c);
00152
00153     B(1,1) = 4; B(1,2) = 2; B(1,3) = 9; B(1,4) = 1;
00154     B(2,1) = 2; B(2,2) = 9; B(2,3) = 2; B(2,4) = 7;
00155     B(3,1) = 2; B(3,2) = 2; B(3,3) = 4; B(3,4) = 9;
00156     B(4,1) = 3; B(4,2) = 4; B(4,3) = 2; B(4,4) = 5;
00157
00158     Matrix C(f, c);
00159
00160     C(1,1) = 337./963   ; C(1,2) = 350./963     ; C(1,3) = 677./963 ; C(1,4) = -271./321;
00161     C(2,1) = 23./963    ; C(2,2) = -179./963    ; C(2,3) = 592./963 ; C(2,4) = 112./321 ;
00162     C(3,1) = 58./963        ; C(3,2) = -577./963; C(3,3) = -475./963    ; C(3,4) = 743./321 ;
00163     C(4,1) = 430./963   ; C(4,2) = 338./963     ; C(4,3) = 98./963  ; C(4,4) = -181./321    ;
00164
00165     Matrix R = A / B;
00166     _assert(m_equals(R, C, 1e-11));
00167
00168     return 0;
00169 }
00170
00171 int m_sum_d_01() {
00172     int f = 4;
00173     int c = 4;
00174     double num=2;
00175
00176     Matrix A(f, c);
00177     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00178     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00179     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00180     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00181
00182
00183     Matrix B(f, c);
00184     B(1,1) = 1+num; B(1,2) = 2+num; B(1,3) = 5+num; B(1,4) = 5+num;
00185     B(2,1) = 2+num; B(2,2) = 1+num; B(2,3) = 3+num; B(2,4) = 6+num;
00186     B(3,1) = 5+num; B(3,2) = 3+num; B(3,3) = 2+num; B(3,4) = 3+num;
00187     B(4,1) = 1+num; B(4,2) = 2+num; B(4,3) = 4+num; B(4,4) = 1+num;
00188
00189
00190     Matrix R=A+num;
00191
00192     _assert(m_equals(B, R, 1e-11));
00193
00194     return 0;
00195 }
00196
00197 int m_sub_d_01() {
00198     int f = 4;
00199     int c = 4;
00200     double num=2;
00201
00202     Matrix A(f, c);
00203     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00204     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00205     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00206     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00207
00208
00209     Matrix B(f, c);
00210     B(1,1) = 1-num; B(1,2) = 2-num; B(1,3) = 5-num; B(1,4) = 5-num;
00211     B(2,1) = 2-num; B(2,2) = 1-num; B(2,3) = 3-num; B(2,4) = 6-num;
00212     B(3,1) = 5-num; B(3,2) = 3-num; B(3,3) = 2-num; B(3,4) = 3-num;
00213     B(4,1) = 1-num; B(4,2) = 2-num; B(4,3) = 4-num; B(4,4) = 1-num;
00214
00215
00216     Matrix R=A-num;
00217
00218     _assert(m_equals(B, R, 1e-11));
```

```
00219
00220     return 0;
00221 }
00222
00223 int m_mul_d_01() {
00224     int f = 4;
00225     int c = 4;
00226     double num=2;
00227
00228     Matrix A(f, c);
00229     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00230     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00231     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00232     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00233
00234
00235     Matrix B(f, c);
00236     B(1,1) = 1.*num; B(1,2) = 2.*num; B(1,3) = 5.*num; B(1,4) = 5.*num;
00237     B(2,1) = 2.*num; B(2,2) = 1.*num; B(2,3) = 3.*num; B(2,4) = 6.*num;
00238     B(3,1) = 5.*num; B(3,2) = 3.*num; B(3,3) = 2.*num; B(3,4) = 3.*num;
00239     B(4,1) = 1.*num; B(4,2) = 2.*num; B(4,3) = 4.*num; B(4,4) = 1.*num;
00240
00241     Matrix R=A*num;
00242
00243     _assert(m_equals(R, B, 1e-11));
00244
00245     return 0;
00246 }
00247 int m_div_d_01() {
00248     int f = 4;
00249     int c = 4;
00250     double num=2;
00251
00252     Matrix A(f, c);
00253     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00254     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00255     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00256     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00257
00258
00259     Matrix B(f, c);
00260     B(1,1) = 1./num; B(1,2) = 2./num; B(1,3) = 5./num; B(1,4) = 5./num;
00261     B(2,1) = 2./num; B(2,2) = 1./num; B(2,3) = 3./num; B(2,4) = 6./num;
00262     B(3,1) = 5./num; B(3,2) = 3./num; B(3,3) = 2./num; B(3,4) = 3./num;
00263     B(4,1) = 1./num; B(4,2) = 2./num; B(4,3) = 4./num; B(4,4) = 1./num;
00264
00265     Matrix R=A/num;
00266
00267     _assert(m_equals(R, B, 1e-11));
00268
00269     return 0;
00270 }
00271 int m_asig_01() {
00272     int f = 4;
00273     int c = 4;
00274
00275     Matrix A(f, c);
