Astrodynamic Coordinates

This document describes several MATLAB functions that can be used for astrodynamic coordinate calculations and transformations. These functions can be used to calculate the following types of astrodynamic coordinates:

- Greenwich mean and apparent sidereal time
- Geodetic and geocentric coordinates
- Inertial position and velocity vectors
- Classical orbital elements
- Modified equinoctial orbital elements
- Hyperbolic and B-plane coordinates
- Flight path coordinates
- Topocentric coordinates
- Moon-centered coordinate transformations
- Mars-centered coordinate transformations

This software suite also contains a script called `csystems.m` that demonstrates how to interact with many of these coordinate routines. The following is a typical user interaction with this script.

The `csystems` script begins by displaying the following coordinate system menu.

```
coordinate system menu

 <1>  Greenwich apparent sidereal time

 <2>  classical orbital elements to eci state vector

 <3>  eci state vector to classical orbital elements

 <4>  flight path coordinates to eci state vector

 <5>  eci state vector to flight path coordinates

 <6>  classical orbital elements to modified equinoctial elements

 <7>  modified equinoctial elements to classical orbital elements

 <8>  geocentric coordinates to geodetic coordinates

 <9>  geodetic coordinates to geocentric coordinates

 <10> osculating orbital elements to mean elements

 <11> mean orbital elements to osculating elements
```

For most these menu items the user can elect to either input the data via the keyboard or use “internal” data already computed by the software. The data source menu appears as follows:

```
   data source menu

    <1>  user input

    <2>  internal data
```

The following example interaction demonstrates this “multiple calculations” feature. The user first calculates the Greenwich apparent sidereal time as follows:

```
   please input the calendar date
   (1 <= month <= 12, 1 <= day <= 31, year = all digits!)
   ? 7,1,1998

   please input the universal time
   (0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
   ? 10,20,30

   calendar date        01-Jul-1998

   universal time       10:20:30

   Julian date          2450995.9309

Greenwich apparent sidereal time  +04h 57m 34.8637s

      < please press any key to continue >
```

After this calculation is complete the script will ask the user for another selection with

```
   another selection (y = yes, n = no)
   ? y
```

A response of `y` for yes will cause the script to display the main menu again. An input of `n` to this response causes the program to terminate.

Suppose the “user” decides to convert classical orbital elements to an ECI state vector and elects the “user input” option as follows:

```
  data source menu

 <1>  user input

 <2>  internal data

? 1

please input the semimajor axis (kilometers)
(semimajor axis > 0)
? 8000

please input the orbital eccentricity (non-dimensional)
(0 <= eccentricity < 1)
? .015
```

```
please input the orbital inclination (degrees)
(0 <= inclination <= 180)
? 28.5

please input the argument of perigee (degrees)
(0 <= argument of perigee <= 360)
? 120

please input the right ascension of the ascending node (degrees)
(0 <= raan <= 360)
? 200

please input the true anomaly (degrees)
(0 <= true anomaly <= 360)
? 45
```

The following is the output created for this program option.

```
eci state vector

        rx (km)                ry (km)               rz (km)               rmag (km)
 +7.79924212852620e+003  +9.23029263046917e+002  +9.77393082400358e+002  +7.91425663201181e+003

        vx (kps)               vy (kps)              vz (kps)              vmag (kps)
 -2.62389316925579e-001  +6.33120225213611e+000  -3.27897813708178e+000  +7.13475071285258e+000


classical orbital elements

        sma (km)             eccentricity         inclination (deg)         argper (deg)
 +8.00000000000000e+003  +1.50000000000000e-002  +2.85000000000000e+001  +1.20000000000000e+002

       raan (deg)          true anomaly (deg)        arglat (deg)            period (min)
 +2.00000000000000e+002  +4.50000000000000e+001  +1.65000000000000e+002  +1.18684693788431e+002
```

GREENWICH SIDEREAL TIME

This section describes three MATLAB functions that be used to compute mean and apparent Greenwich sidereal time. Greenwich sidereal time is the angular location, measured along the fundamental plane (usually the equator), of the Greenwich meridian relative to the x-axis of the inertial coordinate system.

gast1.m – apparent Greenwich sidereal time - low precision

This function calculates the apparent Greenwich sidereal time using the first few terms of the IAU 1980 nutation algorithm.

The Greenwich apparent sidereal time is given by the expression

$$\theta = \theta_m + \Delta\psi \cos\left(\varepsilon_m + \Delta\varepsilon\right)$$

where θ_m is the Greenwich mean sidereal time, $\Delta\psi$ is the nutation in longitude, ε_m is the mean obliquity of the ecliptic and $\Delta\varepsilon$ is the nutation in obliquity.

The Greenwich mean sidereal time is calculated using the expression

$$\theta_m = 280.46061837 + 360.98564736629\left(JD - 2451545.0\right) + 0.000387933T^2 - T^3 / 38710000$$

where $T = (JD - 2451545)/36525$ and *JD* is the Julian date. The mean obliquity of the ecliptic is determined from

$$\varepsilon_m = 23^0 26' 21.''448 - 46.''8150T - 0.''00059T^2 + 0.''001813T^3$$

The nutation in obliquity and longitude involve the following three trigonometric arguments (in degrees):

$$L = 280.4665 + 36000.7698T$$

$$L' = 218.3165 + 481267.8813T$$

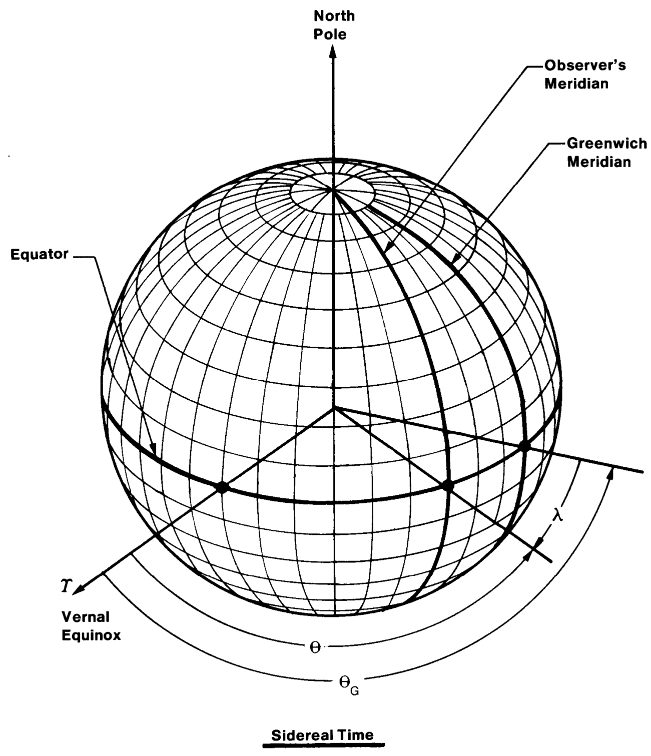
$$\Omega = 125.04452 - 1934.136261T$$

The calculation of nutation use the following two equations:

$$\Delta\psi = -17.20\sin\Omega - 1.32\sin 2L - 0.23\sin 2L' + 0.21\sin 2\Omega$$

$$\Delta\varepsilon = 9.20\cos\Omega + 0.57\cos 2L + 0.10\cos 2L' - 0.09\cos 2\Omega$$

These corrections are in units of arc seconds. The following diagram illustrates the geometry of sidereal time.



Sidereal Time

The syntax of this MATLAB function is

```
function gst = gast1 (jdate)

% Greenwich apparent sidereal time

% input

%  jdate = Julian date

% output

%  gst = Greenwich apparent sidereal time (radians)
%        (0 <= gst <= 2 pi)
```

gast2.m – Greenwich sidereal time - full precision

This function calculates mean or apparent Greenwich sidereal time. For the apparent sidereal time calculation, the obliquity in longitude and obliquity are determined using the `nod` function which implements the full IAU 1980 nutation algorithm. Please note that the Julian date can be passed to this routine in a high-order and low-order part. In the input argument list `k` determines the type of Greenwich sidereal time calculation. This function was ported to MATLAB using the Fortran version of the `NOVAS` code that was developed at the USNO.

For this numerical method, the nutation in longitude is determined from

$$\Delta\psi = \sum_{i=1}^{n} S_i \sin A_i$$

and the nutation in obliquity is determined from

$$\Delta\varepsilon = \sum_{i=1}^{n} C_i \cos A_i$$

where

$$A_i = a_i l + b_i l' + c_i F + d_i D + e_i \Omega$$

and l, l', F, D and Ω are fundamental arguments.

The syntax of this MATLAB function is

```
function gst = gast2 (tjdh, tjdl, k)

% this function computes the greenwich sidereal time
% (either mean or apparent) at julian date tjdh + tjdl

% input

%  tjdh = julian date, high-order part
%  tjdl = julian date, low-order part

%         julian date may be split at any point, but for
%         highest precision, set tjdh to be the integral part of
%         the julian date, and set tjdl to be the fractional part

%  k    = time selection code
%         set k=0 for greenwich mean sidereal time
%         set k=1 for greenwich apparent sidereal time

% output

%  gst = greenwich (mean or apparent) sidereal time in hours
```

gast3.m – Greenwich sidereal time – JPL binary ephmeris

This function calculates mean or apparent Greenwich sidereal time using the nutation terms available on a JPL binary ephemeris file. Please note that the Julian date can be passed to this routine in a high-order and low-order part. In the input argument list `k` determines the type of Greenwich sidereal time calculation.

The syntax of this MATLAB function is

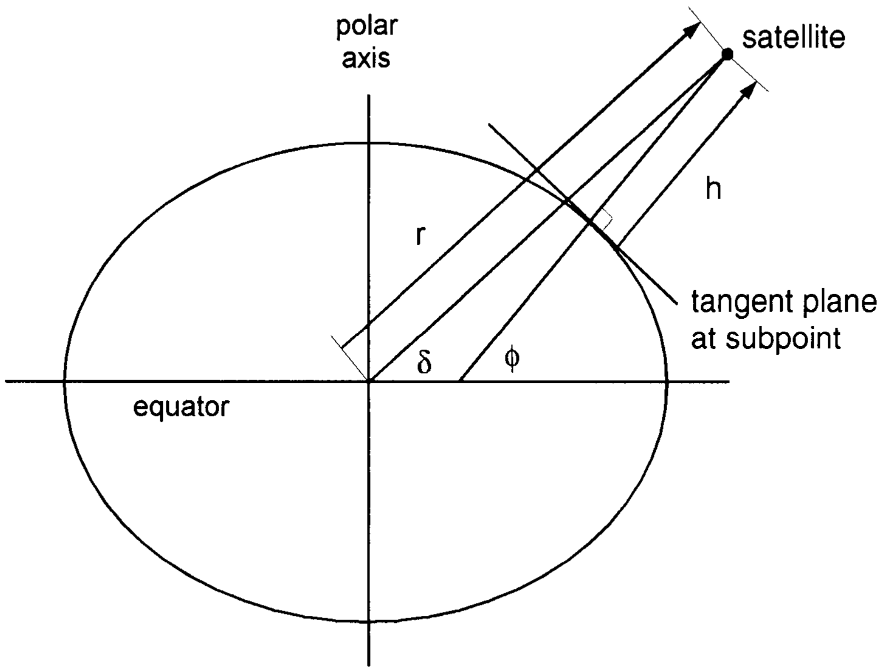
```
function gst = gast3 (tjdh, tjdl, k)

% this function computes the greenwich sidereal time
% (either mean or apparent) at julian date tjdh + tjdl

% nutation terms from a jpl binary ephemeris

% reference
% aoki, et al. (1982) astronomy and astropysics 105, 359-361

% input

%  tjdh = julian date, high-order part

%  tjdl = julian date, low-order part

%         julian date may be split at any point, but for
%         highest precision, set tjdh to be the integral part of
%         the julian date, and set tjdl to be the fractional part

%  k    = time selection code
%         set k = 0 for greenwich mean sidereal time
%         set k = 1 for greenwich apparent sidereal time

% output

%  gst = greenwich (mean or apparent) sidereal time in hours
```

GEODETIC AND GEOCENTRIC COORDINATES

This section describes several MATLAB functions that can be used to convert between geocentric and geodetic coordinates.

geodet1.m – convert geocentric coordinates to geodetic coordinates - series solution

This MATLAB function uses a series solution to convert geocentric distance and declination to geodetic altitude and latitude. The following diagram illustrates the geometric relationship between geocentric and geodetic coordinates.



In this diagram, δ is the geocentric declination, ϕ is the geodetic latitude, *r* is the geocentric distance, and *h* is the geodetic altitude. The exact mathematical relationship between geocentric and geodetic coordinates is given by the following system of two nonlinear equations

$$(c+h)\cos\phi - r\cos\delta = 0$$
$$(s+h)\sin\phi - r\sin\delta = 0$$

where the geodetic constants *c* and *s* are given by

$$c = \frac{r_{eq}}{\sqrt{1-\left(2f - f^2\right)\sin^2\phi}}$$
$$s = c\left(1-f\right)^2$$

and r_{eq} is the Earth equatorial radius (6378.14 kilometers) and f is the flattening factor for the Earth (1/298.257).

The geodetic latitude is determined using the following expression:

$$\phi = \delta + \left(\frac{\sin 2\delta}{\rho} \right) f + \left[\left(\frac{1}{\rho^2} - \frac{1}{4\rho} \right) \sin 4\delta \right] f^2$$

The geodetic altitude is calculated from

$$\hat{h} = \left(\hat{r} - 1 \right) + \left\{ \left(\frac{1 - \cos 2\delta}{2} \right) f + \left[\left(\frac{1}{4\rho} - \frac{1}{16} \right) \left(1 - \cos 4\delta \right) \right] f^2 \right\}$$

In these equations, ρ is the geocentric distance of the satellite, $\hat{h} = h / r_{eq}$ and $\hat{r} = \rho / r_{eq}$.

The syntax of this MATLAB function is

```
function [alt, lat] = geodet1 (rmag, dec)

% geodetic latitude and altitude

% series solution

% input

%  rmag = geocentric radius (kilometers)
%  dec  = geocentric declination (radians)
%         (+north, -south; -pi/2 <= dec <= +pi/2)

% output

%  alt = geodetic altitude (kilometers)
%  lat = geodetic latitude (radians)
%        (+north, -south; -pi/2 <= lat <= +pi/2)
```

geodet2.m – convert geocentric coordinates to geodetic coordinates – exact solution

This MATLAB function uses the solution of K. M. Borkowski (“Accurate Algorithms to Transform Geocentric to Geodetic Coordinates”, *Bullentin Geodesique*, 63 No. 1, 50-56) to convert geocentric distance and declination to geodetic altitude and latitude. The calculation steps for this non-iterative method are as follows:

$$E = \frac{\left[br_z - \left(a^2 - b^2 \right) \right]}{ar}$$

$$F = \frac{\left[br_z + \left(a^2 - b^2 \right) \right]}{ar}$$

$$r = \sqrt{r_x^2 + r_y^2}$$

where *a* is the equatorial radius of the Earth and b is determined from

$$b = \operatorname{sign}\left(r_z \right) \left(a - \frac{a}{f} \right)$$

The calculations continue with

$$P = \frac{4}{3}\left(EF + 1 \right)$$

$$Q = 2\left(E^2 - F^2 \right)$$

$$D = P^3 + Q^2$$

$$v = \left(D^{1/2} - Q \right)^{1/3}$$

$$G = \frac{1}{2}\left[\sqrt{E^2 + v + E} \right]$$

$$t = \sqrt{\left[G^2 + \frac{F - vG}{2G - E} \right]}$$

The geodetic latitude is determined from the expression

$$\phi = \tan^{-1}\left(a \frac{1 - t^2}{2bt} \right)$$

and the geodetic altitude is calculated from

$$h = \left(r - at \right)\cos\phi + \left(r_z - b \right)\sin\phi$$

The syntax of this MATLAB function is

```
function [xlat, alt] = geodet2(decl, rmag)

% geocentric to geodetic coordinates
% exact solution (Borkowski, 1989)

% input
%  decl = geocentric declination (radians)
%  rmag = geocentric distance (kilometers)

% output

%  xlat = geodetic latitude (radians)
%  alt  = geodetic altitude (kilometers)
```

geodet3.m – convert geodetic coordinates to ECF position vector

This function converts geodetic latitude, longitude and altitude to an Earth-centered-fixed (ECF) position vector. The components of this vector are given by

$$\mathbf{r}_{geocentric} = \begin{bmatrix} (N+h)\cos\phi\cos\lambda_e \\ (N+h)\cos\phi\sin\lambda_e \\ \left[N\left(1-e^2\right)+h\right]\sin\phi \end{bmatrix}$$

where

$$N = \frac{r_{eq}}{\sqrt{1-e^2\sin^2\phi}}$$

$$e^2 = 2f - f^2$$

$$f = \text{Earth flattening factor}$$

$$r_{eq} = \text{Earth equatorial radius}$$

$$\phi = \text{geodetic latitude}$$

$$\lambda_e = \text{east longitude}$$

$$h = \text{geodetic altitude}$$

The geocentric distance is determined from the components of the geocentric position vector as follows:

$$r = \sqrt{r_x^2 + r_y^2 + r_z^2}$$

The geocentric declination can be computed from the z component and the scalar magnitude of the geocentric position vector with

$$\delta = \sin^{-1}\left(\frac{r_z}{r}\right)$$

The syntax of this MATLAB function is

```
function r = geodet3 (lat, long, alt)

% ecf position vector
% input

%  lat  = geodetic latitude (radians)
%         (+north, -south; -pi/2 <= lat <= +pi/2)
%  long = geodetic longitude (radians)
%  alt  = geodetic altitude (kilometers)

% output

%  r = ecf position vector (kilometers)
```

geodet4.m – convert geodetic coordinates to geocentric coordinates

This MATLAB function converts geodetic latitude and altitude to geocentric position magnitude and geocentric declination.

The equations for this method are as follows:

$$\delta = \phi + \left(\frac{-\sin 2\phi}{\hat{h}+1} \right) f + \left\{ \frac{-\sin 2\phi}{2\left(\hat{h}+1\right)^2} + \left[\frac{1}{4\left(\hat{h}+1\right)^2} + \frac{1}{4\left(\hat{h}+1\right)} \right] \sin 4\phi \right\} f^2$$

and

$$\hat{\rho} = \left(\hat{h}+1\right) + \left(\frac{\cos 2\phi - 1}{2} \right) f + \left\{ \left[\frac{1}{4\left(\hat{h}+1\right)} + \frac{1}{16} \right] \left(1 - \cos 4\phi\right) \right\} f^2$$

where the geocentric distance r and geodetic altitude h have been normalized by $\hat{\rho} = r / r_{eq}$ and $\hat{h} = h / r_{eq}$, respectively, and r_{eq} is the equatorial radius of the Earth.

The syntax of this MATLAB function is

```
function [rmag, dec] = geodet4 (lat, alt)

% geodetic to geocentric coordinates

% input

%  lat  = geodetic latitude (radians)
%         (+north, -south; -pi/2 <= lat <= +pi/2)
%  alt  = geodetic altitude (kilometers)

% output

%  rmag = geocentric position magnitude (kilometers)
%  dec  = geocentric declination (radians)
%         (+north, -south; -pi/2 <= dec <= +pi/2)
```

geodet5.m – convert geocentric coordinates to geodetic coordinates – Sofair solution

This MATLAB function uses the solution of Isaac Sofair, "Improved Method for Calculating Exact Geodetic Latitude and Altitude", AIAA JGCD, Vol. 20, No. 4 and Vol. 23, No. 2, to convert geocentric distance and declination to geodetic altitude and latitude.

The syntax of this MATLAB function is

```
function [xalt, xlat] = geodet5(req, rpolar, rsc)

% convert geocentric eci position vector to
% geodetic altitude and latitude

% input

%  req    = equatorial radius (kilometers)
%  rpolar = polar radius (kilometers)
%  rsc    = spacecraft eci position vector (kilometers)

% output

%  xalt = geodetic altitude (kilometers)
%  xlat = geodetic latitude (radians)
%         (+north, -south; -pi/2 <= xlat <= +pi/2)

% reference

% "Improved Method for Calculating Exact Geodetic
% Latitude and Altitude", Isaac Sofair

% AIAA JGCD Vol. 20, No. 4 and Vol. 23, No. 2
```

This software suite contains a MATLAB script named `demo_geodet.m` that demonstrates how to interact with these useful functions. The following is the output created with this script.

```
geocentric declination      -19.38148629  degrees

geocentric radius          6497.69095120  kilometers


geodet1 function
================

geodetic latitude           -19.50000000  degrees

geodetic altitude           121.92000000  kilometers


geodet2 function
================

geodetic latitude           -19.50000099  degrees
```

```
   geodetic altitude           121.92003351  kilometers


   geodet5 function
   ================

   geodetic latitude           -19.50000099  degrees

   geodetic altitude           121.92003351  kilometers
```

triaxial.m – geodetic altitude to a triaxial ellipsoid

This MATLAB function calculates the geodetic altitude relative to a triaxial ellipsoidal planet. The algorithm is based on the numerical method described in "Geodetic Altitude to a Triaxial Ellipsoidal Planet", by Charles C. H. Tang, *The Journal of the Astronautical Sciences*, Vol. 36, No. 3, July-September 1988, pp. 279-283.

This function solves for the real root of the following nonlinear equation:

$$f\left(z_p\right)=\left\{1+\frac{c_y}{\left[c_z+\left(b^2-c^2\right)z_p\right]^2}+\frac{c_x}{\left[c_z+\left(a^2-c^2\right)z_p\right]^2}\right\}z_p^2-c^2=0$$

where

$$c_x=\left(acx_s\right)^2$$
$$c_y=\left(bcy_s\right)^2$$
$$c_z=c^2z_s$$
$$a,b,c=\text{ semi-axes }\left(a\ge b>c\right)$$
$$x_s,y_s,z_s=\text{ geocentric coordinates of satellite}$$
$$z_p=\text{ z coordinate of subpoint}$$

The bracketing interval used during the root-finding is $z_{p0}-10\le z_p\le z_{p0}+10$ (kilometers) where

$$z_{p0}=\frac{z_s}{\sqrt{\left(x_s/a\right)^2+\left(y_s/b\right)^2+\left(z_s/c\right)^2}}$$

The syntax of this MATLAB function is

```
   function alt = triaxial(rsc)

   % geodetic altitude relative
   % to a triaxial ellipsoid
```

```
% input

%  rsc = geocentric position vector (km)

% output

%  alt = geodetic altitude (km)
```

The semi-axes in this function are “hardwired” to the values $a = 6378.138$ and $b = 6367$ (kilometers), and the flattening factor is $f = 1/298.257$. These are representative values for the Earth and can be easily changed for other planets or the Moon.

This software suite contains a MATLAB script named `demo_triaxial.m` that demonstrates how to interact with this function. The following is the output created with this script.

```
program demo_triaxial

altitude relative to a triaxial ellipsoid
-----------------------------------------

altitude =   1519.73117837 kilometers

       sma (km)           eccentricity         inclination (deg)        argper (deg)
 +8.00000000000000e+003  +1.50000000000000e-002  +2.85000000000000e+001  +1.20000000000000e+002

      raan (deg)         true anomaly (deg)        arglat (deg)          period (min)
 +4.50000000000000e+001  +3.00000000000000e+001  +1.50000000000000e+002  +1.18684693004297e+002
```

geodesic.m – relative azimuths and distance between two ground sites

This MATLAB function determines the relative azimuths and distance between two ground sites on an oblate Earth. It is based on Sadano’s non-iterative solution.

The syntax of this MATLAB function is

```
function [azim12, azim21, slen] = geodesic (lat1, lat2, long1, long2)

% relative azimuths and distance between two ground sites

% input

%  lat1  = geodetic latitude of point 1 (radians)
%  long1 = east longitude of point 1 (radians)
%  lat2  = geodetic latitude of point 2 (radians)
%  long2 = east longitude of point 2 (radians)

% output

%  azim12 = azimuth from point 1 to 2 (radians)
%  azim21 = azimuth from point 2 to 1 (radians)
%  slen   = geodesic distance from point 1 to 2
%           (same units as req and rpolar)
```

gsite.m – ground site position vector

This MATLAB function calculates either the Earth-centered-inertial (ECI) or Earth-centered-fixed (ECF) position vector of a ground site located on an oblate Earth.

$$r_x = G_1 \cos\phi \cos\theta$$
$$r_y = G_1 \cos\phi \sin\theta$$
$$r_z = G_2 \sin\phi$$

For ECI site coordinates θ is equal to the local sidereal time, and for ECF coordinates θ is equal to the east longitude of the ground site. In these equations ϕ is the geodetic latitude of the ground site. It is positive in the northern hemisphere and negative in the southern hemisphere.
The geodetic constants G_1 and G_2 are calculated as follows:

$$G_1 = \frac{r_{eq}}{\sqrt{1-\left(2f - f^2\right)\sin^2\phi}} + h$$

$$G_2 = \frac{r_{eq}\left(1-f\right)^2}{\sqrt{1-\left(2f - f^2\right)\sin^2\phi}} + h$$

where

f = Earth flattening factor
r_{eq} = Earth equatorial radius
ϕ = geodetic latitude
h = geodetic altitude

The syntax of this MATLAB function is

```
function rsite = gsite (lat, alt, angle)

% ground site position vector

% input

%  lat   = geodetic latitude (radians)
%          (+north, -south; -pi/2 <= lat <= -pi/2)
%  alt   = geodetic altitude (meters)
%          (+ above sea level, - below sea level)
%  angle = local sidereal time or east longitude
%          (radians; 0 <= angle <= 2*pi)

% output

%  rsite = ground site position vector (kilometers)
```

crdr.m – crossrange and downrange distances

This MATLAB function calculates the crossrange and downrange distances of a flight vehicle relative to a *spherical* Earth.

The crossrange angle is determined from the following expression:

$$\sin \nu = -\sin\psi_1 \sin\phi_2 \cos\phi_1 \cos\Delta\lambda - \cos\psi_1 \cos\phi_1 \sin\Delta\lambda + \sin\psi_1 \cos\phi_2 \sin\phi_1$$

The downrange angle is determined from the following three equations:

$$\sin \mu = -\cos\psi_1 \sin\phi_2 \cos\phi_1 \cos\Delta\lambda + \sin\psi_1 \cos\phi_1 \sin\Delta\lambda + \cos\psi_1 \cos\phi_2 \sin\phi_1$$

$$\cos\mu = \cos\phi_2 \cos\phi_1 \cos\Delta\lambda + \sin\phi_2 \sin\phi_1$$

$$\mu = \tan^{-1}\left(\sin\mu, \cos\mu\right)$$

where

$\phi_1 =$ geocentric latitude of the initial point

$\psi_1 =$ flight azimuth at the initial point

$\phi_2 =$ geocentric latitude of the final point

$\Delta\lambda = \lambda_2 - \lambda_1$

$\lambda_1 =$ east longitude of the initial point

$\lambda_2 =$ east longitude of the final point

The crossrange distance d_c and downrange distance d_d are determined from

$$d_c = r_e \nu$$

$$d_d = r_e \mu$$

where r_e is the radius of the Earth. The flight azimuth is measured positive clockwise from north.

The syntax of this MATLAB function is

```
function [cr, dr] = crdr(lat1, long1, azim1, lat2, long2)

% spherical earth crossrange and downrange function

% input

%  lat1  = geocentric declination of point 1 (radians)
%  long1 = east longitude of point 1 (radians)
%  azim1 = flight azimuth of point 1 (radians)
%  lat2  = geocentric declination of point 2 (radians)
%  long2 = east longitude of point 2 (radians)

% output

%  cr = crossrange
%  dr = downrange

% NOTE: output has same unit as req
```

This software suite contains a MATLAB script named `demo_crdr.m` that demonstrates how to interact with this function. The following is the output created with this script.

```
program demo_crdr

crossrange and downrange
------------------------

crossrange               2428.49035140 kilometers

downrange                5655.25277128 kilometers


initial azimuth            90.00000000 degrees

initial declination         0.00000000 degrees

initial longitude           0.00000000 degrees


final declination         -21.81550000 degrees

final longitude            50.80200000 degrees
```

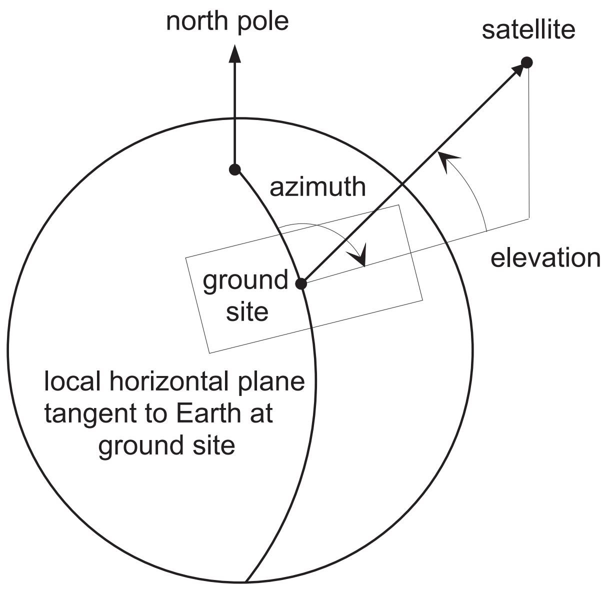
TOPOCENTRIC COORDINATES

This section describes MATLAB functions that can be used to convert between inertial and Topocentric coordinates.

eci2topo.m – ECI to topocentric coordinates

This MATLAB function calculates the topocentric coordinates (azimuth, elevation and slant range) and their rates with respect to an observer or ground site on an oblate Earth.

The following diagram illustrates the geometry of topocentric coordinates. Azimuth is measured positive clockwise from north and elevation is positive above the local horizontal plane.



The transformation of an ECI position vector $\mathbf{r}_{eci}$ to a topocentric position vector $\mathbf{r}_{topo}$ is given by the following vector-matrix operation

$$\mathbf{r}_{topo} = \left[\mathbf{T}\right]\mathbf{r}_{eci} = \begin{bmatrix} \sin\phi\cos\theta & \sin\phi\sin\theta & -\cos\phi \\ -\sin\theta & \cos\theta & 0 \\ \cos\phi\cos\theta & \cos\phi\sin\theta & \sin\phi \end{bmatrix}\mathbf{r}_{eci}$$

where ϕ is the geodetic latitude of the ground site and θ is the local sidereal time at the ground site. The local sidereal time of a ground site is given by

$$\theta = \theta_{g0} + \omega_e t + \lambda_e$$

where θ_{g0} is the Greenwich sidereal time at 0 hours universal time, ω_e is the inertial rotation rate of the Earth, *t* is the elapsed time since 0 hours universal time and λ_e is the east longitude of the ground site.

The ECI position vector used in this transformation is the position of the satellite relative to the observer or ground site. It is determined from the ECI position vectors of the observer $\mathbf{r}_{obs}$ and satellite $\mathbf{r}_{sat}$ according to

$$\mathbf{r}_{eci} = \mathbf{r}_{sat} - \mathbf{r}_{obs}$$

The scalar slant range from the observer to the satellite is computed from the components of this vector according to

$$p = \sqrt{x_{eci}^2 + y_{eci}^2 + z_{eci}^2}$$

The topocentric azimuth angle is calculated from the *x* and *y* components of the topocentric position vector using the following expression

$$A = \tan^{-1}\left(r_{y_{topo}}, -r_{x_{topo}}\right)$$

The topocentric elevation angle is calculated from the *z* component of the topocentric *unit* position vector with this next expression

$$E = \sin^{-1}\left(r_{z_{topo}}\right)$$

Azimuth is measured positive clockwise from north (90° is east, 180° is south, etc.) and elevation is positive above the local horizontal or tangent plane at the observer's geographic location or ground site.

The ECI range-rate vector $\dot{\mathbf{p}}_{eci}$ of the satellite relative to the observer is determined from

$$\dot{\mathbf{p}}_{eci} = \mathbf{v}_{sat} - \mathbf{w} \times \mathbf{r}_{sat}$$

where

$$\mathbf{w} = \omega_e \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

is the inertial rotation vector of the Earth and $\mathbf{v}_{sat}$ is the ECI velocity vector of the satellite.

The topocentric range-rate vector is computed from the transformation

$$\dot{\mathbf{p}}_{topo} = \left[\mathbf{T}\right]\dot{\mathbf{p}}_{eci}$$

The derivative of slant range or range-rate is given by

$$\dot{p} = \frac{\mathbf{r}_{topo} \bullet \dot{\mathbf{p}}_{topo}}{p}$$

The azimuth and elevation rates are determined from the *x*, *y* and *z* components of the topocentric range and range-rate vectors as follows

$$\dot{A} = \frac{\dot{p}_{topo_x} p_{topo_y} - \dot{p}_{topo_y} p_{topo_x}}{p_x^2 + p_y^2}$$

$$\dot{E} = \frac{\dot{p}_{topo_z} - \dot{p}\sin E}{\sqrt{p_x^2 + p_y^2}}$$

The syntax of this MATLAB function is

```
function dvec = eci2topo (rsat, vsat, gast, obslat, obslong, obsalt)

% convert eci position and velocity vectors
% to topocentric coordinates and rates

% input

%  rsat    = satellite eci position vector (kilometers)
%  vsat    = satellite eci velocity vector (kilometers/second)
%  gast    = Greenwich apparent sidereal time (radians)
%  obslat  = ground site geodetic latitude (radians)
%  obslong = ground site longitude (radians)
%  obsalt  = ground site altitude (meters)

% output

%  dvec(1) = topocentric azimuth (radians)
%  dvec(2) = topocentric elevation (radians)
%  dvec(3) = topocentric slant range (kilometers)
%  dvec(4) = azimuth rate (radians/second)
%  dvec(5) = elevation rate (radians/second)
%  dvec(6) = slant range rate (kilometers/second)
```

topo2eci.m – topocentric to ECI coordinates

This function converts the topocentric coordinates of a satellite measured with respect to a ground site or observer on an oblate Earth to the corresponding ECI position and velocity vectors.

The topocentric radius vector of the satellite is

$$\mathbf{r}_{topo} = \begin{Bmatrix} -p\cos E\cos A \\ p\cos E\sin A \\ p\sin E \end{Bmatrix}$$

and the topocentric velocity vector of the satellite is

$$\mathbf{v}_{topo} = \begin{Bmatrix} -\dot{p}\cos E\cos A + p\sin E\cos A\dot{E} + p\cos E\sin A\dot{A} \\ \dot{p}\cos E\sin A - p\sin E\sin A\dot{E} + p\cos E\cos A\dot{A} \\ \dot{p}\sin E + p\cos E\,\dot{E} \end{Bmatrix}$$

The transformation of the Earth-fixed topocentric position vector to an ECI topocentric position vector is computed from the transpose of the $[T]$ matrix given above. This matrix-vector operation is as follows:

$$\mathbf{r} = \left[\mathbf{T}\right]^T \mathbf{r}_{topo} = \begin{bmatrix} \sin\phi\cos\theta & -\sin\theta & \cos\phi\cos\theta \\ \sin\phi\sin\theta & \cos\theta & \cos\phi\sin\theta \\ -\cos\phi & 0 & \sin\phi \end{bmatrix} \mathbf{r}_{topo}$$

The ECI position vector of the satellite is given by

$$\mathbf{r}_{eci} = \mathbf{r} + \mathbf{r}_{site}$$

where $\mathbf{r}_{site}$ is the ECI position vector of the observer or ground site. Likewise, the ECI topocentric velocity vector of the satellite is

$$\mathbf{v} = \left[\mathbf{T}\right]^T \mathbf{v}_{topo} = \begin{bmatrix} \sin\phi\cos\theta & -\sin\theta & \cos\phi\cos\theta \\ \sin\phi\sin\theta & \cos\theta & \cos\phi\sin\theta \\ -\cos\phi & 0 & \sin\phi \end{bmatrix} \mathbf{v}_{topo}$$

The ECI velocity vector of the satellite is

$$\mathbf{v}_{eci} = \mathbf{v} + \mathbf{w} \times \mathbf{r}_{eci}$$

where the inertial rotation vector of the Earth is given by

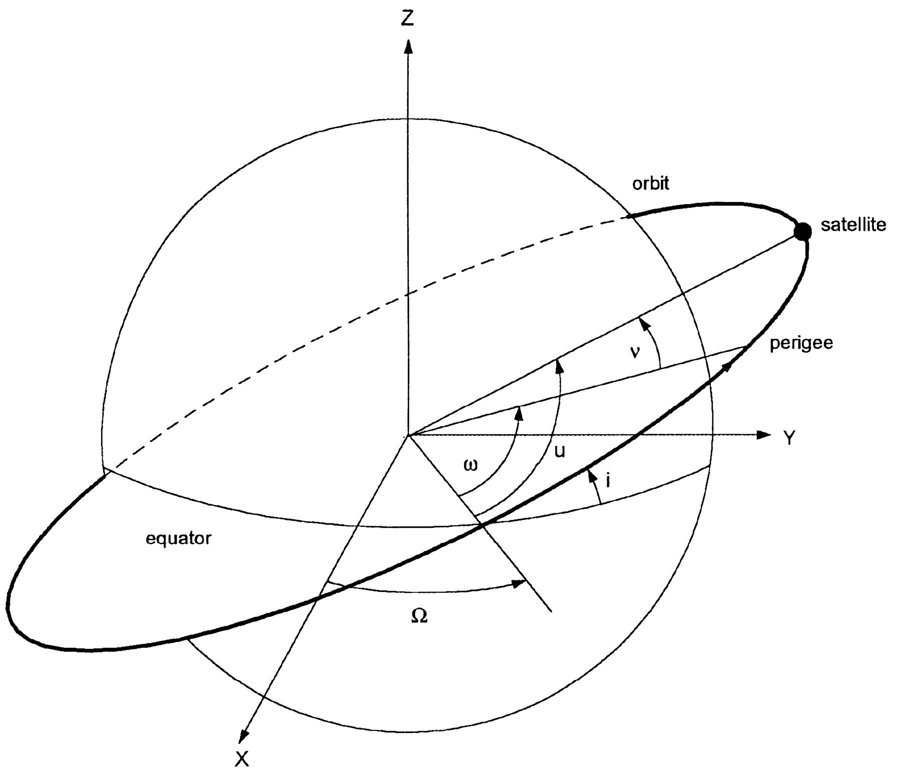
$$\mathbf{w} = \omega_e \left[0,0,1\right]^T$$

The syntax of this MATLAB function is

```
function [rsat, vsat] = topo2eci (gast, obslat, obslong, obsalt, ...
                                  azim, elev, srange, azdot, eldot, srdot)

% convert topocentric coordinates to eci
% position and velocity vectors

% input

%  gast    = Greenwich apparent sidereal time (radians)
%  obslat  = ground site geodetic latitude (radians)
%  obslong = ground site longitude (radians)
%  obsalt  = ground site altitude (meters)
%  azim    = topocentric azimuth (radians)
%  elev    = topocentric elevation (radians)
%  srange  = topocentric slant range (kilometers)
%  azdot   = azimuth rate (radians/second)
%  eldot   = elevation rate (radians/second)
%  srdot   = slant range rate (kilometers/second)

% output

%  rsat = satellite eci position vector (kilometers)
%  vsat = satellite eci velocity vector (kilometers/second)
```

CLASSICAL ORBITAL ELEMENTS

This section describes MATLAB functions that can be used to convert between inertial state vectors and classical orbital elements. The following diagram illustrates the geometry of these orbital elements.



where

$a =$ semimajor axis

$e =$ orbital eccentricity

$i =$ orbital inclination

$\omega =$ argument of perigee

$\Omega =$ right ascension of the ascending node

$\theta =$ true anomaly

The semimajor axis defines the size of the orbit and the orbital eccentricity defines the shape of the orbit. The angular orbital elements are defined with respect to a fundamental x-axis, the vernal equinox, and a fundamental plane, the equator. The z-axis of this system is collinear with the spin axis of the Earth, and the y-axis completes a right-handed coordinate system.