00276     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00277     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00278     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00279     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00280
00281     Matrix B=A;
00282
00283     _assert(m_equals(A, B, 1e-11));
00284
00285     return 0;
00286 }
00287 int m_zeros_01() {
00288     int f = 3;
00289     int c = 4;
00290
00291     Matrix A(f, c);
00292     A(1,1) = 0; A(1,2) = 0; A(1,3) = 0; A(1,4) = 0;
00293     A(2,1) = 0; A(2,2) = 0; A(2,3) = 0; A(2,4) = 0;
00294     A(3,1) = 0; A(3,2) = 0; A(3,3) = 0; A(3,4) = 0;
00295
00296     Matrix B = zeros(3, 4);
00297
00298     _assert(m_equals(A, B, 1e-11));
00299
00300     return 0;
00301 }
00302
00303 int m_eye_01() {
00304     int f = 3;
00305
```

```
00306
00307     Matrix A(f, f);
00308     A(1,1) = 1; A(1,2) = 0; A(1,3) = 0;
00309     A(2,1) = 0; A(2,2) = 1; A(2,3) = 0;
00310     A(3,1) = 0; A(3,2) = 0; A(3,3) = 1;
00311
00312
00313     Matrix B = eye(f);
00314
00315     _assert(m_equals(A, B, 1e-11));
00316
00317
00318     return 0;
00319 }
00320
00321 int m_transpose_01() {
00322     int f = 3;
00323     int c = 3;
00324
00325
00326     Matrix A(f, c);
00327     A(1,1) = 1; A(1,2) = 4; A(1,3) = 9;
00328     A(2,1) = 2; A(2,2) = 3; A(2,3) = 8;
00329     A(3,1) = 5; A(3,2) = 6; A(3,3) = 7;
00330
00331
00332     Matrix B(f,c);
00333
00334     B(1,1) = 1; B(1,2) = 2; B(1,3) = 5;
00335     B(2,1) = 4; B(2,2) = 3; B(2,3) = 6;
00336     B(3,1) = 9; B(3,2) = 8; B(3,3) = 7;
00337
00338     Matrix R=transpose(A);
00339
00340     _assert(m_equals(R, B, 1e-11));
00341
00342     return 0;
00343 }
00344 int m_inv_01() {
00345     int f = 4;
00346
00347
00348     Matrix A(f, f);
00349     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00350     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3; A(2,4) = 6;
00351     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2; A(3,4) = 3;
00352     A(4,1) = 1; A(4,2) = 2; A(4,3) = 4; A(4,4) = 1;
00353
00354     Matrix B(f, f);
00355
00356     B(1,1)=-47./42; B(1,2) = 5./6; B(1,3) = -1./14 ; B(1,4) =17./21;
00357     B(2,1)=41./21; B(2,2) = -5./3; B(2,3) = 4./7 ; B(2,4) =-31./21;
00358     B(3,1) = -17./21 ; B(3,2) = 2./3 ; B(3,3) = -2./7; B(3,4) =19./21;
00359     B(4,1) = 19./42 ; B(4,2) = -1./6; B(4,3) = 1./14 ; B(4,4) = -10./21;
00360
00361     Matrix R=inv(A);
00362     _assert(m_equals(R, B, 1e-11));
00363     return 0;
00364 }
00365 int m_norm_01() {
00366     int f = 4;
00367
00368
00369     Matrix A(f);
00370     A(1,1) = 1; A(1,2) = 2; A(1,3) = 5; A(1,4) = 5;
00371
00372     double B=7.4161984870956629487113974408007;
00373
00374     double R=norm(A);
00375     _assert(fabs(R-B)<1e-11);
00376     return 0;
00377 }
00378
00379 int m_dot_01() {
00380     int f = 3;
00381     Matrix A(f);
00382     A(1,1) = 1; A(1,2) = 2; A(1,3) = 3;
00383
00384     Matrix B(f);
00385     B(1,1)= 1; B(1,2) = 2; B(1,3) = 3 ;
00386
00387
00388     double R=14;
00389
00390     double C=dot(A,B);
00391
00392     _assert(fabs(R-C)<1e-11);
```

```
00393
00394     return 0;
00395 }
00396 int m_cross_01() {
00397     int f = 3;
00398     Matrix A(f);
00399     Matrix B(f);
00400     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00401     B(1,1)= 3; B(1,2) = 5; B(1,3) = 6 ;
00402
00403     Matrix R(f);
00404     R(1,1)= 6; R(1,2) = -12; R(1,3) = 7 ;
00405
00406     Matrix C=cross(A,B);
00407
00408     _assert(m_equals(R, C, 1e-11));
00409
00410     return 0;
00411 }
00412 int m_extract_vector_01() {
00413     int f = 5;
00414     Matrix A(f);
00415
00416     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0; A(1,4) = 5; A(1,5) = 1;
00417
00418     Matrix B=extract_vector(A,1,3);
00419
00420     Matrix R(3);
00421     R(1,1)= 2; R(1,2) = 1; R(1,3) = 0 ;
00422
00423
00424     _assert(m_equals(R, B, 1e-11));
00425
00426     return 0;
00427 }
00428
00429 int m_union_vector_01() {
00430     int f = 3;
00431     Matrix A(f);
00432     Matrix B(f);
00433     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00434     B(1,1)= 3; B(1,2) = 1; B(1,3) = 6 ;
00435
00436     Matrix R(6);
00437     R(1,1) = 2; R(1,2) = 1; R(1,3) = 0;
00438     R(1,4)= 3; R(1,5) = 1;R(1,6) = 6 ;