The orbital inclination is the angle between the equatorial plane and the orbit plane. Satellite orbits with inclinations between 0 and 90 degrees are called *direct* orbits and satellites with inclinations greater than 90 and less than 180 degrees are called *retrograde* orbits. The right ascension of the ascending node (RAAN) is the angle measured from the x-axis (vernal equinox) eastward along the equator to the ascending node. The argument is the angle from the ascending node, measured along the orbit plane in the direction of increasing true anomaly, to the argument of perigee. The true anomaly is the angle from the argument of perigee, measured along the orbit plane in the direction of motion, to the satellite's location. Also shown is the argument of latitude, *u*, which is the angle from the ascending node, measured in the orbit plane, to the satellite's location in the orbit. It is equal to $u = \nu + \omega$.

The orbital eccentricity is an indication of the type of orbit. For values of $0 \le e < 1$, the orbit is circular or elliptic. The orbit is parabolic when $e = 1$ and the orbit is hyperbolic if $e > 1$

The semimajor axis is calculated using the following expression:

$$a = \frac{1}{\frac{2}{r} - \frac{v^2}{\mu}}$$

where $r = |\mathbf{r}| = \sqrt{r_x^2 + r_y^2 + r_z^2}$ and $v = |\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$. The angular orbital elements are calculated from modified equinoctial orbital elements which are calculated from the ECI position and velocity vectors.

The scalar orbital eccentricity is determined from *h* and *k* as follows:

$$e = \sqrt{h^2 + k^2}$$

The orbital inclination is determined from *p* and *q* using the following expression

$$i = 2\tan^{-1}\left(\sqrt{p^2 + q^2}\right)$$

For values of inclination greater than a small value ε , the right ascension of the ascending node (RAAN) is given by

$$\Omega = \tan^{-1}(p, q)$$

Otherwise, the orbit is equatorial and there is no RAAN. If the value of orbital eccentricity is greater than ε , the argument of perigee is determined from

$$\omega = \tan^{-1}(h, k) - \Omega$$

Otherwise, the orbit is circular and there is no argument of perigee. In the MATLAB code for these calculations, $\varepsilon = 10^{-8}$.

Finally, the true anomaly is found from the expression

$$\theta = \lambda - \Omega - \omega$$

In these equations, all two argument inverse tangent calculations use the four quadrant MATLAB function `atan3.m` included with this software suite.

eci2orb1.m – convert ECI state vector to six classical orbital elements

This MATLAB function converts an ECI state vector (position and velocity vectors) to six classical orbital elements.

The syntax of this MATLAB function is

```
function oev = eci2orb1 (mu, r, v)

% convert eci state vector to six
% classical orbital elements via
% equinoctial elements

% input

%  mu = central body gravitational constant (km**3/sec**2)
%  r  = eci position vector (kilometers)
%  v  = eci velocity vector (kilometers/second)

% output

%  oev(1)  = semimajor axis (kilometers)
%  oev(2)  = orbital eccentricity (non-dimensional)
%            (0 <= eccentricity < 1)
%  oev(3)  = orbital inclination (radians)
%            (0 <= inclination <= pi)
%  oev(4)  = argument of perigee (radians)
%            (0 <= argument of perigee <= 2 pi)
%  oev(5)  = right ascension of ascending node (radians)
%            (0 <= raan <= 2 pi)
%  oev(6)  = true anomaly (radians)
%            (0 <= true anomaly <= 2 pi)
```

eci2orb2.m – convert ECI state vector to complete set of classical orbital elements

This MATLAB function converts an ECI state vector (position and velocity vectors) to a complete set of classical orbital elements. In additional to the six classical elements, this routine also calculates the following items:

- orbital period
- argument of latitude
- east longitude of ascending node
- specific orbital energy
- flight path angle

- right ascension
- declination
- geodetic latitude of subpoint
- east longitude of subpoint
- geodetic altitude
- geocentric radius of perigee
- geocentric radius of apogee
- perigee velocity
- apogee velocity
- geodetic altitude of perigee
- geodetic altitude of apogee
- geodetic latitude of perigee
- geodetic latitude of apogee
- mean motion
- mean anomaly
- eccentric anomaly
- velocity magnitude
- time until periapsis passage
- time until ascending node crossing
- nodal period
- anomalistic period

The right ascension of a satellite is determined from the x and y components of the ECI *unit* position vector as follows:

$$\alpha = \tan^{-1}\left(r_{y_{eci}}, r_{x_{eci}}\right)$$

The subpoint of a satellite is the point on the Earth's surface directly below the satellite. The locus of subpoints describes the ground track of a satellite. The east longitude of the subpoint of a satellite can be determined from the x and y components of the ECF *unit* position vector as follows:

$$\lambda = \tan^{-1}\left(r_{y_{ecf}}, r_{x_{ecf}}\right)$$

These two position components can be determined from the x and y components of the ECI position vector and the Greenwich sidereal time θ_g as follows:

$$r_{x_{ecf}} = r_{x_{eci}} \cos\theta_g + r_{y_{eci}} \sin\theta_g$$
$$r_{y_{ecf}} = r_{y_{eci}} \cos\theta_g - r_{x_{eci}} \sin\theta_g$$

The geocentric declination of the satellite subpoint is

$$\delta = \sin^{-1}\left(\frac{r_z}{r}\right)$$

The geocentric declination of perigee and apogee are given by the next two equations:

$$\delta_p = \sin^{-1}\left(\sin i \sin \omega\right)$$
$$\delta_a = \sin^{-1}\left[\sin i \sin\left(180 + \omega\right)\right]$$

If the orbit is elliptic $\left(a > 0\right)$, the geocentric radius of perigee and apogee are given by the next two equations:

$$r_p = a\left(1 - e\right)$$

$$r_a = a\left(1 + e\right)$$

The geodetic latitude and altitude of perigee and apogee can be determined from these geocentric coordinates using one of the geodetic conversion functions described elsewhere in this document.

The perigee and apogee speed can be determined from

$$V_p = \sqrt{2\mu \frac{r_a}{r_p\left(r_p + r_a\right)}}$$

$$V_a = \sqrt{2\mu \frac{r_p}{r_a\left(r_p + r_a\right)}}$$

If the orbit is hyperbolic $\left(a < 0\right)$, there is no apogee and the perigee radius is equal to

$$r_p = a\left(1 + e\right)$$

and the perigee speed is equal to

$$V_p = \sqrt{\frac{2\mu}{r_p} + \frac{\mu}{a}}$$

The flight path angle is the angle between the velocity vector and the local horizontal or tangent plane at the satellite’s location. It can be calculated with the following expression:

$$\gamma = \sin^{-1}\left(\frac{\mathbf{r} \bullet \mathbf{v}}{\left|\mathbf{r} \bullet \mathbf{v}\right|}\right)$$

The specific orbital energy, or the energy per unit mass is determined from this next expression:

$$E = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a}$$

The orbital energy is the sum of a satellite's kinetic and potential energy. Notice that it depends only on the satellite's semimajor axis and the gravitational constant of the central body.

The Keplerian or unperturbed orbital period is

$$\tau = 2\pi\sqrt{\frac{a^3}{\mu}}$$

The nodal and anomalistic periods are computed using the `tperiod.m` function. The times relative to perigee passage and ascending node crossing are computed with the `tevent.m` function. Please see the Programmer's Toolbox documentation for additional information about these two MATLAB functions.

The relationship between true anomaly θ and eccentric anomaly E for an elliptic orbit of eccentricity e is given by

$$\sin E = \sin\theta\sqrt{1-e^2}$$

$$\cos E = e + \cos\theta$$

$$E = \tan^{-1}\left(\sin E, \cos E\right)$$

The syntax of this MATLAB function is

```
function oev = eci2orb2 (mu, gst0, omega, ut, r, v)

% convert eci state vector to complete
% set of classical orbital elements

% input

%  mu    = Earth gravitational constant (km^2/sec^3)
%  gst0  = Greenwich sidereal time at 0 hours ut (radians)
%  omega = Earth sidereal rotation rate (radians/second)
%  ut    = universal time (seconds)
%          (0 <= ut <= 86400)
%  r  = eci position vector (kilometers)
%  v  = eci velocity vector (kilometers/second)

% output

%  oev(1)  = semimajor axis (kilometers)
%  oev(2)  = orbital eccentricity (non-dimensional)
%            (0 <= eccentricity < 1)
%  oev(3)  = orbital inclination (radians)
%            (0 <= inclination <= pi)
%  oev(4)  = argument of perigee (radians)
%            (0 <= argument of perigee <= 2 pi)
%  oev(5)  = right ascension of ascending node (radians)
%            (0 <= raan <= 2 pi)
%  oev(6)  = true anomaly (radians)
%            (0 <= true anomaly <= 2 pi)
%  oev(7)  = orbital period (seconds)
%  oev(8)  = argument of latitude (radians)
%            (0 <= argument of latitude <= 2 pi)
%  oev(9)  = east longitude of ascending node (radians)
%            (0 <= east longitude <= 2 pi)
%  oev(10) = specific orbital energy (kilometer^2/second^2)
%  oev(11) = flight path angle (radians)
%            (-0.5 pi <= fpa <= 0.5 pi)
%  oev(12) = right ascension (radians)
%            (-2 pi <= right ascension <= 2 pi)
%  oev(13) = declination (radians)
%            (-0.5 pi <= declination <= 0.5 pi)
%  oev(14) = geodetic latitude of subpoint (radians)
%            (-0.5 pi <= latitude <= 0.5 pi)
%  oev(15) = east longitude of subpoint (radians)
%            (-2 pi <= latitude <= 2 pi)
%  oev(16) = geodetic altitude (kilometers)
%  oev(17) = geocentric radius of perigee (kilometers)
%  oev(18) = geocentric radius of apogee (kilometers)
%  oev(19) = perigee velocity (kilometers/second)
%  oev(20) = apogee velocity (kilometers/second)
%  oev(21) = geodetic altitude of perigee (kilometers)
%  oev(22) = geodetic altitude of apogee (kilometers)
%  oev(23) = geodetic latitude of perigee (radians)
%            (-0.5 pi <= latitude <= 0.5 pi)
%  oev(24) = geodetic latitude of apogee (radians)
%            (-0.5 pi <= latitude <= 0.5 pi)
%  oev(25) = mean motion (radians/second)
%  oev(26) = mean anomaly (radians)
%            (-2 pi <= mean anomaly <= 2 pi)
%  oev(27) = eccentric anomaly (radians)
%            (-2 pi <= eccentric anomaly <= 2 pi)
%  oev(28) = velocity magnitude (kilometers/second)
%  oev(29) = time until periapsis passage
%  oev(30) = time until ascending node crossing
%  oev(31) = nodal period
%  oev(32) = anomalistic period
```

orb2eci.m – convert classical orbital elements to ECI state vector

This MATLAB function converts the six classical orbital elements to an ECI state vector (position and velocity vectors). The rectangular components of the satellite's inertial position vector are determined from

$$r_x = r\left[\cos\Omega\cos(\omega+\theta) - \sin\Omega\cos i\sin(\omega+\theta)\right]$$

$$r_y = r\left[\sin\Omega\cos(\omega+\theta) + \cos\Omega\cos i\sin(\omega+\theta)\right]$$

$$r_z = r\sin i\sin(\omega+\theta)$$

where $r = \sqrt{r_x^2 + r_y^2 + r_z^2}$.

The components of the inertial velocity vector are computed using the following equations:

$$v_x = -\sqrt{\frac{\mu}{p}}\left[\cos\Omega\left\{\sin\left(\omega+\theta\right)+e\sin\omega\right\}+\sin\Omega\cos i\left\{\cos\left(\omega+\theta\right)+e\cos\omega\right\}\right]$$

$$v_y = -\sqrt{\frac{\mu}{p}}\left[\sin\Omega\left\{\sin\left(\omega+\theta\right)+e\sin\omega\right\}-\cos\Omega\cos i\left\{\cos\left(\omega+\theta\right)+e\cos\omega\right\}\right]$$

$$v_z = \sqrt{\frac{\mu}{p}}\left[\sin i\left\{\cos\left(\omega+\theta\right)+e\cos\omega\right\}\right]$$

where

$a =$ semimajor axis

$e =$ orbital eccentricity

$i =$ orbital inclination

$\omega =$ argument of perigee

$\Omega =$ right ascension of the ascending node

$\theta =$ true anomaly

$p = a\left(1-e^2\right)$

The syntax of this MATLAB function is

```
function [r, v] = orb2eci(mu, oev)

% convert classical orbital elements to eci state vector

% input

%  mu     = gravitational constant (km**3/sec**2)
%  oev(1) = semimajor axis (kilometers)
%  oev(2) = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
%  oev(3) = orbital inclination (radians)
%           (0 <= inclination <= pi)
%  oev(4) = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
%  oev(5) = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
%  oev(6) = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)

% output

%  r = eci position vector (kilometers)
%  v = eci velocity vector (kilometers/second)
```

MODIFIED EQUINOCTIAL ORBITAL ELEMENTS

The modified equinoctial orbital elements are a set of orbital elements that are useful for trajectory analysis and optimization. They are valid for circular, elliptic, and hyperbolic orbits. These *direct* modified equinoctial equations exhibit no singularity for zero eccentricity and orbital inclinations equal to 0 and 90 degrees. However, two of the components are singular for an orbital inclination of 180 degrees.

mee2coe.m – convert modified equinoctial orbital elements to classical orbital elements

This MATLAB function can be used to convert modified equinoctial orbital elements to classical orbital elements.

The syntax of this MATLAB function is

```
function coev = mee2coe(mee)

% convert modified equinoctial elements
% to classical orbit elements

% input

%  mee(1) = semilatus rectum of orbit (kilometers)
%  mee(2) = f equinoctial element
%  mee(3) = g equinoctial element
%  mee(4) = h equinoctial element
%  mee(5) = k equinoctial element
%  mee(6) = true longitude (radians)

% output

%  coev(1) = semimajor axis (kilometers)
%  coev(2) = eccentricity
%  coev(3) = inclination (radians)
%  coev(4) = argument of periapsis (radians)
%  coev(5) = right ascension of ascending node (radians)
%  coev(6) = true anomaly (radians)
```

The relationship between modified equinoctial and classical orbital elements is given by

$$p = a\left(1-e^2\right)$$
$$f = e\cos\left(\omega+\Omega\right)$$
$$g = e\sin\left(\omega+\Omega\right)$$
$$h = \tan\left(i/2\right)\cos\Omega$$
$$k = \tan\left(i/2\right)\sin\Omega$$
$$L = \Omega+\omega+\theta$$

where

$p =$	semiparameter
$a =$	semimajor axis
$e =$	orbital eccentricity
$i =$	orbital inclination
$\omega =$	argument of perigee
$\Omega =$	right ascension of the ascending node
$\theta =$	true anomaly
$L =$	true longitude

The relationship between classical and modified equinoctial orbital elements is as follows:

semimajor axis

$$a = \frac{p}{1 - f^2 - g^2}$$

orbital eccentricity

$$e = \sqrt{f^2 + g^2}$$

orbital inclination

$$i = 2\tan^{-1}\left(\sqrt{h^2 + k^2}\right)$$

argument of periapsis

$$\omega = \tan^{-1}(g/f) - \tan^{-1}(k/h)$$

right ascension of the ascending node

$$\Omega = \tan^{-1}(k/h)$$

true anomaly

$$\theta = L - (\Omega + \omega) = L - \tan^{-1}(g/f)$$

mee2eci.m – convert modified equinoctial orbital elements to ECI state vector

This MATLAB function can be used to convert modified equinoctial orbital elements to ECI position and velocity vectors.

The syntax of this MATLAB function is

```
function [r, v] = mee2eci(mu, mee)
```

```
% convert modified equinoctial orbital
% elements to eci position and velocity vectors

% input

%  mu     = gravitational constant (km**3/sec**2)
%  mee(1) = semilatus rectum of orbit (kilometers)
%  mee(2) = f equinoctial element
%  mee(3) = g equinoctial element
%  mee(4) = h equinoctial element
%  mee(5) = k equinoctial element
%  mee(6) = true longitude (radians)

% output

%  r = eci position vector (kilometers)
%  v = eci velocity vector (kilometers/second)
```

The relationship between ECI position and velocity vectors and modified equinoctial elements is defined by the following series of equations:

position vector

$$\mathbf{r} = \begin{bmatrix} \dfrac{r}{s^2}\left(\cos L + \alpha^2 \cos L + 2hk \sin L\right) \\ \dfrac{r}{s^2}\left(\sin L - \alpha^2 \sin L + 2hk \cos L\right) \\ \dfrac{r}{s^2}\left(h \sin L - k \cos L\right) \end{bmatrix}$$

velocity vector

$$\mathbf{v} = \begin{bmatrix} -\dfrac{1}{s^2}\sqrt{\dfrac{\mu}{p}}\left(\sin L + \alpha^2 \sin L - 2hk \cos L + g - 2fhk + \alpha^2 g\right) \\ -\dfrac{1}{s^2}\sqrt{\dfrac{\mu}{p}}\left(-\cos L + \alpha^2 \cos L + 2hk \sin L + g - f + 2ghk + \alpha^2 f\right) \\ \dfrac{2}{s^2}\sqrt{\dfrac{\mu}{p}}\left(h \cos L + k \sin L + fh + gk\right) \end{bmatrix}$$

where

$$\alpha^2 = h^2 - k^2 \qquad s^2 = 1 + h^2 + k^2$$

$$r = \frac{p}{w} \qquad w = 1 + f \cos L + g \sin L$$

eci2mee.m – convert ECI state vector to modified equinoctial elements

This MATLAB function can be used to convert an ECI state vector (position and velocity vectors) to modified equinoctial orbital elements.