00439     Matrix C=union_vector(A,B);
00440
00441     _assert(m_equals(R, C, 1e-11));
00442
00443     return 0;
00444 }
00445
00446 int m_extract_row_01() {
00447     int f = 3;
00448     Matrix A(f,f);
00449
00450     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00451     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3;
00452     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2;
00453
00454     Matrix B=extract_row(A,f);
00455
00456     Matrix R(f);
00457     R(1,1)= 5; R(1,2) = 3; R(1,3) = 2 ;
00458
00459     _assert(m_equals(R, B, 1e-11));
00460
00461     return 0;
00462
00463 }
00464
00465 int m_extract_column_01() {
00466     int f = 3;
00467     Matrix A(f,f);
00468
00469     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00470     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3;
00471     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2;
00472
00473     Matrix B=extract_column(A,f);
00474
00475     Matrix R(f);
00476     R(1,1)= 0; R(1,2) = 3; R(1,3) = 2 ;
00477
00478
00479     _assert(m_equals(R, B, 1e-11));
```

```
00480
00481     return 0;
00482 }
00483
00484 int m_assign_row_01() {
00485     int f = 3;
00486     Matrix A(f,f);
00487
00488     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00489     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3;
00490     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2;
00491
00492     Matrix B(f);
00493
00494     B(1,1)= 3; B(1,2) = 5; B(1,3) = 6 ;
00495
00496
00497     Matrix C=assign_row(A,B,f);
00498
00499     Matrix R(f,f);
00500     R(1,1) = 2; R(1,2) = 1; R(1,3) = 0;
00501     R(2,1) = 2; R(2,2) = 1; R(2,3) = 3;
00502     R(3,1) = 3; R(3,2) = 5; R(3,3) = 6;
00503
00504     _assert(m_equals(R, C, 1e-11));
00505
00506     return 0;
00507 }
00508
00509 int m_assign_column_01() {
00510     int f = 3;
00511     Matrix A(f,f);
00512
00513     A(1,1) = 2; A(1,2) = 1; A(1,3) = 0;
00514     A(2,1) = 2; A(2,2) = 1; A(2,3) = 3;
00515     A(3,1) = 5; A(3,2) = 3; A(3,3) = 2;
00516
00517     Matrix B(f);
00518
00519     B(1,1)= 3; B(1,2) = 5; B(1,3) = 6 ;
00520
00521     Matrix C=assign_column(A,B,f);
00522
00523     Matrix R(f,f);
00524     R(1,1) = 2; R(1,2) = 1; R(1,3) = 3;
00525     R(2,1) = 2; R(2,2) = 1; R(2,3) = 5;
00526     R(3,1) = 5; R(3,2) = 3; R(3,3) = 6;
00527
00528
00529     _assert(m_equals(R, C, 1e-11));
00530
00531     return 0;
00532 }
00533
00534 int m_R_x_01() {
00535     Matrix A=R_x(10);
00536
00537
00538     Matrix R(3,3);
00539     R(1,1) = 1; R(1,2) = 0; R(1,3) = 0;
00540     R(2,1) = 0; R(2,2) = -0.839071529076452; R(2,3) = -0.54402111088937;
00541     R(3,1) = 0; R(3,2) = 0.54402111088937; R(3,3) = -0.839071529076452;
00542
00543     _assert(m_equals(R, A, 1e-11));
00544
00545     return 0;
00546 }
00547
00548 int m_R_y_01() {
00549
00550     Matrix A=R_y(10);
00551
00552     Matrix R(3,3);
00553     R(1,1) = -0.839071529076452 ; R(1,2) = 0; R(1,3) = 0.54402111088937;
00554     R(2,1) = 0                  ; R(2,2) = 1; R(2,3) = 0;
00555     R(3,1) = -0.54402111088937  ; R(3,2) = 0; R(3,3) = -0.839071529076452;
00556
00557
00558     _assert(m_equals(R, A, 1e-11));
00559
00560     return 0;
00561 }
00562
00563 int m_R_z_01() {
00564
00565     Matrix A=R_z(10);
00566
```

```
00567
00568     Matrix R(3,3);
00569     R(1,1) = -0.839071529076452; R(1,2) = -0.54402111088937; R(1,3) = 0;
00570     R(2,1) = 0.54402111088937; R(2,2) = -0.839071529076452; R(2,3) = 0;
00571     R(3,1) = 0; R(3,2) = 0; R(3,3) = 1;
00572
00573
00574     _assert(m_equals(R, A, 1e-11));
00575
00576     return 0;
00577 }
00578
00579 int m_AccelPointMass_01() {
00580
00581     Matrix A(3);
00582
00583     A(1,1) = 1; A(1,2) = 1; A(1,3) = 1;
00584
00585     Matrix B(3);
00586
00587     B(1,1)= 2; B(1,2) = 3; B(1,3) = 4 ;
00588
00589     Matrix C=AccelPointMass(A,B,10);
00590
00591     Matrix R(3);
00592     R(1,1) = 0.0628351366133708; R(1,2) = 0.189703148460208; R(1,3) = 0.316571160307045;
00593
00594     _assert(m_equals(R, A, 1e-11));
00595
00596     return 0;
00597 }
00598 int m_Cheb3D_01() {
00599     double f = 3;
00600
00601     Matrix A(f);
00602
00603     A(1,1) = 1; A(1,2) = 2; A(1,3) = 3;
00604
00605     Matrix B(f);
00606
00607     B(1,1)= 1; B(1,2) = 2; B(1,3) = 3;
00608
00609     Matrix C(f);
00610
00611     C(1,1)= 5; C(1,2) = 2; C(1,3) = 3;
00612
00613     Matrix D=Cheb3D(1,3,0.5,1,A,B,C);
00614
00615     Matrix R(f);
00616     R(1,1) = 6; R(1,2) = 6; R(1,3) = 10;
00617
00618     _assert(m_equals(R, D, 1e-11));
00619
00620     return 0;
00621 }
00622 int m_EccAnom_01() {
00623
00624     double R = 2.38006127313934;
00625     double D=EccAnom(1,2);
00626
00627     _assert(fabs(R-D)< 1e-11);
00628
00629     return 0;
00630 }
00631 int m_Frac_01() {
00632
00633     double R = 0.3801;
00634     double D=Frac(2.3801);
00635
00636     _assert(fabs(R-D)< 1e-11);
00637
00638     return 0;
00639 }
00640
00641 int m_MeanObliquity_01() {
00642
00643     double R = 0.409412815476201;
00644     double D=MeanObliquity(41);
00645
00646     _assert(fabs(R-D)< 1e-11);
00647
00648     return 0;
00649 }
00650
00651
00652 int m_Mjday_01() {
00653
```

```
00654     double R = 60800;
00655     double D= Mjday(2025,5,5);
00656 
00657     _assert(fabs(R-D)< 1e-11);
00658 
00659     return 0;
00660 }
00661 int m_Mjday_TDB_01() {
00662 
00663     double R = 2025.0000000092;
00664     double D= Mjday_TDB(2025);
00665 
00666     _assert(fabs(R-D)< 1e-11);
00667 
00668     return 0;
00669 }
00670 int m_Position_01() {
00671 
00672     Matrix R(3);
00673     R(1) = 2.627855739427486e+06; R(2) = -5.741969545549633e+06; R(3) = 8.941173180321892e+05;
00674 
00675     Matrix D= Position(2,3,4);
00676     _assert(m_equals(R, D, 1e-8));
00677 
00678     return 0;
00679 }
00680 int m_sign__01() {
00681 
00682     double R = -4;
00683     double D= sign_(4,-3);
00684 
00685     _assert(fabs(R-D)< 1e-11);
00686 
00687     return 0;
00688 }
00689 int m_timediff_01() {
00690 
00691     double R0 = -6;
00692     double R1 = 9;
00693     double R2 = 13;
00694     double R3 = 42.184;
00695     double R4 = -9;
00696     auto D= timediff(4,10);
00697     _assert(fabs(get<0>(D)-R0)< 1e-11);
00698     _assert(fabs(get<1>(D)-R1)< 1e-11);
00699     _assert(fabs(get<2>(D)-R2)< 1e-11);
00700     _assert(fabs(get<3>(D)-R3)< 1e-11);
00701     _assert(fabs(get<4>(D)-R4)< 1e-11);
00702 
00703     return 0;
00704 }
00705 int m_AzElPa_01() {
00706     Matrix A(3);
00707 
00708     A(1) = 1; A(2) = 2; A(3) = 3;
00709 
00710     double R0=0.463647609000806;
00711 
00712     double R1=0.930274014115472;
00713 
00714     Matrix R2(3);
00715 
00716     R2(1) = 0.4; R2(2) = -0.2; R2(3) = 0;
00717     Matrix R3(3);
00718 
00719     R3(1) = -0.095831484749991; R3(2) = -0.191662969499982; R3(3) = 0.159719141249985;
00720 
00721     auto [Az, El, dAds, dEds]= AzElPa(A);
00722     _assert(fabs(Az-R0)< 1e-11);
00723     _assert(fabs(El-R1)< 1e-11);
00724     _assert(m_equals(dAds,R2,1e-11));
00725     _assert(m_equals(dEds,R3,1e-11));
00726 
00727 
00728     return 0;
00729 }
00730 
00731 int m_IERS_01() {
00732 
00733 
00734     eop19620101(21413);
00735 
00736     double R0 = -5.59518621231704e-07;
00737     double R1 = 2.33458634442529e-06;
00738     double R2 =  0.3260677;
00739     double R3 = 0.0027213;
00740     double R4 = -1.16864337831454e-07;
```

```
00741     double R5 = -2.48709418409192e-08;
00742     double R6 = -8.19335121075116e-10;
00743     double R7 =  -1.53201123230613e-09;
00744     double R8 = 29;
00745 
00746 
00747     auto [x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC]= IERS(eopdata,49746,'l');
00748     _assert(fabs(x_pole-R0)< 1e-11);
00749     _assert(fabs(y_pole-R1)< 1e-11);
00750     _assert(fabs(UT1_UTC-R2)< 1e-11);
00751     _assert(fabs(LOD-R3)< 1e-11);
00752     _assert(fabs(dpsi-R4)< 1e-11);
00753     _assert(fabs(deps-R5)< 1e-11);
00754     _assert(fabs(dx_pole-R6)< 1e-11);
00755     _assert(fabs(dy_pole-R7)< 1e-11);
00756     _assert(fabs(TAI_UTC-R8)< 1e-11);
00757 
00758 
00759     return 0;
00760 }
00761 
00762 int m_Legendre_01() {
00763 
00764 
00765     Matrix R0 (4,4);
00766     R0(1,1) = 1; R0(1,2) = 0; R0(1,3) = 0; R0(1,4) = 0;
00767     R0(2,1) =  1.4574704987823; R0(2,2) =  0.935831045210238; R0(2,3) = 0; R0(2,4) = 0;
00768     R0(3,1) =  1.25691645573063 ; R0(3,2) = 1.76084689542256  ; R0(3,3) = 0.565313394670859 ; R0(3,4)
      = 0;
00769     R0(4,1) = 0.601515831515714; R0(4,2) = 2.22381140389174 ; R0(4,3) = 1.25857019087392 ; R0(4,4) =