The syntax of this MATLAB function is

```
function mee = eci2mee(mu, reci, veci)

% convert eci state vector to
% modified equinoctial elements

% input

%  mu   = gravitational constant (km**3/sec**2)
%  reci = eci position vector (kilometers)
%  veci = eci velocity vector (kilometers/second)

% output

%  mee(1) = semiparameter of orbit (kilometers)
%  mee(2) = f equinoctial element
%  mee(3) = g equinoctial element
%  mee(4) = h equinoctial element
%  mee(5) = k equinoctial element
%  mee(6) = true longitude (radians)
```

The conversion of ECI position and velocity vectors **r** and **v** to modified equinoctial elements is defined by the following series of equations:

$$p = h^2 / \mu$$

$$k = \hat{\mathbf{h}}_x / 1 + \hat{\mathbf{h}}_z$$

$$h = -\hat{\mathbf{h}}_y / 1 + \hat{\mathbf{h}}_z$$

where $\hat{\mathbf{h}}_x$ $\hat{\mathbf{h}}_y$, and $\hat{\mathbf{h}}_z$ are the x, y, and z components of the unit angular momentum vector.

$$f = \mathbf{e} \bullet \tilde{\mathbf{f}}$$

$$g = \mathbf{e} \bullet \tilde{\mathbf{g}}$$

The eccentricity vector $\mathbf{e}$ points in the direction of perigee and is computed from

$$\mathbf{e} = \frac{\mathbf{v} \times \mathbf{h}}{\mu} - \frac{\mathbf{r}}{|\mathbf{r}|}$$

$$\tilde{\mathbf{f}} = \hat{\mathbf{f}} \big/ 1 + k^2 + h^2$$

$$\tilde{\mathbf{g}} = \hat{\mathbf{g}} \big/ 1 + k^2 + h^2$$

$$\hat{\mathbf{f}} = \begin{Bmatrix} 1 - k^2 + h^2 \\ 2kh \\ 2k \end{Bmatrix}$$

$$\hat{\mathbf{g}} = \begin{Bmatrix} 2kh \\ 1 + k^2 - h^2 \\ 2h \end{Bmatrix}$$

The angular momentum vector is computed from

$$\mathbf{h} = \mathbf{r} \times \mathbf{v}$$

where **r** and **v** are the inertial position and velocity vectors, respectively.

Finally, the true longitude is determined from

$$L = \tan^{-1}\left(\hat{\mathbf{r}}_x + \hat{\mathbf{v}}_y, \hat{\mathbf{r}}_y - \hat{\mathbf{v}}_x\right)$$

where $\hat{\mathbf{r}}_x \;\; \hat{\mathbf{r}}_y$, and $\hat{\mathbf{v}}_x \;\; \hat{\mathbf{v}}_y$ are the x and y components of the unit position and velocity vectors.

ueci2umee.m – convert ECI unit vector to unit vector in modified equinoctial element system

This MATLAB function can be used to convert an ECI unit vector to a unit vector in the modified equinoctial orbital element coordinate system.

The syntax of this MATLAB function is

```
function umee = ueci2umee(reci, veci, ueci)

% convect eci unit vector to mee unit vector

% input

%  reci = eci position vector (kilometers)
%  veci = eci velocity vector (kilometers/second)
%  ueci = eci unit vector

% output

%  umee = unit vector in modified equinoctial frame
```

The relationship between a unit vector in the ECI coordinate system $\hat{\mathbf{u}}_{ECI}$ and the corresponding unit vector in the modified equinoctial system $\hat{\mathbf{u}}_{MEE}$ is given by

$$\hat{\mathbf{u}}_{ECI} = \begin{bmatrix} \hat{\mathbf{i}}_r & \hat{\mathbf{i}}_t & \hat{\mathbf{i}}_n \end{bmatrix} \hat{\mathbf{u}}_{MEE}$$

where

$$\hat{\mathbf{i}}_r = \frac{\mathbf{r}}{|\mathbf{r}|} \qquad \hat{\mathbf{i}}_n = \frac{\mathbf{r} \times \mathbf{v}}{|\mathbf{r} \times \mathbf{v}|} \qquad \hat{\mathbf{i}}_t = \hat{\mathbf{i}}_n \times \hat{\mathbf{i}}_r = \frac{(\mathbf{r} \times \mathbf{v}) \times \mathbf{r}}{|\mathbf{r} \times \mathbf{v}||\mathbf{r}|}$$

This relationship can also be expressed as

$$\hat{\mathbf{u}}_{ECI} = [Q]\hat{\mathbf{u}}_{MEE} = \begin{bmatrix} \hat{\mathbf{r}}_x & (\hat{\mathbf{h}} \times \hat{\mathbf{r}})_x & \hat{\mathbf{h}}_x \\ \hat{\mathbf{r}}_y & (\hat{\mathbf{h}} \times \hat{\mathbf{r}})_y & \hat{\mathbf{h}}_y \\ \hat{\mathbf{r}}_z & (\hat{\mathbf{h}} \times \hat{\mathbf{r}})_z & \hat{\mathbf{h}}_z \end{bmatrix} \hat{\mathbf{u}}_{MEE}$$

where **r** and **v** are the ECI position and velocity vectors.

FLIGHT PATH COORDINATES

This section describes several MATLAB functions that can be used to convert between relative (Earth-fixed) flight path coordinates and inertial position and velocity vectors. The relative flight path coordinates used in these functions are

$r =$ geocentric radius
$V =$ speed
$\gamma =$ flight path angle
$\delta =$ geocentric declination
$\lambda =$ geographic longitude $(+\text{ east})$
$\psi =$ flight azimuth $(+\text{ clockwise from north})$

Please note the sign and direction convention.

The following are several useful equations that summarize the relationships between inertial and relative flight path coordinates.

$$v_r \sin\gamma_r = v_i \sin\gamma_i$$

$$v_r \cos\gamma_r \cos\psi_r = v_i \cos\gamma_i \cos\psi_i$$

$$v_r \cos\gamma_r \sin\psi_r + \omega_e r \cos\delta = v_i \cos\gamma_i \sin\psi_i$$

where the *r* subscript denotes relative coordinates and the *i* subscript inertial coordinates.

The inertial speed can also be computed from the following expression

$$v_i = \sqrt{v^2 + 2v\,r\omega\cos\gamma\sin\psi\cos\delta + r^2\omega^2\cos^2\delta}$$

The inertial flight path angle can be computed from

$$\cos\gamma_i = \sqrt{\frac{v^2\cos^2\gamma + 2vr\omega\cos\gamma\cos\psi\cos\delta + r^2\omega^2\cos^2\delta}{v^2 + 2vr\omega\cos\gamma\cos\psi\cos\delta + r^2\omega^2\cos^2\delta}}$$

The inertial azimuth can be computed from

$$\cos\psi_i = \frac{v\cos\gamma\cos\psi + r\omega\cos\delta}{\sqrt{v^2\cos^2\gamma + 2vr\omega\cos\gamma\cos\psi\cos\delta + r^2\omega^2\cos^2\delta}}$$

where all coordinates on the right-hand-side of these equations are relative to a rotating Earth.

eci2fpc1.m – convert ECI state vector to flight path coordinates

This MATLAB function convert an ECI state vector consisting of a position and velocity vector to relative flight path coordinates.

The syntax of this MATLAB function is

```
function fpc = eci2fpc1(gast, reci, veci)

% convert inertial state vector to flight path coordinates

% input

%  gast = greenwich apparent sidereal time (radians)
%  reci = inertial position vector (kilometers)
%  veci = inertial velocity vector (kilometers/second)

% output

%  fpc(1) = east longitude (radians)
%  fpc(2) = geocentric declination (radians)
%  fpc(3) = flight path angle (radians)
%  fpc(4) = azimuth (radians)
%  fpc(5) = position magnitude (kilometers)
%  fpc(6) = velocity magnitude (kilometers/second)
```

The transformation of an Earth-centered inertial (ECI) position vector $\mathbf{r}_{ECI}$ to an Earth-centered fixed (ECF) position vector $\mathbf{r}_{ECF}$ is given by the following vector-matrix operation

$$\mathbf{r}_{ECF} = [\mathbf{T}]\mathbf{r}_{ECI}$$

where the elements of the transformation matrix $[\mathbf{T}]$ are given by

$$[\mathbf{T}] = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and θ is the Greenwich apparent sidereal time at the moment of interest. Greenwich sidereal time is given by the following expression:

$$\theta = \theta_{g0} + \omega_e t$$

where θ_{g0} is the Greenwich sidereal time at 0 hours UT, ω_e is the inertial rotation rate of the Earth, and *t* is the elapsed time since 0 hours UT.

Finally, the flight path coordinates are determined from the following set of equations

$$r = \sqrt{r_{ECF}^2 + r_{ECF_y}^2 + r_{ECF}^2}$$

$$v = \sqrt{v_{ECF}^2 + v_{ECF_y}^2 + v_{ECF_z}^2}$$

$$\lambda = \tan^{-1}\left(r_{ECF_y}, r_{ECF_x}\right)$$

$$\delta = \sin^{-1}\left(\frac{r_{ECF_z}}{\left|\mathbf{r}_{ECF}\right|}\right)$$

$$\gamma = \sin^{-1}\left(-\frac{v_{R_z}}{\left|\mathbf{v}_R\right|}\right)$$

$$\psi = \tan^{-1}\left[v_{R_y}, v_{R_x}\right]$$

where

$$\mathbf{v}_R = \begin{bmatrix} -\sin\delta\cos\lambda & -\sin\delta\sin\lambda & \cos\delta \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\delta\cos\lambda & -\cos\delta\sin\lambda & -\sin\delta \end{bmatrix}\mathbf{v}_{ECF}$$

Please note that the two argument inverse tangent calculation is a four quadrant operation.

eci2fpc2.m – convert ECI state vector to flight path coordinates

This MATLAB function computes the relative flight path coordinates by first converting the ECI state vector to ECF coordinates and then to flight path coordinates. The following is the source code that performs these calculations.

```
   % compute ecf state vector

   [recf, vecf] = eci2ecf(gast, reci, veci);

   % compute relative flight path coordinates

   fpc = rv2fpc(recf, vecf);
```

The syntax of this MATLAB function is

```
   function fpc = eci2fpc2(gast, reci, veci)

   % convert inertial state vector to relative flight path coordinates

   % input

   %  gast = greenwich apparent sidereal time (radians)
   %  reci = inertial position vector (kilometers)
   %  veci = inertial velocity vector (kilometers/second)

   % output

   %  fpc(1) = east longitude (radians)
   %  fpc(2) = geocentric declination (radians)
   %  fpc(3) = flight path angle (radians)
   %  fpc(4) = azimuth (radians)
   %  fpc(5) = position magnitude (kilometers)
   %  fpc(6) = velocity magnitude (kilometers/second)
```

The syntax of the MATLAB function that converts the state vector to flight path coordinates is

```
   function fpc = rv2fpc (r, v)

   % transform from cartesian coordinates
   % to flight path coordinates

   % input

   %  r = eci or ecf position vector
   %  v = eci or ecf velocity vector

   % output

   %  fpc(1) = geocentric right ascension or east longitude (radians)
   %  fpc(2) = geocentric declination (radians)
   %  fpc(3) = flight path angle (radians)
   %  fpc(4) = azimuth (radians)
```

```
%  fpc(5) = position magnitude (kilometers)
%  fpc(6) = velocity magnitude (kilometers/second)

% NOTE: if ECI input, output is inertial
%       if ECF input, output is Earth relative
```

This software suite contains a MATLAB script named `demo_eci2fpc` that demonstrates how to interact with both of these functions. The input required by this script is "hardwired" within the source code. The following is the output created by this script.

```
eci2fpc1 function
=================

UTC julian date           2458337.83361944


inertial coordinates
--------------------

        rx (km)                ry (km)                rz (km)                rmag (km)
 -5.86479273288000e+003  -1.78173078828000e+003  -2.15629990858000e+003  +6.49769095119867e+003

        vx (kps)               vy (kps)               vz (kps)               vmag (kps)
 +4.92973713131000e-001  -7.31828662424000e+000  +8.19193017342000e+000  +1.09958202132700e+001


flight path coordinates
-----------------------

east longitude              121.00000000  degrees

geocentric declination      -19.38148629  degrees

flight path angle            -6.20000000  degrees

relative azimuth             38.98298719  degrees

position magnitude         6497.69095120  kilometers

velocity magnitude           10.71074956  kilometers/second


geodetic coordinates
--------------------

geodetic latitude           -19.50000000  degrees

geodetic altitude           121.92000000  kilometers


eci2fpc2 function
=================

UTC julian date           2458337.83361944


inertial coordinates
--------------------

        rx (km)                ry (km)                rz (km)                rmag (km)
 -5.86479273288000e+003  -1.78173078828000e+003  -2.15629990858000e+003  +6.49769095119867e+003

        vx (kps)               vy (kps)               vz (kps)               vmag (kps)
 +4.92973713131000e-001  -7.31828662424000e+000  +8.19193017342000e+000  +1.09958202132700e+001


flight path coordinates
```

```
----------------------

east longitude              121.00000000  degrees

geocentric declination      -19.38148629  degrees

flight path angle            -6.20000000  degrees

relative azimuth             38.98298719  degrees

position magnitude         6497.69095120  kilometers

velocity magnitude           10.71074956  kilometers/second

geodetic coordinates
--------------------

geodetic latitude           -19.50000000  degrees

geodetic altitude           121.92000000  kilometers
```

fpc2eci.m – convert flight path coordinates to ECI state vector

This MATLAB function transforms relative flight path coordinates to an ECI state vector (position and velocity vectors).

The syntax of this MATLAB function is

```
function [reci, veci] = fpc2eci(gst, fpc)

% transform relative flight path coordinates to inertial state vector

% input

%  gst    = greenwich apparent sidereal time (radians)
%  fpc(1) = east longitude (radians)
%  fpc(2) = geocentric declination (radians)
%  fpc(3) = relative flight path angle (radians)
%  fpc(4) = relative azimuth (radians)
%  fpc(5) = position magnitude (kilometers)
%  fpc(6) = relative velocity magnitude (kilometers/second)

% output

%  reci = inertial position vector (kilometers)
%  veci = inertial velocity vector (kilometers/second)
```

This function performs this conversion by first calculating the ECF state vector and then the ECI state vector using the following MATLAB source code:

```
% compute ecf state vector

[recf, vecf] = fpc2ecf(fpc);

% compute eci state vector
```

```
   [reci, veci] = ecf2eci(gst, recf, vecf);
```

LOCAL-VERTICAL-LOCAL- HORIZONTAL COORDINATES

eci2lvlh.m – convert ECI state vector to LVLH coordinates

This MATLAB function converts an ECI state vector (position and velocity vectors) to local-vertical-local-horizontal (LVLH) coordinates. The matrix-vector operation for this transformation is as follows:

$$\mathbf{v}_{LVLH} = \begin{bmatrix} -h_x & -h_y & -h_z \\ (\mathbf{h}\times\mathbf{r})_x & (\mathbf{h}\times\mathbf{r})_y & (\mathbf{h}\times\mathbf{r})_z \\ r_x & r_y & r_z \end{bmatrix} \mathbf{v}_{ECI}$$

The x-axis of the LVLH coordinate system is in the direction of the negative unit angular momentum vector, the y-axis points in the direction of the unit $\mathbf{h}\times\mathbf{r}$ vector, and the z-axis is collinear with the unit position vector.

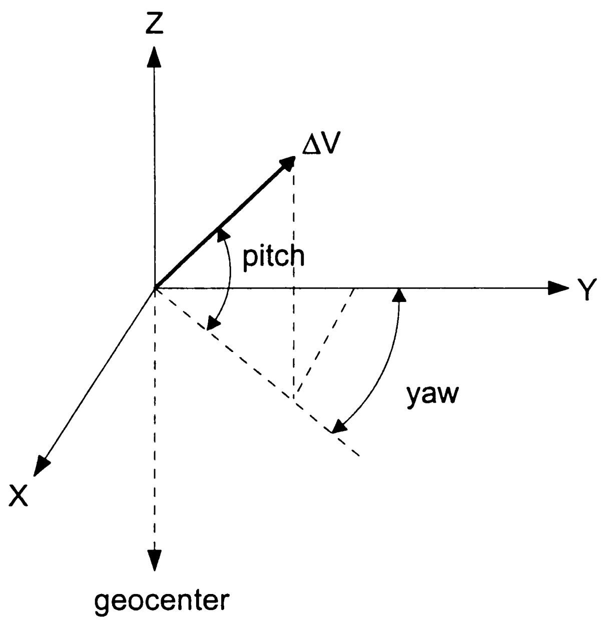
The pitch angle in the LVLH coordinate system is determined from

$$pitch = \sin^{-1}\left(v_{z_{LVLH}}\right)$$

and the yaw angle in this system is given by

$$yaw = \tan^{-1}\left(v_{x_{LVLH}}, v_{y_{LVLH}}\right)$$

The pitch angle is positive above the local horizontal plane and negative below. The yaw angle is measured positive in a direction to the right of the “forward” motion. The order of the rotations is yaw and then pitch. The following diagram illustrates the geometry of these angles.



The syntax of this MATLAB function is

```
function [uplvlh, pitch, yaw] = eci2lvlh (r, v, upeci)

% convert eci unit pointing vector to local
% vertical local horizontal coordinate system

% input

%  r     = eci position vector (kilometers)
%  v     = eci velocity vector (kilometers/second)
%  upeci = eci unit pointing vector (non-dimensional)

% output

%  uplvlh = lvlh unit pointing vector (non-dimensional)
%  pitch  = lvlh pitch angle (radians)
%           (-pi/2 <= pitch <= +pi/2)
%  yaw    = lvlh yaw angle (radians)
%           (0 <= yaw <= 2 pi)
```

RADIAL-TANGENTIAL-NORMAL COORDINATES

eci2rtn1.m – eci to rtn transformation – orbital elements version

This MATLAB function calculates the matrix that transforms coordinates from the Earth-centered-inertial (ECI) coordinate system to the radial-tangential-normal (RTN) coordinate system. This version uses the classical orbital elements to calculate this matrix. The ECI-to-RTN transformation is defined by the following matrix-vector operation:

$$\mathbf{v}_{RTN} = \begin{bmatrix} \cos u \cos\Omega - \sin u \cos i \sin\Omega & \cos u \sin\Omega + \sin u \cos i \cos\Omega & \sin u \sin i \\ -\sin u \cos\Omega - \cos u \cos i \sin\Omega & -\sin u \sin\Omega + \cos u \cos i \cos\Omega & \cos u \sin i \\ \sin i \sin\Omega & -\sin i \cos\Omega & \cos i \end{bmatrix} \mathbf{v}_{ECI}$$

where $u = \omega + \theta$ is the argument of latitude.