      0.329913047636197;
00770     Matrix R1 (4,4);
00771     R1(1,1) = 0; R1(1,2) = 0; R1(1,3) = 0; R1(1,4) = 0;
00772     R1(2,1) = 0.935831045210238 ; R1(2,2) = -1.4574704987823 ; R1(2,3) = 0; R1(2,4) = 0;
00773     R1(3,1) =  3.0498762872218; R1(3,2) = -1.61172976752398 ; R1(3,3) = -1.76084689542256  ; R1(3,4) =
      0;
00774     R1(4,1) = 5.44720322371707  ; R1(4,2) =  0.516567339757783 ; R1(4,3) =  -3.11209524837966 ;
      R1(4,4) = -1.54142738655916;
00775 
00776     auto [pnm, dpnm]= Legendre(3,3,1);
00777     _assert(m_equals(pnm,R0, 1e-11));
00778     _assert(m_equals(dpnm,R1,1e-11));
00779 
00780 
00781     return 0;
00782 }
00783 int m_NutAngles_01() {
00784 
00785 
00786     double R0 = 2.72256565175042e-05;
00787     double R1 =  3.87947551912632e-05;
00788 
00789     auto [dpsi, deps]= NutAngles(3);
00790     _assert(fabs(dpsi-R0)< 1e-11);
00791     _assert(fabs(deps-R1)< 1e-11);
00792 
00793 
00794     return 0;
00795 }
00796 
00797 int m_TimeUpdate_01() {
00798 
00799     Matrix A(3, 3);
00800     A(1,1) = 1; A(1,2) = 4; A(1,3) = 9;
00801     A(2,1) = 2; A(2,2) = 3; A(2,3) = 8;
00802     A(3,1) = 5; A(3,2) = 6; A(3,3) = 7;
00803 
00804 
00805     Matrix B(3,3);
00806 
00807     B(1,1) = 1; B(1,2) = 2; B(1,3) = 5;
00808     B(2,1) = 4; B(2,2) = 3; B(2,3) = 6;
00809     B(3,1) = 9; B(3,2) = 8; B(3,3) = 7;
00810 
00811     Matrix C(3, 3);
00812 
00813     C(1,1) = 52 ; C(1,2) = 102  ; C(1,3) = 77   ;
00814     C(2,1) = 22 ; C(2,2) = 34   ; C(2,3) = 34   ;
00815     C(3,1) = 23 ; C(3,2) = 36   ; C(3,3) = 32   ;
00816 
00817 
00818     Matrix D(3, 3);
00819 
00820     D(1,1) = 462    ; D(1,2) = 702  ; D(1,3) = 1087 ;
00821     D(2,1) = 694    ; D(2,2) = 989  ; D(2,3) = 1596 ;
00822     D(3,1) = 1257   ; D(3,2) = 1746 ; D(3,3) = 2746 ;
00823 
```

```
00824     Matrix R = TimeUpdate(A,B,C);
00825
00826     _assert(m_equals(R,D,1e-11));
00827
00828     return 0;
00829 }
00830 int m_AccelHarmonic_01() {
00831
00832     Matrix R0(3);
00833     R0(1)=2.42488766379455e+34;
00834     R0(2)=2.65762182552943e+34;
00835     R0(3)=2.65762182552943e+34;
00836
00837     Matrix A(3);
00838     A(1)=1.0;
00839     A(2)=2.0;
00840     A(3)=3.0;
00841     A=transpose(A);
00842     Matrix B(3,3);
00843     B(1,1)= 1.0; B(1,2) = 1.0; B(1,3) = 1.0 ;
00844     B(2,1)= 1.0; B(2,2) = 2.0; B(2,3) = 2.0 ;
00845     B(3,1)= 1.0; B(3,2) = 3.0; B(3,3) = 3.0 ;
00846
00847     Matrix R = AccelHarmonic(A,B,5,5);
00848     _assert(m_equals(R,R0,R0(1)*1e-11));
00849
00850     return 0;
00851 }
00852
00853 int m_EqnEquinox_01() {
00854
00855     double R0=2.6045897022442e-05;
00856     double R = EqnEquinox(5);
00857
00858     _assert(fabs(R-R0)< 1e-11);
00859
00860     return 0;
00861 }
00862
00863 int m_JPL_Eph_DE430_01() {
00864
00865     Matrix R0(3);
00866     R0(1)=147208460159.245;
00867     R0(2)=  54592844683.9181;
00868     R0(3)= 15319523517.8098;
00869     Matrix R1(3);
00870     R1(1)= 72752904522.483;
00871     R1(2)=2340227175.73022;
00872     R1(3)=1670913926.26657;
00873     Matrix R2(3);
00874     R2(1)=-108493583087.765;
00875     R2(2)=-97599455066.7732;
00876     R2(3)=-42280555048.3341;
00877     Matrix R3(3);
00878     R3(1)=-131548434829.954;
00879     R3(2)= 156673495960.063;
00880     R3(3)= 75876841465.0958;
00881     Matrix R4(3);
00882     R4(1)=125152801267.633;
00883     R4(2)= 801496792415.556;
00884     R4(3)=343590087271.679;
00885     Matrix R5(3);
00886     R5(1)= 1532648188929.54;
00887     R5(2)=-30170201818.4012;
00888     R5(3)=-71835402852.1091;
00889     Matrix R6(3);
00890     R6(1)= 1707620424797.68;
00891     R6(2)= 2344787198692.63;
00892     R6(3)=1003869836966.23;
00893     Matrix R7(3);
00894     R7(1)= 4578089056071.27;
00895     R7(2)= 104288160816.022;
00896     R7(3)= -66258775563.9107;
00897     Matrix R8(3);
00898     R8(1)=2885822907234;
00899     R8(2)= -3876433792795.09;
00900     R8(3)=-2034695099132.45;
00901     Matrix R9(3);
00902     R9(1)=-295931131.772483;
00903     R9(2)= 224622149.622798;
00904     R9(3)= 120216992.495936;
00905     Matrix R10(3);
00906     R10(1)= 107770931597.498;
00907     R10(2)=96865992179.6201;
00908     R10(3)=41989334136.8052;