The syntax of this MATLAB function is

```
function tm = eci2rtn1(oev)

% eci-to-rtn transformation matrix

% orbital elements version

% input

%  oev(1) = semimajor axis (kilometers)
%  oev(2) = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
%  oev(3) = orbital inclination (radians)
%           (0 <= inclination <= pi)
%  oev(4) = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
%  oev(5) = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
%  oev(6) = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)

% output

%  tm = eci-to-rtn transformation matrix
```

eci2rtn2.m – eci to rtn transformation – state vector version

This MATLAB function calculates the matrix that transforms coordinates from the Earth-centered-inertial (ECI) coordinate system to the radial-tangential-normal (RTN) coordinate system. This version uses the ECI state vector (position and velocity vectors) to calculate this matrix.

The coordinate transformation is a function of the components of the ECI unit position vector, r_x, r_y, r_z, the unit angular momentum vector components h_x, h_y and h_z, and the components of the unit $\mathbf{h} \times \mathbf{r}$ vector as follows:

$$\mathbf{v}_{RTN} = \begin{bmatrix} r_x & r_y & r_z \\ (\mathbf{h} \times \mathbf{r})_x & (\mathbf{h} \times \mathbf{r})_y & (\mathbf{h} \times \mathbf{r})_z \\ h_x & h_y & h_z \end{bmatrix} \mathbf{v}_{ECI}$$

The syntax of this MATLAB function is

```
function tm = eci2rtn2(r, v)

% eci-to-rtn transformation matrix

% state vector version

% input

%  r = eci position vector (kilometers)
%  v = eci velocity vector (kilometers/second)

% output

%  tm = eci-to-rtn transformation matrix
```

EARTH-CENTERED-FIXED COORDINATES

eci2ecf.m – ECI state vector to ECF state vector

This function transforms an ECI state vector (position and velocity vectors) to an ECF state vector. The x-axis of the ECF coordinate system is along the Greenwich or prime meridian.

The transformation of an ECI position vector $\mathbf{r}_{eci}$ to an ECF position vector $\mathbf{r}_{ecf}$ is given by the following vector-matrix operation

$$\mathbf{r}_{ECF} = \left[\mathbf{T}\right]\mathbf{r}_{ECI}$$

where the elements of the transformation matrix $\left[\mathbf{T}\right]$ are given by

$$\left[\mathbf{T}\right] = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is the Greenwich sidereal time at the moment of interest. Greenwich sidereal time is given by the following expression:

$$\theta = \theta_{g0} + \omega_e t$$

where θ_{g0} is the Greenwich sidereal time at 0 hours UT, ω_e is the inertial rotation rate of the Earth, and *t* is the elapsed time since 0 hours UT.

The ECF velocity vector is determined by differentiating this expression

$$\mathbf{v}_{ECF} = \left[\mathbf{T}\right]\dot{\mathbf{r}}_{ECI} + \left[\dot{\mathbf{T}}\right]\mathbf{r}_{ECI} = \left[\mathbf{T}\right]\mathbf{v}_{ECI} + \left[\dot{\mathbf{T}}\right]\mathbf{r}_{ECI}$$

The elements of the $\left[\dot{\mathbf{T}}\right]$ matrix are as follows:

$$\left[\dot{\mathbf{T}}\right]=\begin{bmatrix}-\omega_e\sin\theta & \omega_e\cos\theta & 0\\ -\omega_e\cos\theta & -\omega_e\sin\theta & 0\\ 0 & 0 & 0\end{bmatrix}$$

The syntax of this MATLAB function is

```
function [recf, vecf] = eci2ecf (gast, reci, veci)

% eci-to-ecf transformation

% input

%  gast = apparent sidereal time (radians)
%         (0 <= gast <= 2 pi)
%  reci = position vector (kilometers)
%  veci = velocity vector (kilometers/second)

% output

%  recf = position vector (kilometers)
%  vecf = velocity vector (kilometers/second)
```

ecf2eci.m – ECF state vector to ECI state vector

This MATLAB function transforms an ECF state vector (position and velocity vectors) to an ECI state vector (position and velocity vectors).

The transformation from ECF to ECI coordinates involves the transpose of the ECI-to-ECF transformation matrices described above as follows:

$$\mathbf{r}_{ECI}=\left[\mathbf{T}\right]^T\mathbf{r}_{ECF}$$

$$\mathbf{v}_{ECI}=\left[\mathbf{T}\right]^T\dot{\mathbf{r}}_{ECF}+\left[\dot{\mathbf{T}}\right]^T\mathbf{r}_{ECF}=\left[\mathbf{T}\right]^T\mathbf{v}_{ECF}+\left[\dot{\mathbf{T}}\right]^T\mathbf{r}_{ECF}$$

The syntax of this MATLAB function is

```
function [reci, veci] = ecf2eci (gast, recf, vecf)

% ecf-to-eci transformation

% input

%  gast = apparent sidereal time (radians)
%         (0 <= gast <= 2 pi)
%  recf = position vector (kilometers)
%  vecf = velocity vector (kilometers/second)
% output

%  reci = position vector (kilometers)
%  veci = velocity vector (kilometers/second)
```

lla2eci.m – convert latitude, longitude and altitude to ECI position vector

This MATLAB function converts geodetic latitude, geodetic altitude and geographic longitude to an ECI position vector.

The syntax of this MATLAB function is

```
function reci = lla2eci (gast, lat, long, alt)

% convert geodetic altitude, latitude and longitude to eci position vector

% input

%  gast = apparent sidereal time (radians)
%         (0 <= gast <= 2 pi)
%  lat  = geodetic latitude (radians)
%  long = geographic longitude (radians)
%  alt  = geodetic altitude (kilometers)

% output

%  reci = eci position vector (kilometers)
```

The ECI position vector can be determined from the geocentric distance *r* and the declination and right ascension according to

$$\mathbf{r} = \left\{ \begin{matrix} r\cos\delta\cos\alpha \\ r\cos\delta\sin\alpha \\ r\sin\delta \end{matrix} \right\}$$

The geocentric declination and distance are determined from the geodetic latitude and altitude using the `geodet4.m` function. The right ascension is computed using

$$\alpha = \theta_g + \lambda^E$$

where θ_g is the Greenwich apparent sidereal time and λ^E is the east longitude.

This software suite also includes a MATLAB script named `elong2ra` that can be used to convert an east longitude coordinate to its corresponding right ascension. The following is a typical user interaction with this script along with the results computed by the script. The script output `gst` is the Greenwich apparent sidereal time in degrees at the moment of interest.

```
                program elong2ra

 < convert east longitude to right ascension >

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? **10,21,2008**

please input the universal time
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
? **10,20,30**


please input the east longitude (degrees)
(0 <= east longitude <= 360)
? **100**

                program elong2ra

 < convert east longitude to right ascension >


calendar date       21-Oct-2008

universal time      10:20:30.000


east longitude     100.00000000  degrees

right ascension    285.36225448  degrees

gst                185.36225448  degrees

 right ascension = gst + east longitude
```

HYPERBOLIC COORDINATES

This section describes several functions that can be used to compute important characteristics of hyperbolic orbits.

rv2hyper.m – convert position and velocity to C3, RLA and DLA

This routine can be used to convert an inertial position and velocity vector pair to the equivalent energy of the orbit and the orientation of the asymptote.

The syntax of this MATLAB function is

```
function [c3, rla, dla] = rv2hyper (mu, rsc, vsc)

% convert position and velocity vectors to
% hyperbolic c3, rla and dla

% input
% mu   = gravitational constant (km**3/sec**2)
% rsc  = spacecraft position vector (kilometers)
% vsc  = spacecraft velocity vector (kilometers/second)

% output

% c3  = specific orbital energy (km/sec)**2
% rla = right ascension of asymptote (radians)
% dla = declination of asymptote (radians)
```

The asymptote unit vector of a hyperbolic orbit is given by

$$\hat{\mathbf{s}} = \begin{Bmatrix} \cos\delta_\infty \cos\alpha_\infty \\ \cos\delta_\infty \sin\alpha_\infty \\ \sin\delta_\infty \end{Bmatrix}$$

In this expression, α_∞ is the right ascension of the asymptote (RLA), and δ_∞ is the declination of the asymptote (DLA).

The asymptote unit vector at any trajectory time can be computed from

$$\hat{\mathbf{s}} = \frac{1}{1 + C_3 \dfrac{h^2}{\mu^2}} \left\{ \left(\frac{\sqrt{C_3}}{\mu} \right) \mathbf{h} \times \mathbf{e} - \mathbf{e} \right\} = \frac{1}{1 + C_3 \dfrac{p}{\mu}} \left\{ \left(\frac{\sqrt{C_3}}{\mu} \right) \mathbf{h} \times \mathbf{e} - \mathbf{e} \right\}$$

where **h** and **e** are the angular momentum and orbital eccentricity vectors, respectively. In the second expression, *p* is the semiparameter of the hyperbolic orbit which can be computed from

$$p = a\left(1 - e^2\right)$$

The angular momentum and eccentricity vectors are computed using the following equations;

$$\mathbf{h} = \mathbf{r} \times \mathbf{v}$$

$$\mathbf{e} = \frac{\mathbf{v} \times \mathbf{h}}{\mu} - \frac{\mathbf{r}}{r} = \frac{1}{\mu}\left[\left(v^2 - \frac{\mu}{r} \right) \mathbf{r} - \left(\mathbf{r} \bullet \mathbf{v} \right) \mathbf{v} \right]$$

C3 is the “twice specific” orbital energy which is determined from the position **r** and velocity **v** vectors according to

$$C_3 = \left|\mathbf{v}\right|^2 - \frac{2\mu}{\left|\mathbf{r}\right|}$$

The right ascension and declination of the asymptote can be computed from components of the unit asymptote vector according to

$$\alpha_\infty = \tan^{-1}\left(s_x, s_y\right)$$

$$\delta_\infty = \sin^{-1}\left(s_z\right)$$

The right ascension and declination will be with respect to the coordinate system of the input state vector (EME2000 for example).

orb2hyper.m – convert classical orbital elements to asymptote coordinates

This routine can be used to convert classical orbital elements of a hyperbolic orbit to the corresponding set of asymptote coordinates.

The syntax of this MATLAB function is

```
function [shat, rasc_asy, decl_asy] = orb2hyper(oev)

% this function converts classical orbital elements
% of a hyperbolic orbit to asymptote coordinates

% input

%  oev(1) = semimajor axis (kilometers)
%  oev(2) = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
%  oev(3) = orbital inclination (radians)
%           (0 <= inclination <= pi)
%  oev(4) = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
%  oev(5) = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
%  oev(6) = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)

% output

%  shat     = asymptote unit vector
%  rasc_asy = right ascension of asymptote (radians)
%  decl_asy = declination of asymptote (radians)
```

This software suite includes a MATLAB script named `demo_hyper` that demonstrates how to interact with these two functions. The following is the output created by this script.

```
demo_hyper


rv2hyper function

specific orbital energy         8.787141  (km/sec)**2

asymptote right ascension     349.621260  degrees
```

```
 asymptote declination          -6.697329  degrees


 orb2hyper function

 asymptote right ascension     349.621260  degrees

 asymptote declination          -6.697329  degrees

 asymptote unit vector-x        -0.957048

 asymptote unit vector-y        -0.261889

 asymptote unit vector-z        -0.124391


 state vector

       rx (km)                 ry (km)                 rz (km)                 rmag (km)
 -6.28143245744413e+003  -1.71886519445504e+003  -8.16419427413681e+002  +6.56334000000000e+003

       vx (kps)                vy (kps)                vz (kps)                vmag (kps)
 +3.30316298967422e+000  -9.56155991173246e+000  -5.28351302498913e+000  +1.14127044808508e+001


 classical orbital elements

       sma (km)              eccentricity          inclination (deg)         argper (deg)
 -4.53617896303621e+004  +1.14468873590488e+000  +2.86442848562298e+001  +1.95039684255199e+002

      raan (deg)          true anomaly (deg)          arglat (deg)             period (min)
 +2.03552732637654e+000  +5.08888749034163e-014  +1.95039684255199e+002  +1.66666500000000e+003
```

The asymptote unit vector in terms of the classical orbital elements of a hyperbolic orbit is given by the following

$$\hat{\mathbf{s}} = \begin{Bmatrix} \cos\Omega\cos(\omega+\theta) - \sin\Omega\sin(\omega+\theta)\cos i \\ \sin\Omega\cos(\omega+\theta) + \cos\Omega\sin(\omega+\theta)\cos i \\ \sin(\omega+\theta)\sin i \end{Bmatrix}$$

In this expression, Ω is the right ascension of the ascending node (RAAN), ω is the argument of periapsis and θ is the true anomaly.

The declination of the asymptote (DLA) is given by

$$\delta_\infty = \sin^{-1}\left[\sin(\omega+\theta_\infty)\sin i\right] = \sin^{-1}\left[\sin(u_\infty)\sin i\right]$$

where $u_\infty = \omega + \theta_\infty$ is the argument of latitude of the launch asymptote. In this expression θ_∞ is the true anomaly of the launch hyperbola “at infinity” and is a function of the orbital eccentricity e of the hyperbola according to $\theta_\infty = \cos^{-1}(-1/e)$.

From the following two expressions

$$\sin\left(\alpha_\infty - \Omega\right) = \frac{\tan\delta_\infty}{\tan i}$$

$$\cos\left(\alpha_\infty - \Omega\right) = \frac{\cos u_\infty}{\cos\delta_\infty}$$

the right ascension of the asymptote (RLA) can be determined from

$$\alpha_\infty = \Omega + \tan^{-1}\left(\frac{\tan\delta_\infty}{\tan i}, \frac{\cos u_\infty}{\cos\delta_\infty}\right)$$

where the inverse tangent in this equation is a four quadrant operation.

fbhyper.m – orbital elements of a flyby hyperbola

This MATLAB function can be used to determine the classical orbital elements of a hyperbolic flyby orbit at periapsis of the trajectory. The inputs to this function are the incoming and outgoing v-infinity velocity vectors and the periapsis radius of the flyby hyperbola.

The syntax of this MATLAB function is

```
function oev = fbhyper(mu, vinfi, vinfo, rp)

% input

%  mu    = central body gravitational constant (km**3/sec**2)
%  vinfi = incoming v-infinity vector (kilometers/second)
%  vinfo = outgoing v-infinity vector (kilometers/second)
%  rp    = planet-centered periapsis distance (kilometers)

% output

%  oev(1) = semimajor axis (kilometers)
%  oev(2) = orbital eccentricity (non-dimensional)
%  oev(3) = orbital inclination (radians)
%  oev(4) = argument of perigee (radians)
%  oev(5) = right ascension of ascending node (radians)
%  oev(6) = true anomaly (radians)
```

This software suite includes a MATLAB script named `demo_fbhyper` that demonstrates how to interact with this function. The following is the output created by this script.

```
 incoming v-infinity vector

        vx (kps)                vy (kps)               vz (kps)
  +2.67535684274476e+000  -6.75718806279550e-002  +7.01867693601819e+000
 outgoing v-infinity vector

         vx (kps)                 vy (kps)                 vz (kps)
  +4.41520858875324e+000  +5.19205889928481e+000  +3.15791064678697e+000

 classical orbital elements of the flyby orbit

        sma (km)              eccentricity          inclination (deg)          argper (deg)
  -5.75746179497095e+003  +2.22482982781748e+000  +7.17505314149549e+001  +3.70157442119651e+001

        raan (deg)          true anomaly (deg)           arglat (deg)            period (min)
  +2.38411091477756e+002  +0.00000000000000e+000  +3.70157442119651e+001  +1.66666500000000e+003
```

The semimajor axis can be determined from the gravitational constant of the flyby planet and the scalar magnitude of the flyby v-infinity vector according to the expression

$$a = -\frac{\mu}{v_\infty^2}$$

The orbital eccentricity of the flyby hyperbola is given by

$$e = 1 - \frac{r_p}{a} = 1 + \frac{r_p v_\infty^2}{\mu}$$

A unit vector parallel to the incoming asymptote is given by this next equation:

$$\hat{\mathbf{s}} = \frac{\mathbf{v}_\infty^-}{\left|\mathbf{v}_\infty^-\right|}$$

A unit vector perpendicular to the orbit plane of the flyby hyperbola is calculated from

$$\hat{\mathbf{w}} = \frac{\mathbf{v}_\infty^- \times \mathbf{v}_\infty^+}{\left|\mathbf{v}_\infty^- \times \mathbf{v}_\infty^+\right|}$$

The orbital inclination relative to the ecliptic plane is determined from the z-component of this unit vector as follows:

$$i = \cos^{-1}\left(w_z\right)$$

A unit vector along the z-axis of the ecliptic coordinate system is given by

$$\hat{\mathbf{u}}_z = \left(0,1,1\right)^T$$

From a unit vector in the direction of the ascending node given by

$$\hat{\mathbf{n}} = \hat{\mathbf{u}}_z \times \hat{\mathbf{w}}$$

the right ascension of the ascending node can be computed using a four quadrant inverse tangent function as follows:

$$\Omega = \tan^{-1}\left(n_y, n_x\right)$$

A unit vector in the direction of the periapsis of the flyby hyperbola is given by

$$\hat{\mathbf{p}} = -\cos\theta_\infty \hat{\mathbf{s}} + \sin\theta_\infty \hat{\mathbf{b}}$$

where

$$\cos\theta_\infty = -1/e$$

$$\sin\theta_\infty = \sqrt{1-1/e^2}$$

and $\hat{\mathbf{b}} = \hat{\mathbf{s}} \times \hat{\mathbf{w}}$. In these expressions θ_∞ is the orbital true anomaly at infinity.

The sine and cosine of the argument of periapsis are given by

$$\cos\omega = \hat{\mathbf{n}} \bullet \hat{\mathbf{p}}$$

$$\sin\omega = \sqrt{1-\cos^2\omega}$$

If $p_z < 0$ then $\sin\omega = -\sin\omega$. Finally, the argument of periapsis is determined from a four quadrant inverse tangent function as $\omega = \tan^{-1}\left(\sin\omega, \cos\omega\right)$.

launch.m – orbital elements of a hyperbolic launch trajectory

This MATLAB function can be used to determine the characteristics of an Earth-centered interplanetary departure hyperbola. In this implementation, interplanetary injection is assumed to occur *impulsively* at perigee of the launch hyperbola. The method described here is based on the fundamental characteristics of the B-plane coordinate system.

The syntax of this MATLAB function is

```
function [rhyper, vhyper] = launch(mu, vinf, decl_asy, rasc_asy, rpmag, xinc)

% orbital elements of a launch hyperbola

% input

%  mu       = gravitational constant (km**3/sec**2)
%  vinf     = v-infinity magnitude (kilometers/second)
%  decl_asy = declination of outgoing asymptote (radians)
%  rasc_asy = right ascension of outgoing asymptote (radians)
%  rpmag    = perigee radius of launch hyperbola (kilometers)
%  xinc     = launch hyperbola inclination (radians)
% output

%  rhyper = position vector at periapsis of launch hyperbola (km)
%  vhyper = velocity vector at periapsis of launch hyperbola (km/sec)
```

This software suite includes a MATLAB script named `demo_launch` that demonstrates how to interact with this function. The following is the output created by this script.

```
demo_launch - launch trajectory design


v-infinity magnitude         2964.311219  meters/second

asymptote right ascension     349.621260  degrees

asymptote declination          -6.697329  degrees

perigee altitude              185.200000  kilometers

launch azimuth                 93.000000  degrees

launch site latitude           28.500000  degrees


departure delta-v vector and magnitude

x-component of delta-v       1047.636830  meters/second
y-component of delta-v      -3032.560715  meters/second
z-component of delta-v      -1675.728038  meters/second

delta-v magnitude            3619.672896  meters/second


periapsis state vector of the launch hyperbola

       rx (km)                 ry (km)                 rz (km)                 rmag (km)
 -6.28143245744413e+003  -1.71886519445504e+003  -8.16419427413681e+002  +6.56334000000000e+003

       vx (kps)                vy (kps)                vz (kps)                vmag (kps)
 +3.30316298967422e+000  -9.56155991173246e+000  -5.28351302498913e+000  +1.14127044808508e+001


classical orbital elements of the launch hyperbola

       sma (km)              eccentricity           inclination (deg)         argper (deg)
 -4.53617896303624e+004  +1.14468873590487e+000  +2.86442848562298e+001  +1.95039684255199e+002

      raan (deg)           true anomaly (deg)          arglat (deg)            period (min)
 +2.03552732637651e+000  +0.00000000000000e+000  +1.95039684255199e+002  +1.66666500000000e+003
```

The departure trajectory for interplanetary missions can be determined from the orbital energy C_3, and the right ascension α_∞ and declination δ_∞ of the outgoing asymptote. The perigee radius of the departure hyperbola is specified and the orbital inclination is computed from the user-defined launch azimuth $\sum_L$ and launch site geocentric latitude ϕ_L with the equation

$$i = \cos^{-1}\left(\cos\phi_L \sin\Sigma_L\right)$$

This algorithm is valid for geocentric orbit inclinations that satisfy the following constraint

$$|\,i\,| > |\delta_\infty|$$

The software will display the error message `|inclination| must be > |asymptote declination|` if this condition is not true.

A unit vector in the direction of the departure asymptote is given by

$$\hat{\mathbf{S}} = \begin{Bmatrix} \cos\delta_\infty \cos\alpha_\infty \\ \cos\delta_\infty \sin\alpha_\infty \\ \sin\delta_\infty \end{Bmatrix}$$

where

$\alpha_\infty =$ right ascension of departure asymptote

$\delta_\infty =$ declination of departure asymptote

The T-axis direction of the B-plane coordinate system is determined from the following vector cross product:

$$\hat{\mathbf{T}} = \hat{\mathbf{S}} \times \hat{\mathbf{u}}_z$$

where $\hat{\mathbf{u}}_z = \left[0\;0\;1\right]^T$ is a unit vector perpendicular to the Earth's equator.

The following cross product operation completes the B-plane coordinate system.

$$\hat{\mathbf{R}} = \hat{\mathbf{S}} \times \hat{\mathbf{T}}$$

The B-plane angle is determined from the orbital inclination of the departure hyperbola *i* and the declination of the outgoing asymptote according to

$$\cos\theta = \frac{\cos i}{\cos\delta_\infty}$$

The unit angular momentum vector of the departure hyperbola is given by

$$\hat{\mathbf{h}} = \hat{\mathbf{T}}\sin\theta - \hat{\mathbf{R}}\cos\theta$$

The sine and cosine of the true anomaly at infinity are given by the next two equations

$$\cos\theta_\infty = -\frac{\mu}{r_p V_\infty^2 + \mu}$$

$$\sin\theta_\infty = \sqrt{1-\cos^2\theta_\infty}$$

where $V_\infty = \sqrt{C_3} = V_L - V_p$ is the spacecraft's velocity at infinity, V_L is the heliocentric departure velocity determined from the Lambert solution, V_p is the heliocentric velocity of the departure planet, and r_p is the user-specified perigee radius of the departure hyperbola.

A unit vector in the direction of perigee of the departure hyperbola is determined from

$$\hat{\mathbf{r}}_p = \hat{\mathbf{S}}\cos\theta_\infty - \left(\hat{\mathbf{h}}\times\hat{\mathbf{S}}\right)\sin\theta_\infty$$

The ECI position vector at perigee is

$$\mathbf{r}_p = r_p\,\hat{\mathbf{r}}_p$$

The scalar magnitude of the perigee velocity can be determined from

$$V_p = \sqrt{\frac{2\mu}{r_p} + V_\infty^2}$$

A unit vector aligned with the velocity vector at perigee is

$$\hat{\mathbf{v}}_p = \hat{\mathbf{h}}\times\hat{\mathbf{r}}_p$$

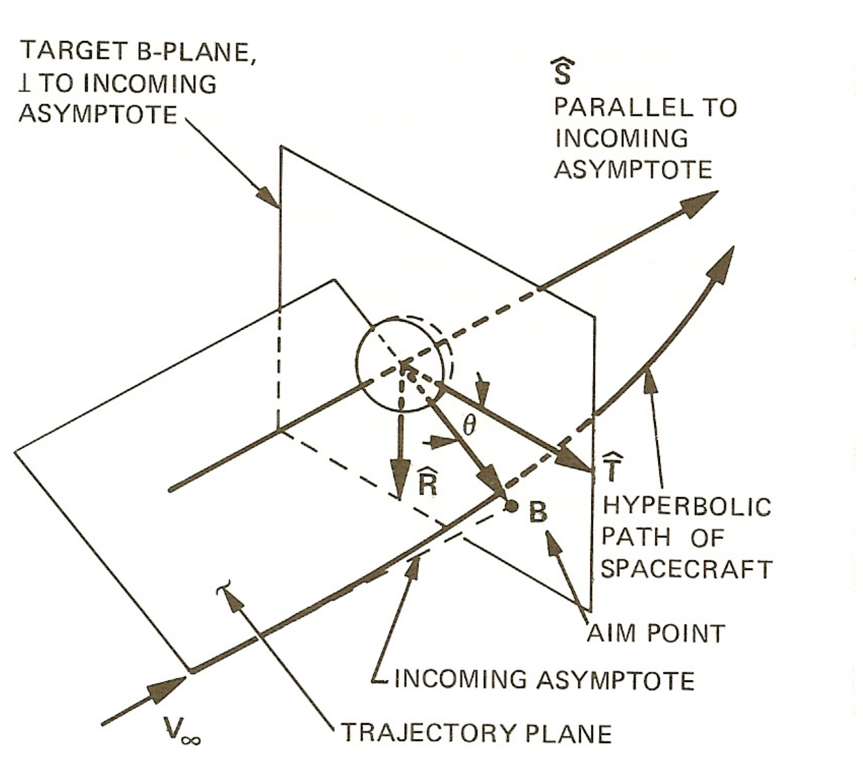
The ECI velocity vector at perigee of the departure hyperbola is given by

$$\mathbf{v}_p = V_p\hat{\mathbf{v}}_p$$

Finally, the classical orbital elements of the departure hyperbola can be determined from the position and velocity vectors at perigee. The injection delta-v vector and magnitude can be determined from the velocity difference between the park orbit and departure hyperbola at the orbital location of the impulsive maneuver.

B-PLANE COORDINATES

The derivation of B-plane coordinates is described in the classic JPL reports, "A Method of Describing Miss Distances for Lunar and Interplanetary Trajectories" and "Some Orbital Elements Useful in Space Trajectory Calculations", both by William Kizner. The following diagram illustrates the fundamental geometry of the B-plane coordinate system.


TARGET B-PLANE, ⊥ TO INCOMING ASYMPTOTE
Ŝ PARALLEL TO INCOMING ASYMPTOTE
θ
R̂
T̂
B
HYPERBOLIC PATH OF SPACECRAFT
AIM POINT
INCOMING ASYMPTOTE
V∞
TRAJECTORY PLANE

The arrival asymptote unit vector $\hat{\mathbf{S}}$ is given by

$$\hat{\mathbf{S}} = \begin{Bmatrix} \cos\delta_\infty \cos\alpha_\infty \\ \cos\delta_\infty \sin\alpha_\infty \\ \sin\delta_\infty \end{Bmatrix}$$

where δ_∞ and α_∞ are the declination and right ascension of the asymptote of the incoming hyperbola.

The following computational steps summarize the calculation of the B-plane vector from a body-centered position vector **r** and velocity vector **v** at closest approach.

angular momentum vector

$$\mathbf{h} = \mathbf{r} \times \mathbf{v}$$

radius rate

$$\dot{r} = \frac{\mathbf{r} \bullet \mathbf{v}}{|\mathbf{r}|}$$

semiparameter

$$p = \frac{h^2}{\mu}$$

semimajor axis

$$a = \frac{r}{\left(2 - \frac{r\,v^2}{\mu}\right)}$$

orbital eccentricity

$$e = \sqrt{1 - p/a}$$

true anomaly

$$\cos\theta = \frac{p - r}{e\,r}$$

$$\sin\theta = \frac{\dot{r}\,h}{e\mu}$$

B-plane magnitude

$$B = \sqrt{p\left|a\right|}$$

fundamental vectors

$$\hat{\mathbf{z}} = \frac{r\,\mathbf{v} - \dot{r}\,\mathbf{r}}{h}$$

$$\hat{\mathbf{p}} = \cos\theta\,\hat{\mathbf{r}} - \sin\theta\,\hat{\mathbf{z}}$$

$$\hat{\mathbf{q}} = \sin\theta\,\hat{\mathbf{r}} + \cos\theta\,\hat{\mathbf{z}}$$

S vector

$$\mathbf{S} = -\frac{a}{\sqrt{a^2 + b^2}}\,\hat{\mathbf{p}} + \frac{b}{\sqrt{a^2 + b^2}}\,\hat{\mathbf{q}}$$

B vector

$$\mathbf{B} = \frac{b^2}{\sqrt{a^2 + b^2}}\,\hat{\mathbf{p}} + \frac{ab}{\sqrt{a^2 + b^2}}\,\hat{\mathbf{q}}$$

T vector

$$\mathbf{T} = \frac{\left(S_y^2, -S_x^2, 0\right)^T}{\sqrt{S_x^2 + S_y^2}}$$

R vector

$$\mathbf{R} = \mathbf{S} \times \mathbf{T} = \left(-S_z T_y,\, S_z T_x,\, S_x T_y - S_y T_x\right)^T$$

rv2bp1.m – convert position and velocity to basic set of B-plane coordinates

This routine can be used to convert an inertial position and velocity vector pair to coordinates in the B-plane coordinate system. The B-plane coordinates computed by this function include the B-plane magnitude, $\mathbf{B \bullet T}$, $\mathbf{B \bullet R}$ and the B-plane angle θ.

Please note that the B-plane coordinates will be with respect to the coordinate system of the input state vector (true-of-date or EME2000 for example).

The syntax of this MATLAB function is

```
function [bmag, bdott, bdotr, theta] = rv2bp1(mu, r, v)

% convert position and velocity vectors to b-plane coordinates

% input

%  mu = gravitational constant (km**3/sec**2)
%  r  = planet-centered position vector (kilometers)
%  v  = planet-centered velocity vector (kilometers/second)

% output

%  bmag  = b-plane magnitude (kilometers)
%  bdt   = b dot t
%  bdr   = b dot r
%  theta = b-plane angle (radians)
```

rv2bp2.m – convert position and velocity to a comprehensive set of B-plane coordinates

This routine can be used to convert an inertial position and velocity vector pair to a comprehensive set of coordinates in the B-plane coordinate system. The B-plane coordinates computed by this function will be with respect to the coordinate system of the input state vector (true-of-date or EME2000 for example).

The syntax of this MATLAB function is

```
function [bplane, rv, tv, ibperr] = rv2bp2(mu, r, v)

% convert body-centered cartesian position
% and velocity vectors to b-plane elements

% input

%  mu = gravitational constant (km**3/sec**2)
%  r  = position vector (kilometers)
```

```
  %  v  = velocity vector (kilometers/second)

  % output

  %  bplane(1)  = hyperbolic speed (kilometers/second)
  %  bplane(2)  = declination of asymptote (radians)
  %  bplane(3)  = right ascension of asymptote (radians)
  %  bplane(4)  = radius magnitude (kilometers)
  %  bplane(5)  = periapsis radius (kilometers)
  %  bplane(6)  = b-plane angle (radians)
  %  bplane(7)  = b dot t
  %  bplane(8)  = b dot r
  %  bplane(9)  = b magnitude (kilometers)
  %  bplane(10) = b-plane vector x-component
  %  bplane(11) = b-plane vector y-component
  %  bplane(12) = b-plane vector z-component
  %  rv         = b-plane r-vector
  %  tv         = b-plane t vector
  %  ibperr     = error flag (1 ==> not hyperbola)
```

This software suite includes a MATLAB script named `demo_bplane` that demonstrates how to interact with these two functions. The following is the output created by this script.

```
 state vector of the hyperbolic orbit

        rx (km)                 ry (km)                 rz (km)                 rmag (km)
 +2.40472583440000e+001  +1.06726117210000e+003  +1.49620163200000e+003  +1.83800000100805e+003

        vx (kps)                vy (kps)                vz (kps)                vmag (kps)
 +4.54874246850000e-002  +2.01881484540000e+000  -1.44077943110000e+000  +2.48063053574948e+000


 classical orbital elements of the hyperbolic orbit

        sma (km)              eccentricity          inclination (deg)         argper (deg)
 -5.98926246356783e+003  +1.30688252722075e+000  +9.00000000006169e+001  +1.25507658933524e+002

       raan (deg)           true anomaly (deg)          arglat (deg)            period (min)
 +2.68709244411305e+002  +2.18348779057465e-007  +1.25507659151872e+002  +1.66666500000000e+003


 b-plane coordinates of the hyperbolic orbit

 rv2bp1 function

 b-magnitude                 5039.322656  kilometers
 b dot r                    -5039.322656
 b dot t                       -0.000000
 b-plane angle                270.000000  degrees


 rv2bp2 function

 b-magnitude                 5039.322656  kilometers
 b dot r                    -5039.322656
 b dot t                       -0.000000
 b-plane angle                270.000000  degrees
 v-infinity                   904.764239  meters/second
 r-periapsis                 1838.000001  kilometers
 decl-asymptote                14.415324  degrees
 rasc-asymptote                88.709244  degrees
```

REFRACTION, NUTATION AND PRECESSION

refract1.m – refraction correction function

This MATLAB function corrects a topocentric elevation angle for atmospheric refraction. The correction to be added to the calculated topocentric elevation angle is as follows:

$$\Delta E = \frac{1.02}{\tan\left(E + \dfrac{10.3}{E + 5.11}\right)}$$

where E is the true topocentric elevation angle in degrees.

The syntax of this MATLAB function is

```
function elev2 = refract(elev1)

% refraction correction

% input

%  elev1 = uncorrected angle (radians)

% output

%  elev2 = angle corrected for refraction (radians)
```

refract2.m – refraction correction function including temperature/pressure effects

This MATLAB function determines the effect of atmospheric refraction using an algorithm that includes the effects of ambient pressure and temperature. The code is based on the numerical method described in “A New Angular Tropospheric Refraction Model”, by A. L. Berman and S. T. Rockwell, JPL Deep Space Network Progress Report 42-24.

The syntax of this MATLAB function is

```
function elev2 = refract2(press, temp, elev1)

% optical refraction correction

% input

%  press = atmospheric pressure (mm of Hg)
%  temp = atmospheric temperature (degrees C)
%  elev1 = uncorrected elevation angle (radians)
% output

%  elev2 = elevation angle corrected
%          for refraction (radians)
```

nod.m – IAU 1980 nutation in longitude and obliquity

This MATLAB function calculates the nutation in longitude and obliquity based on the IAU 1980 theory. This function requires initialization the first time it is called. This can be accomplished by placing the following statement in the main script along with a `global inutate` statement.

```
      inutate = 1;
```

This MATLAB function also requires the comma-separated data file named `nut80.csv`.

The nutation in longitude is determined from a series of the form

$$\Delta\psi = \sum_{i=1}^{n} S_i \sin A_i$$

Likewise, the nutation in obliquity is determined from

$$\Delta\varepsilon = \sum_{i=1}^{n} C_i \cos A_i$$

where

$$A_i = a_i l + b_i l' + c_i F + d_i D + e_i \Omega$$

and l, l', F, D and Ω are fundamental arguments.

The nutation matrix defined by

$$\mathbf{N} = \begin{bmatrix} \cos\Delta\psi & -\sin\Delta\psi\cos\varepsilon_0 & -\sin\Delta\psi\sin\varepsilon_0 \\ \sin\Delta\psi\cos\varepsilon & \cos\Delta\psi\cos\varepsilon\cos\varepsilon_0 + \sin\varepsilon\sin\varepsilon_0 & \cos\Delta\psi\cos\varepsilon\cos\varepsilon_0 - \sin\varepsilon\cos\varepsilon_0 \\ \sin\Delta\psi\sin\varepsilon & \cos\Delta\psi\sin\varepsilon\cos\varepsilon_0 - \cos\varepsilon\sin\varepsilon_0 & \cos\Delta\psi\sin\varepsilon\sin\varepsilon_0 + \cos\varepsilon\cos\varepsilon_0 \end{bmatrix}$$

can be used to transform a mean equinox of date position vector $\mathbf{r}_0$ to a true equinox of date position vector $\mathbf{r}$ as follows:

$$\mathbf{r} = [\mathbf{N}]\mathbf{r}_0$$

In this matrix ε_0 is the mean obliquity of the ecliptic and $\varepsilon = \varepsilon_0 + \Delta\varepsilon$ is the true obliquity.

The nutation matrix can also be expressed as a combination of individual rotations according to

$$\mathbf{N} = \mathbf{R}_1\left(-\varepsilon\right)\mathbf{R}_3\left(-\Delta\psi\right)\mathbf{R}_1\left(+\varepsilon_0\right)$$

The mean obliquity of the ecliptic is calculated from

$$\varepsilon_0 = 23^0 26'21.''448 - 46.''8150T - 0.''00059T^2 + 0.''001813T^3$$

where $T = \left(JD - 2451545.0\right)/36525$ and JD is the Julian Date.