00909     auto [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus,r_Neptune,r_Pluto,r_Moon,r_Sun]
      = JPL_Eph_DE430(60800);
```

```
00910     _assert(m_equals(r_Mercury,R0, abs(R0(1)*1e-11)));
00911     _assert(m_equals(r_Venus,R1,abs(R1(1)*1e-11)));
00912     _assert(m_equals(r_Earth,R2, abs(R2(1)*1e-11)));
00913     _assert(m_equals(r_Mars,R3,abs(R3(1)*1e-11)));
00914     _assert(m_equals(r_Jupiter,R4, abs(R4(1)*1e-11)));
00915     _assert(m_equals(r_Saturn,R5,abs(R5(1)*1e-11)));
00916     _assert(m_equals(r_Uranus,R6, abs(R6(1)*1e-11)));
00917     _assert(m_equals(r_Neptune,R7,abs(R7(1)*1e-11)));
00918     _assert(m_equals(r_Pluto,R8, abs(R8(1)*1e-11)));
00919     _assert(m_equals(r_Moon,R9,abs(R9(1)*1e-11)));
00920     _assert(m_equals(r_Sun,R10, abs(R10(1)*1e-11)));
00921
00922     return 0;
00923 }
00924
00925
00926 int m_LTC_01() {
00927
00928     Matrix A=LTC(10,10);
00929
00930
00931     Matrix R(3,3);
00932     R(1,1) = 0.54402111088937; R(1,2) = -0.839071529076452; R(1,3) = 0;
00933     R(2,1) = -0.456472625363814; R(2,2) = -0.295958969093304; R(2,3) = -0.839071529076452;
00934     R(3,1) = 0.704041030906696; R(3,2) = 0.456472625363814; R(3,3) = -0.54402111088937;
00935
00936
00937     _assert(m_equals(R, A, 1e-11));
00938
00939     return 0;
00940 }
00941
00942
00943 int m_NutMatrix_01() {
00944
00945     Matrix A=NutMatrix(10);
00946
00947
00948     Matrix R(3,3);
00949     R(1,1) = 0.999999999492159; R(1,2) = -2.92358797494727e-05 ; R(1,3) = -1.26864277798066e-05;
00950     R(2,1) = 2.9235393806329e-05 ; R(2,2) = 0.999999998839099; R(2,3) = -3.83026695232602e-05;
00951     R(3,1) = 1.26875475773192e-05; R(3,2) =  3.83022986110704e-05 ; R(3,3) =  0.99999999918598;
00952
00953
00954     _assert(m_equals(R, A, 1e-11));
00955
00956     return 0;
00957 }
00958 int m_PoleMatrix_01() {
00959
00960     Matrix A=PoleMatrix(10,10);
00961
00962
00963     Matrix R(3,3);
00964     R(1,1) = -0.839071529076452; R(1,2) = 0.295958969093304; R(1,3) = 0.456472625363814;
00965     R(2,1) = 0; R(2,2) = -0.839071529076452; R(2,3) = 0.54402111088937;
00966     R(3,1) = 0.54402111088937; R(3,2) = 0.456472625363814; R(3,3) = 0.704041030906696;
00967
00968
00969     _assert(m_equals(R, A, 1e-11));
00970
00971     return 0;
00972    }
00973
00974 int m_PrecMatrix_01() {
00975
00976     Matrix A=PrecMatrix(100,1);
00977
00978
00979     Matrix R(3,3);
00980     R(1,1) = 0.999999997819034; R(1,2) =  6.05590736738844e-05; R(1,3) = 2.63539319986234e-05;
00981     R(2,1) =  -6.05590736738844e-05; R(2,2) = 0.999999998166299 ; R(2,3) = -7.97984483806257e-10;
00982     R(3,1) = -2.63539319986234e-05; R(3,2) =   -7.97985227435292e-10; R(3,3) =  0.999999999652735;
00983
00984
00985     _assert(m_equals(R, A, 1e-11));
00986
00987     return 0;
00988    }
00989    int m_gmst_01() {
00990
00991     double A=gmst(10);
00992
00993
00994     double R=1.14523606099042;
00995
00996
```

```
00997     _assert(fabs(R-A)< 1e-11);
00998
00999     return 0;
01000    }
01001
01002    int m_gast_01() {
01003
01004     double A=gast(10);
01005
01006
01007     double R=1.14526529687017;
01008
01009
01010     _assert(fabs(R-A)< 1e-11);
01011
01012     return 0;
01013    }
01014
01015 int m_MeasUpdate_01() {
01016
01017     Matrix A(3);
01018     A(1)=1;
01019     A(2)=2;
01020     A(3)=3;
01021     Matrix B=transpose(A);
01022
01023     Matrix C(3,3);
01024     C(1,1) = 1; C(1,2) = 2; C(1,3) = 3;
01025     C(2,1) = 6; C(2,2) = 2; C(2,3) = 3;
01026     C(3,1) = 8; C(3,2) = 2; C(3,3) = 3;
01027
01028     auto [K, x, P]=MeasUpdate(B,2,3,4,A,C,3);
01029
01030
01031     Matrix R0(3);
01032     R0(1)=0.106870229007634;
01033     R0(2)=  0.145038167938931;
01034     R0(3)= 0.16030534351145;
01035     R0=transpose(R0);
01036     Matrix R1(3);
01037     R1(1)= 0.893129770992366;
01038     R1(2)= 1.85496183206107;
01039     R1(3)=2.83969465648855;
01040     R1=transpose(R1);
01041
01042     Matrix R2(3,3);
01043     R2(1,1) = -2.95419847328244  ; R2(1,2) = 0.717557251908397; R2(1,3) = 1.0763358778626;
01044     R2(2,1) = 0.633587786259541; R2(2,2) = 0.2595419847328244; R2(2,3) = 0.389312977099237;
01045     R2(3,1) = 2.06870229007634; R2(3,2) = 0.0763358778625954 ; R2(3,3) = 0.1145038167938931;
01046     _assert(m_equals(R0, K, 1e-11));