If second-order terms are neglected, a <u>linearized</u> nutation matrix can be calculated from

$$\mathbf{N} = \begin{bmatrix} 1 & -\Delta\psi\cos\varepsilon & -\Delta\psi\sin\varepsilon \\ \Delta\psi\cos\varepsilon & 1 & -\Delta\varepsilon \\ \Delta\psi\sin\varepsilon & \Delta\varepsilon & 1 \end{bmatrix}$$

Mean equinox equatorial rectangular position coordinates can be converted to true equinox coordinates by adding the following corrections to the respective components:

$$\Delta r_x = -\left(r_y\cos\varepsilon + r_z\sin\varepsilon\right)\Delta\psi$$

$$\Delta r_y = r_x\cos\varepsilon\,\Delta\psi - r_z\Delta\varepsilon$$

$$\Delta r_z = r_x\sin\varepsilon\,\Delta\psi + r_y\Delta\varepsilon$$

Finally, the corrections to right ascension and declination are given by

$$\Delta\alpha = \left(\cos\varepsilon + \sin\varepsilon\sin\alpha\tan\delta\right)\Delta\psi - \cos\alpha\tan\delta\Delta\varepsilon$$

$$\Delta\delta = \sin\varepsilon\cos\alpha\Delta\psi + \sin\alpha\Delta\varepsilon$$

The syntax of this MATLAB function is

```
function [dpsi, deps] = nod(jdate)

% this function evaluates the nutation series and returns the
% values for nutation in longitude and nutation in obliquity.
% wahr nutation series for axis b for gilbert & dziewonski earth
% model 1066a. see seidelmann (1982) celestial mechanics 27,
% 79-106. 1980 iau theory of nutation.

% jdate = tdb julian date (in)
% dpsi  = nutation in longitude in arcseconds (out)
% deps  = nutation in obliquity in arcseconds (out)

% NOTE: requires first time initialization via inutate flag
```

The first few terms of the `nut80.csv` data file are as follows

```
 0.,  0.,  0.,  0.,  1., -171996., -174.2,  92025.,  8.9
 0.,  0.,  2., -2.,  2.,  -13187.,   -1.6,   5736., -3.1
 0.,  0.,  2.,  0.,  2.,   -2274.,   -0.2,    977., -0.5
 0.,  0.,  0.,  0.,  2.,    2062.,    0.2,   -895.,  0.5
 0.,  1.,  0.,  0.,  0.,    1426.,   -3.4,     54., -0.1
 1.,  0.,  0.,  0.,  0.,     712.,    0.1,     -7.,  0.0
 0.,  1.,  2., -2.,  2.,    -517.,    1.2,    224., -0.6
 0.,  0.,  2.,  0.,  1.,    -386.,   -0.4,    200.,  0.0
 1.,  0.,  2.,  0.,  2.,    -301.,    0.0,    129., -0.1
 0., -1.,  2., -2.,  2.,     217.,   -0.5,    -95.,  0.3
 1.,  0.,  0., -2.,  0.,    -158.,    0.0,     -1.,  0.0
 0.,  0.,  2., -2.,  1.,     129.,    0.1,    -70.,  0.0
-1.,  0.,  2.,  0.,  2.,     123.,    0.0,    -53.,  0.0
 1.,  0.,  0.,  0.,  1.,      63.,    0.1,    -33.,  0.0
 0.,  0.,  0.,  2.,  0.,      63.,    0.0,     -2.,  0.0
-1.,  0.,  2.,  2.,  2.,     -59.,    0.0,     26.,  0.0
-1.,  0.,  0.,  0.,  1.,     -58.,   -0.1,     32.,  0.0
```

precess.m – precession transformation

This MATLAB function precesses equatorial rectangular coordinates from one epoch to another. The coordinates are referred to the <u>mean</u> equator and equinox of the two respective epochs. This function was ported to MATLAB using the Fortran version of the NOVAS (Naval Observatory Vector Astrometry Subroutines) source code written by G. Kaplan at the U.S. Naval Observatory.

According to J. H. Lieske, “Precession Matrix Based on IAU (1976) System of Astronomical Constants”, *Astronomy and Astrophysics*, 73, 282-284 (1979), the fundamental precession matrix is defined by

$$\mathbf{P} = \begin{bmatrix} \cos z_a \cos\theta_a \cos\zeta_a - \sin z_a \sin\zeta_a & -\cos z_a \cos\theta_a \sin\zeta_a - \sin z_a \cos\zeta_a & -\cos z_a \sin\theta_a \\ \sin z_a \cos\theta_a \cos\zeta_a + \cos z_a \sin\zeta_a & -\sin z_a \cos\theta_a \sin\zeta_a + \cos z_a \cos\zeta_a & -\sin z_a \sin\theta_a \\ \sin\theta_a \cos\zeta_a & -\sin\theta_a \sin\zeta_a & \cos\theta_a \end{bmatrix}$$

The precession transformation of a position vector at epoch 1, $\mathbf{r}_1$, to a position vector at epoch 2, $\mathbf{r}_2$, is determined from

$$\mathbf{r}_2 = \begin{Bmatrix} r_{2_x} \\ r_{2_y} \\ r_{2_z} \end{Bmatrix} = [\mathbf{P}]\mathbf{r}_1 = [\mathbf{P}] \begin{Bmatrix} r_{1_x} \\ r_{1_y} \\ r_{1_z} \end{Bmatrix}$$

The precession angles are given by

$$\zeta_a = \left(2306.2181 + 1.39656T - 0.000139T^2\right)t + \left(0.30188 - 0.000344T\right)t^2 + 0.017998t^3$$
$$z_a = \left(2306.2181 + 1.39656T - 0.000139T^2\right)t + \left(1.09468 + 0.000066T\right)t^2 + 0.018203t^3$$
$$\theta_a = \left(2004.3109 - 0.85330T - 0.000217T^2\right)t + \left(-0.42665 - 0.000217T\right)t^2 - 0.041833t^3$$

where the unit of these angular arguments is arc seconds and the time arguments are

$$T = \left(JED_1 - 2451545\right)/36525$$
$$t = \left(JED_1 - JED_2\right)/36525$$

In these two equations JED_1 is the TDB Julian Date of the first epoch and JED_2 is the TDB Julian Date of the second epoch.

The syntax of this MATLAB function is

```
function pos2 = precess (tjd1, pos1, tjd2)

% this function precesses equatorial rectangular coordinates from
% one epoch to another.  the coordinates are referred to the mean
% equator and equinox of the two respective epochs. see pages 30-34
% of the explanatory supplement to the ae, lieske, et al. (1977)
% astronomy and astrophysics 58, 1-16, and lieske (1979) astronomy
% and astrophysics 73, 282-284.

% input

%  tjd1 = tdb julian date of first epoch

%  pos1 = position vector, geocentric equatorial rectangular
%         coordinates, referred to mean equator and equinox of
%         first epoch
%  tjd2 = tdb julian date of second epoch

% output

%  pos2 = position vector, geocentric equatorial rectangular
%         coordinates, referred to mean equator and equinox of
%         second epoch
```

ecl2eq.m – ecliptic to equatorial transformation matrix

This MATLAB function transforms a vector from the ecliptic coordinate frame to the equatorial coordinate frame. The required transformation is given by

$$\mathbf{r}_{eq} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varepsilon & -\sin\varepsilon \\ 0 & \sin\varepsilon & \cos\varepsilon \end{bmatrix} \mathbf{r}_{ec}$$

where $\mathbf{r}_{ec}$ is the vector in the ecliptic frame, $\mathbf{r}_{eq}$ is the vector in the equatorial frame and ε is the obliquity of the ecliptic which may be either the mean or true value. The transformation of a vector from the equatorial frame to the ecliptic frame involves the transpose of this matrix. The mean obliquity of the ecliptic can be determined from

$$\varepsilon = 23^0 26' 21.''448 - 46.''8150T - 0.''00059T^2 + 0.''001813T^3$$

The syntax of this MATLAB function is

```
function rmatrix = ecl2eq(obliq)

% ecliptic to equatorial transformation matrix

% input

%  obliq = obliquity of the ecliptic (radians)

% output

%  rmatrix = ecliptic-to-equator transformation matrix
```

TRUE-OF-DATE AND MEAN-OF-DATE COORDINATE CONVERSIONS

This section describes two MATLAB functions and a script that demonstrates how to convert between true-of-date and Earth mean equator and equinox of J2000 (EME2000) inertial position and velocity vectors.

eme2tod.m – convert Earth-mean-equator vector to true-of-date vector

This function can be used to transform coordinates from an Earth-mean equator system to the Earth true-of-date system.

The syntax of this MATLAB function is

```
function pos2 = eme2tod (tjd1, tjd2, pos1)

% convert earth-mean-equator (eme) vector
% to true-of-date (tod) vector

% input

%  tjd1 = initial tdb julian date
%  tjd2 = final tdb julian date
%  pos1 = eme vector at tjd1

% output

%  pos2 = tod vector at tjd2
```

tod2eme.m – convert true-of-date vector to Earth-mean-equator vector

This function can be used to transform a position or velocity vector from an Earth true-of-date to the Earth-mean equator system.

The syntax of this MATLAB function is

```
function pos2 = tod2eme (tjd1, tjd2, pos1)

% convert true-of-date (tod) vector
% to earth-mean-equator (eme) vector

% input

%  tjd1 = initial tdb julian date
%  tjd2 = final tdb julian date
%  pos1 = tod vector at tjd1

% output

%  pos2 = eme vector at tjd2
```

This software suite includes a MATLAB script named `demo_eme_tod` that demonstrates how to interact with these functions. This script reads a simple ASCII data file created by the user to perform the appropriate calculations. The following are the contents of a typical data file. Do not change the number of lines of information in this file. Please note the format and units for each item in this type of file.

```
*******************************************************
data file for Matlab script eme_tod.m
converts between true-of-date and eme2000 state vectors
*******************************************************

type of coordinate conversion
(1 = eme2000-to-true-of-date, 2 = true-of-date-to-eme2000)
----------------------------------------------------------
2

UTC calendar date
-----------------
4, 6, 2004

UTC time
--------
7, 51, 28.386009

TAI-UTC (seconds)
-----------------
32.0

x-component of position vector (kilometers)
-------------------------------------------
5094.5147804
```

```
  y-component of position vector (kilometers)
  -------------------------------------------
  6127.3664612

  z-component of position vector (kilometers)
  -------------------------------------------
  6380.3445328

  x-component of velocity vector (kilometers/second)
  --------------------------------------------------
  -4.746088567

  y-component of velocity vector (kilometers/second)
  --------------------------------------------------
  0.786077222

  z-component of velocity vector (kilometers/second)
  --------------------------------------------------
  5.531931288

  gravitational constant (km**3/sec**2)
  -------------------------------------
  398600.4415
```

Here’s the script output for a true-of-date-to-EME2000 conversion example.

```
 true-of-date-to-eme2000 conversion
 ==================================

 UTC epoch        +07h 51m 28.3860s

 TDB Julian date  +2.45310182815476e+006


 true-of-date state vector and orbital elements
 ----------------------------------------------

         rx (km)                 ry (km)                 rz (km)                 rmag (km)
  +5.09451478040000e+003  +6.12736646120000e+003  +6.38034453280000e+003  +1.02082073330621e+004

         vx (kps)                vy (kps)                vz (kps)                vmag (kps)
  -4.74608856700000e+000  +7.86077222000000e-001  +5.53193128800000e+000  +7.33113482756310e+000


         sma (km)              eccentricity           inclination (deg)          argper (deg)
  +1.63705868475222e+004  +4.24975713684234e-001  +6.30975466212788e+001  +2.13967654661943e+000

        raan (deg)           true anomaly (deg)         arglat (deg)             period (min)
  +2.62889141316872e+001  +4.23572145023630e+001  +4.44968910489824e+001  +3.47421029180553e+002


 eme2000 state vector and orbital elements
 -----------------------------------------

         rx (km)                 ry (km)                 rz (km)                 rmag (km)
  +5.10250960001290e+003  +6.12301152004832e+003  +6.37813630004215e+003  +1.02082073330621e+004

         vx (kps)                vy (kps)                vz (kps)                vmag (kps)
  -4.74321959957887e+000  +7.90536600258154e-001  +5.53375619030436e+000  +7.33113482756310e+000


         sma (km)              eccentricity           inclination (deg)          argper (deg)
  +1.63705868475222e+004  +4.24975713684234e-001  +6.31056273943844e+001  +2.11617005988962e+000

        raan (deg)           true anomaly (deg)         arglat (deg)             period (min)
  +2.62480670113999e+001  +4.23572145023630e+001  +4.44733845622526e+001  +3.47421029180552e+002
```

Here's the script output for an EME2000-to-true-of-date state vector conversion.

```
eme2000-to-true-of-date conversion
==================================

UTC epoch        +07h 51m 28.3860s

TDB Julian date  +2.45310182815476e+006


true-of-date state vector and orbital elements
----------------------------------------------

        rx (km)                 ry (km)                rz (km)                rmag (km)
 +5.09451478038716e+003  +6.12736646115167e+003  +6.38034453275785e+003  +1.02082073330004e+004

        vx (kps)                vy (kps)               vz (kps)               vmag (kps)
 -4.74608856702113e+000  +7.86077222001827e-001  +5.53193128799563e+000  +7.33113482757368e+000


        sma (km)              eccentricity          inclination (deg)         argper (deg)
 +1.63705868473088e+004  +4.24975713679050e-001  +6.30975466212560e+001  +2.13967654694322e+000

       raan (deg)         true anomaly (deg)         arglat (deg)             period (min)
 +2.62889141315341e+001  +4.23572145020192e+001  +4.44968910489625e+001  +3.47421029173760e+002

eme2000 state vector and orbital elements
-----------------------------------------

        rx (km)                 ry (km)                rz (km)                rmag (km)
 +5.10250960000000e+003  +6.12301152000000e+003  +6.37813630000000e+003  +1.02082073330004e+004

        vx (kps)                vy (kps)               vz (kps)               vmag (kps)
 -4.74321959960000e+000  +7.90536600260000e-001  +5.53375619030000e+000  +7.33113482757368e+000


        sma (km)              eccentricity          inclination (deg)         argper (deg)
 +1.63705868473088e+004  +4.24975713679050e-001  +6.31056273943615e+001  +2.11617006021336e+000

       raan (deg)         true anomaly (deg)         arglat (deg)             period (min)
 +2.62480670112469e+001  +4.23572145020193e+001  +4.44733845622326e+001  +3.47421029173760e+002
```

FORMATTED DISPLAY OF COORDINATES

This section describes several MATLAB functions that can be used to display coordinates created by other scripts and functions.

oeprint1.m – print six classical orbital elements (orbital period in minutes)

This function prints a formatted display of the six classical orbital elements with the orbital period displayed in minutes.

The syntax of this MATLAB function is

```
function oeprint1(mu, oev)

% print six classical orbital elements
% and orbital period in minutes

% input
% mu     = gravitational constant (km**3/sec**2)
% oev(1)  = semimajor axis (kilometers)
% oev(2)  = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oev(3)  = orbital inclination (radians)
%           (0 <= inclination <= pi)
% oev(4)  = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
% oev(5)  = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
% oev(6)  = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)
```

oeprint2.m – print six classical orbital elements (orbital period in days)

This function prints a formatted display of the six classical orbital elements with the orbital period displayed in days. This function is typically used to display heliocentric orbital elements.

The syntax of this MATLAB function is

```
function oeprint2(mu, oev)

% print six classical orbital elements,
% argument of latitude and orbital period in days

% input

% mu      = gravitational constant (km**3/sec**2)
% oev(1)  = semimajor axis (kilometers)
% oev(2)  = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oev(3)  = orbital inclination (radians)
%           (0 <= inclination <= pi)
% oev(4)  = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
% oev(5)  = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
% oev(6)  = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)
```

oeprint3.m – print complete set of orbital elements

This function prints a formatted display of a complete set of classical orbital elements with the orbital period displayed in minutes.

The syntax of this MATLAB function is

```
function oeprint3(oev)

% print complete set of orbital elements

% input
% oev(1)  = semimajor axis (kilometers)
% oev(2)  = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oev(3)  = orbital inclination (radians)
%           (0 <= inclination <= pi)
% oev(4)  = argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
% oev(5)  = right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
% oev(6)  = true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)
% oev(7)  = orbital period (seconds)
% oev(8)  = argument of latitude (radians)
%           (0 <= argument of latitude <= 2 pi)
% oev(9)  = east longitude of ascending node (radians)
%           (0 <= east longitude <= 2 pi)
% oev(10) = specific orbital energy (kilometer^2/second^2)
% oev(11) = flight path angle (radians)
%           (-0.5 pi <= fpa <= 0.5 pi)
% oev(12) = right ascension (radians)
%           (-2 pi <= right ascension <= 2 pi)
% oev(13) = declination (radians)
%           (-0.5 pi <= declination <= 0.5 pi)
% oev(14) = geodetic latitude of subpoint (radians)
%           (-0.5 pi <= latitude <= 0.5 pi)
% oev(15) = east longitude of subpoint (radians)
%           (-2 pi <= latitude <= 2 pi)
% oev(16) = geodetic altitude (kilometers)
% oev(17) = geocentric radius of perigee (kilometers)
% oev(18) = geocentric radius of apogee (kilometers)
% oev(19) = perigee velocity (kilometers/second)
% oev(20) = apogee velocity (kilometers/second)
% oev(21) = geodetic altitude of perigee (kilometers)
% oev(22) = geodetic altitude of apogee (kilometers)
% oev(23) = geodetic latitude of perigee (radians)
%           (-0.5 pi <= latitude <= 0.5 pi)
% oev(24) = geodetic latitude of apogee (radians)
%           (-0.5 pi <= latitude <= 0.5 pi)
% oev(25) = mean motion (radians/second)
% oev(26) = mean anomaly (radians)
%           (-2 pi <= mean anomaly <= 2 pi)
% oev(27) = eccentric anomaly (radians)
%           (-2 pi <= eccentric anomaly <= 2 pi)
```

svprint.m – formatted state vector screen display

This function prints a formatted display of the components and scalar magnitudes of a state vector (position and velocity vectors and magnitudes).

The syntax of this MATLAB function is

```
function svprint(r, v)

% print position and velocity vectors and magnitudes
% input

%  r = eci position vector (kilometers)
%  v = eci velocity vector (kilometers/second)
```

MOON-CENTERED COORDINATES AND TRANSFORMATIONS

This section describes MATLAB functions and demonstration scripts that can be used to compute important moon-centered (selenocentric) coordinate information and transformations. Functions are provided for computing the orientation of the moon with respect to the Earth mean equator and equinox of J2000 (EME2000) system along with lunar libration angles extracted from a JPL binary ephemeris file.

Most of the information in this section was extracted from JPL D-32296, "Lunar Constants and Models Document" which is available at ssd.jpl.nasa.gov/dat/lunar_cmd_2005_jpl_d32296.pdf. Another useful reference is "Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 2000", *Celestial Mechanics and Dynamical Astronomy*, **82**: 83-110, 2002.

The following figure illustrates the geometry of the EME2000 coordinate system. The origin of this ECI inertial coordinate system is the geocenter and the fundamental plane is the Earth's mean equator. The z-axis of this system is normal to the Earth's mean equator at epoch J2000, the x-axis is parallel to the vernal equinox of the Earth's mean orbit at epoch J2000, and the y-axis completes the right-handed coordinate system. The epoch J2000 is the Julian Date 2451545.0 which corresponds to January 1, 2000, 12 hours ephemeris time.