01047     _assert(m_equals(R1, x, 1e-11));
01048     _assert(m_equals(R2, P, 1e-11));
01049
01050     return 0;
01051    }
01052    int m_G_AccelHarmonic_01() {
01053
01054     Matrix R2(3,3);
01055     R2(1,1) = -2.0122905124052e+34   ; R2(1,2) =-3.29511632095482e+34 ; R2(1,3) =
      -3.29511632095482e+34;
01056     R2(2,1) = -3.11072371483497e+34  ; R2(2,2) = -3.38401367341354e+34 ; R2(2,3) =
      -3.38401367341354e+34;
01057     R2(3,1) =  -3.11072371483497e+34 ; R2(3,2) =  -3.38401367341354e+34 ; R2(3,3) =
      -3.38401367341354e+34;
01058
01059     Matrix A(3);
01060     A(1)=1.0;
01061     A(2)=2.0;
01062     A(3)=3.0;
01063     A=transpose(A);
01064     Matrix B(3,3);
01065     B(1,1)= 1.0; B(1,2) = 1.0; B(1,3) = 1.0 ;
01066     B(2,1)= 1.0; B(2,2) = 2.0; B(2,3) = 2.0 ;
01067     B(3,1)= 1.0; B(3,2) = 3.0; B(3,3) = 3.0 ;
01068
01069     Matrix R = G_AccelHarmonic(A,B,5,5);
01070     _assert(m_equals(R,R2,fabs(R2(1)*1e-11)));
01071
01072     return 0;
01073 }
01074 int m_GHAMatrix_01() {
01075
01076     Matrix R(3,3);
01077     R(1,1) = 0.412804512414729; R(1,2) = 0.910819649837463 ; R(1,3) = 0;
01078     R(2,1) = -0.910819649837463 ; R(2,2) = 0.412804512414729; R(2,3) = 0;
01079     R(3,1) = 0; R(3,2) = 0; R(3,3) = 1;
01080
```

```
01081
01082     Matrix A = GHAMatrix(10);
01083
01084     _assert(m_equals(R,A,1e-11));
01085
01086     return 0;
01087 }
01088
01089 int m_Accel_01() {
01090
01091     Matrix R(6);
01092     R(1)=1.0;
01093     R(2)=2.0;
01094     R(3)=3.0;
01095     R(4)=-9.52489066332755e+131;
01096     R(5)=-1.68703107956274e+132;
01097     R(6)=-4.07471909292663e+132;
01098
01099     Matrix A(6);
01100     A(1)=1.0;
01101     A(2)=2.0;
01102     A(3)=3.0;
01103     A(4)=1.0;
01104     A(5)=2.0;
01105     A(6)=3.0;
01106     A=transpose(A);
01107     Matrix B = Accel(10,A);
01108
01109
01110     _assert(m_equals(R,B,abs(R(6)*1e-11)));
01111
01112     return 0;
01113 }
01114 int m_VarEqn_01() {
01115 Matrix A(42);
01116     A(1)=7101800.90695315;
01117     A(2)=1293997.58115302;
01118     A(3)=10114.014948955;
01119     A(4)= 573.068082065557;
01120     A(5)= -3085.15736953138;
01121     A(6)=      -6736.03068347156;
01122     A(7)=        1.0000293469741;
01123     A(8)=    8.22733917593032e-06;
01124     A(9)=   2.17104932968693e-07;
01125     A(10)=    1.08925458231315e-05;
01126     A(11)=   3.04673932160225e-06;
01127     A(12)=    6.63504292706821e-08;
01128     A(13)=    8.22733944423959e-06;
01129     A(14)=      0.999986101965304;
01130     A(15)=   3.99927483270551e-08;
01131     A(16)=    3.04673960163327e-06;
01132     A(17)=   -5.1596062466179e-06;
01133     A(18)=    1.22075292404534e-08;
01134     A(19)=   2.17105640392839e-07;
01135     A(20)=     3.9992870847826e-08;
01136     A(21)=       0.999984551298692;
01137     A(22)=    6.63510875632706e-08;
01138     A(23)=    1.22076480274715e-08;
01139     A(24)=  -5.73276287738792e-06;
01140     A(25)=      5.38976081674752;
01141     A(26)=    1.47507305174403e-05;
01142     A(27)=   3.21241787851554e-07;
01143     A(28)=       1.00002936035846;
01144     A(29)=   8.19365458482084e-06;
01145     A(30)=    1.40504658112974e-07;
01146     A(31)=    1.47507306419397e-05;
01147     A(32)=        5.38968310056198;
01148     A(33)=    5.90697768748029e-08;
01149     A(34)=   8.19365482653896e-06;
01150     A(35)=          0.9999860891763;
01151     A(36)=    2.58022974647481e-08;
01152     A(37)=     3.21242427100724e-07;
01153     A(38)=  5.90698876854246e-08;
01154     A(39)=         5.38968032557769;
01155     A(40)=      1.4050537070756e-07;
01156     A(41)=    2.58024285760964e-08;
01157     A(42)=       0.999984550703337;
01158     Matrix R(42);
01159     R(1) = 573.068082065557;
01160     R(2) = -3085.15736953138;
01161     R(3) = -6736.03068347156;
01162     R(4) = -7.53489822593659;
01163     R(5) = -1.37294429126638;
01164     R(6) = -0.0107597986473575;
01165     R(7) = 1.08925458231315e-05;
01166     R(8) = 3.04673932160225e-06;
01167     R(9) = 6.63504292706821e-08;
```

```
01168     R(10) = 2.02239897508587e-06;
01169     R(11) = 5.61811901849645e-07;
01170     R(12) = 4.39846387071934e-09;
01171     R(13) = 3.04673960163327e-06;