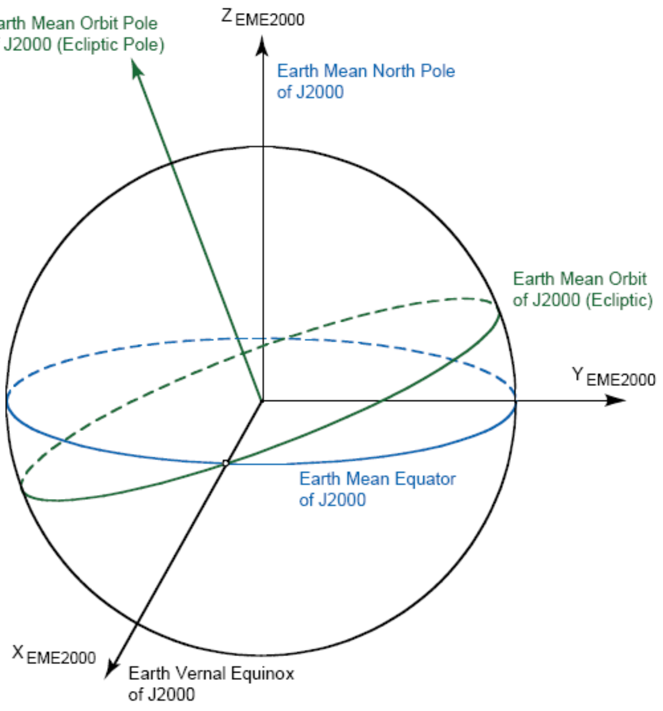

Earth Mean Orbit Pole of J2000 (Ecliptic Pole)
Z EME2000
Earth Mean North Pole of J2000
Earth Mean Orbit of J2000 (Ecliptic)
Y EME2000
Earth Mean Equator of J2000
X EME2000
Earth Vernal Equinox of J2000

Figure 1. Earth mean equator and equinox of J2000 (EME2000) coordinate system

The following figure illustrates the orientation of the lunar mean equator and IAU node of epoch coordinate system relative to the Earth's mean equator and north pole of J2000. The x-axis or Q-vector is formed from the cross product of the Earth's mean pole of J2000 and the Moon's north pole relative to EME2000. The x-axis is aligned with the IAU node of epoch.

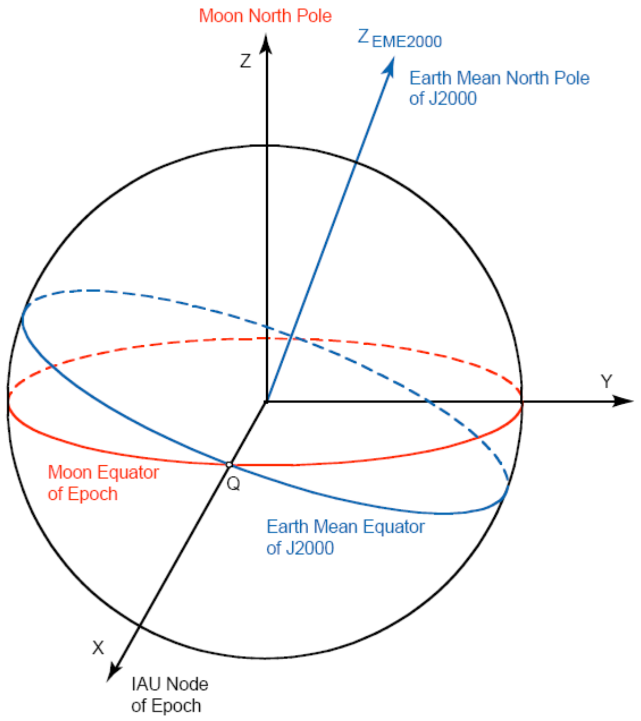


Figure 2. Moon mean equator and IAU node of epoch coordinate system

Lunar orientation with respect to EME2000

The following two equations describe the time evolution of the right ascension and declination of the moon's <u>mean</u> pole, in degrees, with respect to the Earth mean equator and equinox of J2000 (EME2000) coordinate system.

$$\begin{aligned}\alpha_p &= 269.9949 + 0.0031T - 3.8787\sin E1 - 0.1204\sin E2 + 0.0700\sin E3 \\ &\quad -0.0172\sin E4 + 0.0072\sin E6 - 0.0052\sin E10 + 0.0043\sin E13\end{aligned}$$

$$\begin{aligned}\delta_p &= 66.5392 + 0.0130T + 1.5419\cos E1 + 0.0239\cos E2 - 0.0278\cos E3 \\ &\quad +0.0068\cos E4 - 0.0029\cos E6 + 0.0009\cos E7 + 0.0008\cos E10 - 0.0009\cos E13\end{aligned}$$

The equation for the prime meridian of the Moon, in degrees, with respect to the IAU node vector is given by the following expression

$$\begin{aligned}W &= W_p + 3.5610\sin E1 + 0.1208\sin E2 - 0.0642\sin E3 + 0.0158\sin E4 \\ &\quad +0.0252\sin E5 - 0.0066\sin E6 - 0.0047\sin E7 - 0.0046\sin E8 \\ &\quad +0.0028\sin E9 + 0.0052\sin E10 + 0.0040\sin E11 + 0.0019\sin E12 - 0.0044\sin E13\end{aligned}$$

where $W_p = 38.3213 + \dot{W} D - 1.4 \bullet 10^{-12} D^2$ degrees, $\dot{W} = 13.17635815$ degrees/day and $D = JD - 2451545.0$.

In these equations, *T* is the time in Julian centuries given by $T = \left(JD - 2451545.0\right)/36525$ and *JD* is the Barycentric Dynamical Time (TDB) Julian Date.

The trigonometric arguments, in degrees, for these equations are

$$\begin{aligned}
E1 &= 125.045 - 0.0529921d \\
E2 &= 250.089 - 0.1059842d \\
E3 &= 260.008 + 13.0120009d \\
E4 &= 176.625 + 13.3407154d \\
E5 &= 357.529 + 0.9856003d \\
E6 &= 311.589 + 26.4057084d \\
E7 &= 134.963 + 13.0649930d \\
E8 &= 276.617 + 0.3287146d \\
E9 &= 34.226 + 1.7484877d \\
E10 &= 15.134 - 0.1589763d \\
E11 &= 119.743 + 0.0036096d \\
E12 &= 239.961 + 0.1643573d \\
E13 &= 25.053 + 12.9590088d
\end{aligned}$$

where $d = JD - 2451545$ is the number of days since January 1.5, 2000. These equations can also be found in “Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 2000”, *Celestial Mechanics and Dynamical Astronomy*, **82**: 83-110, 2002.

A unit vector in the direction of the pole of the moon can be determined from

$$\hat{\mathbf{p}}_{Moon} = \begin{bmatrix} \cos\alpha_p \cos\delta_p \\ \sin\alpha_p \cos\delta_p \\ \sin\delta_p \end{bmatrix}$$

The unit vector in the x-axis direction of this selenocentric coordinate system is given by

$$\hat{\mathbf{x}} = \hat{\mathbf{z}} \times \hat{\mathbf{p}}_{Moon}$$

where $\hat{\mathbf{z}} = \left[0 \quad 0 \quad 1\right]^T$. The unit vector in the y-axis direction can be determined using

$$\hat{\mathbf{y}} = \hat{\mathbf{p}}_{Moon} \times \hat{\mathbf{x}}$$

Finally, the components of the matrix that transforms coordinates from the EME2000 system to the moon-centered (selenocentric) mean equator and IAU node of epoch system are as follows:

$$\mathbf{M} = \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\mathbf{y}} \\ \hat{\mathbf{p}}_{Moon} \end{bmatrix}$$

moon_angles.m – orientation angles of the moon with respect to EME2000

This MATLAB function computes the orientation angles of the moon with respect to EME2000. It implements the polynomial equations described above.

The syntax of this MATLAB function is

```
function [rasc_pole, decl_pole, rasc_pm] = moon_angles (jdate)

% orientation angles of the moon with respect to EME2000

% input

%  jdate = julian date

% output

%  rasc_pole = right ascension of the lunar pole (radians)
%  decl_pole = declination of the lunar pole (radians)
%  rasc_pm   = right ascension of the lunar prime meridian (radians)
```

mm2000.m – EME2000-to-selenocentric coordinate transformation

This MATLAB function computes the matrix which transforms coordinates between the Earth mean equator and equinox of J2000 (EME2000) and lunar mean equator and IAU node of epoch coordinate systems. It implements the equations described above.

The syntax of this MATLAB function is

```
function tmatrix = mm2000 (xjdate)

% eme2000 to moon mean equator and IAU node of epoch

% input

%  xjdate = julian date

% output

%  tmatrix = transformation matrix
```

lunarlib.m – lunar libration angles

This MATLAB function computes lunar libration angles and rates using information available on modern JPL binary ephemeris files.

The syntax of this MATLAB function is

```
function [phi, theta, psi, phi_dot, theta_dot, psi_dot] = lunarlib(jdate)

% lunar libration angles and rates using a JPL binary ephemeris

% input

%  jdate = TDB julian date

% output

%  phi, theta, psi = libration angles (radians)
%  phi_dot, theta_dot, psi_dot = libration angle rates (radians/day)
```

This software suite includes a MATLAB script named `demo_llib` that demonstrates how to interact with this function. The following is a summary of the results computed by this script.

```
program demo_llib

lunar libration angles and rates

TDB Julian date           2451545.000000

phi             -3.10247126 degrees

theta           24.34245494 degrees

psi             41.17669108 degrees


phi_dot         -0.00000008 degrees/day

theta_dot        0.00000003 degrees/day

psi_dot          0.00015259 degrees/day
```

moon_pa1.m – lunar principal axis coordinate transformation – JPL binary ephemeris

This MATLAB function determines a matrix that can be used to transform coordinates from the lunar mean equator and IAU node of J2000 (which we'll call the moon_j2000 system) to the lunar principal axes (PA) system. The principal axis frame is aligned with the three maximum moments of inertia of the Moon.

The syntax of this MATLAB function is

```
function tmatrix = moon_pa1(jdate)

% transformation matrix from lunar mean equator and IAU node of j2000
% to the lunar principal axes system using JPL binary ephemeris

% input

%  jdate = TDB julian date

% output

%  tmatrix = transformation matrix
```

moon_pa2.m – lunar principal axis coordinate transformation – JPL approximation

This MATLAB function determines a matrix that can be used to transform coordinates from the lunar mean equator and IAU node of J2000 (which we'll call the moon_j2000 system) to the lunar principal axes (PA) system. The principal axis frame is aligned with the three maximum moments of inertia of the Moon.

The syntax of this MATLAB function is

```
function tmatrix = moon_pa2(jdate)

% transformation matrix from lunar mean equator and IAU node of j2000
% to the lunar principal axes system using JPL approximate equations

% input

%  jdate = TDB julian date

% output

%  tmatrix = transformation matrix
```

According to "Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements: 2006", the transformation matrix from the Earth mean equator and equinox of J2000 (EME2000) coordinate system to the moon-centered, body-fixed lunar principal axis (PA) system is given by the following (3-1-3) rotation sequence;

$$[M]_{EME2000}^{PA} = R_z(\psi) R_x(\theta) R_z(\varphi)$$

In this equation, φ is the angle along the International Celestial Reference Frame (ICRF) equator, from the ICRF x-axis to the ascending node of the lunar equator, θ is the inclination of the lunar equator to the ICRF equator, and ψ is the angle along the lunar equator from the node to the lunar prime meridian. These three Euler angles represent the numerically integrated physical librations of the Moon.

The relationship between these angles and the classical IAU orientation angles is

$$\alpha = \varphi - 90^\circ$$

$$\delta = 90^\circ - \theta$$

$$W = \psi$$

The transformation from the moon_j2000 system to the PA system is given by the following matrix multiplication;

$$\left[N\right]_{moon\_j2000}^{PA} = \left[M\right]_{EME2000}^{PA} \left[P\right]_{moon\_j2000}^{EME2000}$$

The numerical components of the <u>constant</u> moon_j2000-to-EME200 transformation matrix are as follows;

```
  0.998496505205088      4.993572939853833E-2   -2.260867140418499E-2
 -5.481540926807404E-2   0.909610125238044      -0.411830900942612
  0.000000000000000      0.412451018902688       0.910979778593430
```

Approximate lunar pole right ascension, declination and prime meridian in the PA system

Page 7 of the JPL document also provides the following "tweaks" to the orientation of the moon in order to approximate the orientation in the PA system.

$$\alpha_{PA} = \alpha_{IAU} + 0.0553\cos W_p + 0.0034\cos\left(W_p + E1\right)$$

$$\delta_{PA} = \delta_{IAU} + 0.0220\sin W_p + 0.0007\sin\left(W_p + E1\right)$$

$$W_{PA} = W_{IAU} + 0.01775 - 0.0507\cos W_p - 0.0034\cos\left(W_p + E1\right)$$

where W_p is the polynomial part of the prime meridian equation given by

$$W_p = 38.3213 + \dot{W}d - 1.4 \bullet 10^{-12} d^2$$

and

$$\begin{aligned}\alpha_{IAU} &= 269.9949 + 0.0031T - 3.8787\sin E1 - 0.1204\sin E2 \\ &+0.0700\sin E3 - 0.0172\sin E4 + 0.0072\sin E6 \\ &-0.0052\sin E10 + 0.0043\sin E13\end{aligned}$$

$$\begin{aligned}\delta_{IAU} = 66.5392 + 0.0130T + 1.5419\cos E1 + 0.0239\cos E2 \\ -0.0278\cos E3 + 0.0068\cos E4 - 0.0029\cos E6 \\ +0.0009\cos E7 + 0.0008\cos E10 - 0.0009\cos E13\end{aligned}$$

$$\begin{aligned}W_{IAU} = W_p + 3.5610\sin E1 + 0.1208\sin E2 + \\ -0.0642\sin E3 + 0.0158\sin E4 + 0.0252\cos E5 \\ -0.0066\sin E6 - 0.0047\sin E7 - 0.0046\cos E8 \\ +0.0028\sin E9 + 0.0052\sin E10 + 0.0040\sin E11 \\ +0.0019\sin E12 - 0.0044\sin E13\end{aligned}$$

moon_me2pa.m – lunar Mean Earth/polar axis to principal axis coordinate transformation

This MATLAB function determines a matrix that can be used to transform coordinates from the lunar Mean Earth/polar axis (ME) to the lunar principal axes (PA) system. The principal axis frame is aligned with the three maximum moments of inertia of the Moon.

The syntax of this MATLAB function is

```
function tmatrix = moon_me2pa

% transformation matrix from the lunar Mean Earth/polar axis (ME)
% system to the lunar principal axes (PA) system

% output

%  tmatrix = me-to-pa transformation matrix
```

According to JPL D-32296, "Lunar Constants and Models Document" and the IAU 2000 resolutions, the constant transformation matrix from the lunar Mean Earth/polar axis (ME) system to the lunar Principal axis (PA) system is given by the following (1-2-3) rotation sequence;

$$[PA] = R_z\left(63.8986''\right) R_y\left(79.0768''\right) R_x\left(0.1462''\right)[ME]$$

The transformation matrix from the PA system to the ME system is given by

$$R_x\left(-0.1462''\right) R_y\left(-79.0768''\right) R_z\left(-63.8986''\right)$$

The numerical components of the ME-to-PA transformation matrix are as follows;

```
  0.999999878527094       3.097894216177013E-004 -3.833748976184077E-004
 -3.097891271165531E-004  0.999999952015005       8.275630251118771E-007
  3.833751355924360E-004 -7.087975496937868E-007  0.999999926511499
```

The PA-to-ME transformation is the transpose of this matrix.

This software suite includes a MATLAB script named `demo_moon` that demonstrates how to interact with several of these coordinate transformation functions. The following is a summary of the results computed by this script.

```
program demo_moon

TDB Julian date          2451545.000000

orientation angles of the moon with respect to EME2000

rasc_pole          266.85773344 degrees

decl_pole           65.64110275 degrees

rasc_pm             41.19526398 degrees


eme2000 to moon mean equator and IAU node
of epoch transformation matrix

   +9.98496505205088e-001  -5.48154092680678e-002  +0.00000000000000e+000
   +4.99357293985326e-002  +9.09610125238044e-001  +4.12451018902689e-001
   -2.26086714041825e-002  -4.11830900942612e-001  +9.10979778593429e-001


lunar mean equator and IAU node of J2000 (moon_j2000)
to lunar principal axis (PA) transformation matrix

moon_pa1 function

   +7.52265999003059e-001  +6.58859395564263e-001  -4.04500463000584e-004
   -6.58859457533997e-001  +7.52266052983559e-001  -2.73229941726294e-005
   +2.86289955305899e-004  +2.87063115131547e-004  +9.99999917816412e-001

moon_pa2 function

   +7.52264777076062e-001  +6.58860807363059e-001  -3.76419448610194e-004
   -6.58860851635045e-001  +7.52264832430686e-001  +8.41278651081412e-006
   +2.88709968745162e-004  +2.41679395513839e-004  +9.99999929118810e-001


lunar mean Earth/polar axis (ME) to lunar
principal axis (PA) transformation matrix

   +9.99999878527094e-001  +3.09789421617701e-004  -3.83374897618408e-004
   -3.09789127116553e-004  +9.99999952015005e-001  +8.27563025111877e-007
   +3.83375135592436e-004  -7.08797549693787e-007  +9.99999926511499e-001
```

MARS-CENTERED TRANSFORMATIONS

mme2000.m – geocentric-to-areocentric coordinate transformation

This function creates a transformation matrix that can be used to convert coordinates from the EME2000 coordinate system to the Mars mean equator and IAU node of epoch system. The equations implemented in this function can be found in “Report of the IAU/IAG Working Group on Cartographic Coordinates and Rotational Elements: 2006”.

Additional information about Mars-centered coordinate systems can be found in JPL D-3444, “Mars Observer Planetary Constants and Models”, JPL IOM 312.B/015-99, “Update to Mars Coordinate Frame Definitions” and JPL IOM 343B-2006-004, “MSL Update to Mars Coordinate Frame Definitions”.

The syntax of this MATLAB function is

```
function tmatrix = mme2000 (jdate)

% eme2000-to-mars-mean-equator and
% IAU-vector of epoch transformation

% input

%  xjdate = julian date

% output

%  tmatrix = transformation matrix
```

A unit vector in the direction of the pole of Mars can be determined from

$$\hat{\mathbf{p}}_{Mars} = \begin{bmatrix} \cos\alpha_p \cos\delta_p \\ \sin\alpha_p \cos\delta_p \\ \sin\delta_p \end{bmatrix}$$

The IAU 2000 right ascension and declination of the pole of Mars in the EME2000 coordinate system are given by the following expressions

$$\alpha_p = 317.68143 - 0.1061T$$

$$\delta_p = 52.88650 - 0.0609T$$

where T is the time in Julian centuries given by $T = (JD - 2451545.0)/36525$ and JD is the TDB Julian Date.

The unit vector in the direction of the *IAU-defined* x-axis is computed from

$$\hat{\mathbf{x}} = \hat{\mathbf{p}}_{J2000} \times \hat{\mathbf{p}}_{Mars}$$

where $\hat{\mathbf{p}}_{J2000} = [0\;0\;1]^T$ is unit vector in the direction of the pole of the J2000 coordinate system.

The unit vector in the y-axis direction of this coordinate system is

$$\hat{\mathbf{y}} = \hat{\mathbf{p}}_{Mars} \times \hat{\mathbf{x}}$$

Finally, the components of the matrix that transforms coordinates from the EME2000 system to the Mars-centered mean equator and IAU node of epoch system are as follows:

$$\mathbf{M} = \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\mathbf{y}} \\ \hat{\mathbf{p}}_{Mars} \end{bmatrix}$$