01172     R(14) = -5.1596062466179e-06;
01173     R(15) = 1.22075292404534e-08;
01174     R(16) = 5.61812134084449e-07;
01175     R(17) = -9.58613689243416e-07;
01176     R(18) = 8.05616500343474e-10;
01177     R(19) = 6.63510875632706e-08;
01178     R(20) = 1.22076480274715e-08;
01179     R(21) = -5.73276287738792e-06;
01180     R(22) = 4.39895597958216e-09;
01181     R(23) = 8.0570607835305e-10;
01182     R(24) = -1.06368693580442e-06;
01183     R(25) = 1.00002936035846;
01184     R(26) = 8.19365458482084e-06;
01185     R(27) = 1.40504658112974e-07;
01186     R(28) = 1.08999102436198e-05;
01187     R(29) = 3.02797128053784e-06;
01188     R(30) = 2.37068516291712e-08;
01189     R(31) = 8.19365482653896e-06;
01190     R(32) = 0.9999860891763;
01191     R(33) = 2.58022974647481e-08;
01192     R(34) = 3.02797160153579e-06;
01193     R(35) = -5.16671243316801e-06;
01194     R(36) = 4.34211426867344e-09;
01195     R(37) = 1.4050537070756e-07;
01196     R(38) = 2.58024285760964e-08;
01197     R(39) = 0.999984550703337;
01198     R(40) = 2.37075280907946e-08;
01199     R(41) = 4.34223837651307e-09;
01200     R(42) = -5.73302112206999e-06;
01201     R=transpose(R);
01202     Matrix B = VarEqn( 5.38970808087706,A);
01203
01204     _assert(m_equals(R,B,abs(R(1)*1e-11)));
01205
01206     return 0;
01207 }
01208 int m_DEInteg_01() {
01209
01210     Matrix R(6);
01211     R(1)=5542555.89427452;
01212     R(2)=3213514.83814162;
01213     R(3)= 3990892.92789074;
01214     R(4)=5394.06894044389;
01215     R(5)=-2365.2129057402;
01216     R(6)=-7061.8448137347;
01217
01218     Matrix A(6);
01219     A(1)=   6221397.62857869;
01220     A(2)=   2867713.77965738;
01221     A(3)=   3006155.98509949;
01222     A(4)=   4645.04725161806;
01223     A(5)=  -2752.21591588204;
01224     A(6)=  -7507.99940987031;
01225
01226     A=transpose(A);
01227     Matrix B = DEInteg(Accel,0,-134.999991953373,1e-13,1e-6,6,A);
01228
01229
01230     _assert(m_equals(R,B,abs(R(5)*1e-11)));
01231
01232     return 0;
01233 }
01234
01235 int all_tests()
01236 {
01237     _verify(m_sum_01);
01238     _verify(m_sub_01);
01239     _verify(m_mul_01);
01240     _verify(m_div_01);
01241     _verify(m_sum_d_01);
01242     _verify(m_sub_d_01);
01243     _verify(m_mul_d_01);
01244     _verify(m_div_d_01);
01245     _verify(m_asig_01);
01246     _verify(m_zeros_01);
01247     _verify(m_eye_01);
01248     _verify(m_transpose_01);
01249     _verify(m_inv_01);
01250     _verify(m_norm_01);
01251     _verify(m_dot_01);
01252     _verify(m_cross_01);
01253     _verify(m_extract_vector_01);
01254     _verify(m_union_vector_01);
```

```
01255     _verify(m_extract_row_01);
01256     _verify(m_extract_column_01);
01257     _verify(m_assign_row_01);
01258     _verify(m_assign_column_01);
01259     _verify(m_R_x_01);
01260     _verify(m_R_y_01);
01261     _verify(m_R_z_01);
01262     _verify(m_Cheb3D_01);
01263     _verify(m_EccAnom_01);
01264     _verify(m_Frac_01);
01265     _verify(m_MeanObliquity_01);
01266     _verify(m_Mjday_01);
01267     _verify(m_Mjday_TDB_01);
01268     _verify(m_Position_01);
01269     _verify(m_sign__01);
01270     _verify(m_timediff_01);
01271     _verify(m_AzElPa_01);
01272     _verify(m_IERS_01);
01273     _verify(m_Legendre_01);
01274     _verify(m_NutAngles_01);
01275     _verify(m_TimeUpdate_01);
01276     _verify(m_AccelHarmonic_01);
01277     _verify(m_EqnEquinox_01);
01278     _verify(m_JPL_Eph_DE430_01);
01279     _verify(m_LTC_01);
01280     _verify(m_NutMatrix_01);
01281     _verify(m_PoleMatrix_01);
01282     _verify(m_PrecMatrix_01);
01283     _verify(m_gmst_01);
01284     _verify(m_gast_01);
01285     _verify(m_MeasUpdate_01);
01286     _verify(m_G_AccelHarmonic_01);
01287     _verify(m_GHAMatrix_01);
01288     _verify(m_Accel_01);
01289     _verify(m_VarEqn_01);
01290     _verify(m_DEInteg_01);
01291
01292
01293     return 0;
01294 }
01295
01296
01297 int main()
01298 {
01299     AuxParamLoad();
01300     eop19620101();
01301     GGM03S();
01302     DE430Coeff();
01303     GEOS3();
01304     int result = all_tests();
01305
01306     if (result == 0)
01307         printf("PASSED\n");
01308
01309     printf("Tests run: %d\n", tests_run);
01310
01311     return (result != 0);
01312 }
```

Index